"SPACECRAFT IN MINIATURE": A TOOL FOR THE ACQUISITION OF MENTAL REPRESENTATIONS OF LARGE, COMPLEX 3-D VIRTUAL ENVIRONMENTS

by

JESSICA J. MÁRQUEZ

B.S.E Mechanical Engineering
Princeton University, 1999

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICS AND ASTRONAUTICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2002

Author: [illegible]
Jessica J. Márquez
Department of Aeronautics and Astronautics
January 25, 2002

Certified by:
Charles M. Oman
Director, Man-Vehicle Laboratory
Department of Aeronautics and Astronautics
Thesis Supervisor

Accepted by: [illegible]
Wallace E. Vander Velde
Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

"SPACECRAFT IN MINIATURE": A TOOL FOR THE ACQUISITION OF MENTAL REPRESENTATIONS OF LARGE, COMPLEX 3-D VIRTUAL ENVIRONMENTS

by

Jessica J. Márquez

Submitted to the Department of Aeronautics and Astronautics
on January 25, 2002 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Aeronautics and Astronautics

ABSTRACT

"Spacecraft in Miniature" (SIM) is a navigational training tool for use in large virtual environments (VE) such as space station simulations where six-degree-of-freedom movement is possible. SIM extends the "Worlds In Miniature" (WIM) terrestrial navigation tool concept of Pausch, et al, to three dimensions. It is designed to facilitate acquisition of a mental representation of the environment by providing the user with a miniature 3D model of the VE that includes an avatar representation of user. It also allows users to substitute model movements for potentially disorienting head rotations in the full-scale virtual environment. To date, there have been no quantitative studies of navigation performance after training with WIM-like tools. We set out to show that after training in a virtual environment using SIM, users acquire a better mental representation when required to learn a complex 3D environment than a control group who only could view the virtual environment directly, by translating and rotating within the station without the miniature model. We hypothesized that SIM users, like terrestrial map learners, would have better survey knowledge, and superior understanding of where modules were located and how they were oriented. They should also be more accurate and quicker to respond to locations and orientations of modules when their own orientation is changed, implying an orientation-free mental representation. On the other hand, we expected the control subjects would have better landmark and route knowledge (describing routes based on landmarks) than SIM users, since their training compelled them to rely directly on visual cues in the local environment.

Ten of the 14 SIM subjects and nine of the 12 control subjects achieved satisfactory performance. Of these, the SIM subjects had significantly better survey knowledge than Control subjects in terms of their ability to point to unseen landmarks. They were approximately twice as accurate and twice as fast than the Control group. SIM subjects performed especially well for targets in the forward direction and surpassed Control subjects in the tasks when their initial body roll angle was changed. Unexpectedly, the two groups did not differ significantly in their answers to most of the route description questions, which measured landmark and route knowledge, possibly because SIM tool allowed the subjects to also utilize local visual cues. SIM subjects were significantly more likely to correctly answer a question that demonstrated understanding of relationships between modules' vertical upright. Results suggest SIM is a promising tool for space station training.

Supported by Grant NAG9-1004 from NASA Johnson Space Center Space Human Factors Engineering program.
Thesis Supervisor: Charles M. Oman
Title: Director of the Man-Vehicle Laboratory

ACKNOWLEDGEMENTS

To the enduring stars that constantly remind me why I keep doing this.

I'd like to thank:

Dr. Charles Oman, who has taught me what being a researcher is all about, and for giving me the opportunity to work in this amazing project;

Dr. Alan Natapoff, for all his advice and for sitting with me for many hours teaching me all about data analysis;

Dr. Andy Liu, who endured my hardware and software complaining and was always willing to aid me in whichever way possible;

Dr. Andy Beall, for all his help with the programming aspect of my project;

My wonderful UROP'ers, Rick Sheridan and Jen Underwood, who constructed the avatar and stations;

(Future Dr.) Tom McGuire, who spent innumerable hours wearing the HMD and running all my pilots and keeping me sane these last few months;

All my friends (the Guinea Pigs), who volunteered most willingly to be unpaid subjects;

My family, who always support me in all I do and made me the person I am today;

Natalie Audage, for always being the excellent friend she is;

Jake Vachon, for just being there for me since the beginning;

And foremost, God, who had gives me the strength to keep going.

Words can't express my gratitude to all of you. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

We have built a unique place in the sky that is unlike any other on Earth, where you can walk on the walls, eat upside-down, where fire can fly and there is only one exit door. The International Space Station (ISS) Alpha is that structure, and it provides a home and a workplace for a diverse group of scientists. In order to provide safety in this new environment, the astronauts sent there must be as prepared as possible to face the unexpected.

The problem that we focus on is orientation, or rather, disorientation in space. Inside a station or a shuttle, the visual surroundings do not follow the rules of structure and orientation that we are accustomed to on Earth, and in microgravity, we no longer have the cue of Earth's gravity to determine "down". With these two important cues disrupted, the astronaut's interpretation of orientation becomes erratic, and he/she must rely more on certain visual and proprioceptive cues and their internal representation of self-orientation (Lathan and Clement 1997; Richards, Clark et al. 2001).

Even though astronauts adapt to the new sensory inputs of space, spatial disorientation still occurs in flight, be it in the U.S. shuttle or in one of the stations that have orbited (Skylab, Mir). It has been reported from experiences inside Mir space station:

> "Even by the end of the flight, this crewmember had no mental survey knowledge that included all the modules. He said he could not have pointed to places in the other modules from base block, or vice versa. He never did get the big picture from inside and felt he had a better understanding of Mir from the outside in."(Richards, Clark et al. 2001)

Some crewmembers found navigation hard or even impossible without the constant view of a landmark.

> "I stop at a node to look to see where the … were. So you'd have to look 90 degrees, and in that process, you know you'd stop yourself, you'd come around and you'd stop. You'd change your orientation a little bit, and I wouldn't remember which direction I was going and I'd have to re-orient and say, ok where's the landmark?"(Richards, Clark et al. 2001)

What are the visual cues used for orientation and navigation? What is a person to expect if they were to go visit ISS? On Earth, our visual scene has a natural "upright" determined by our own experiences and the physical effects of gravity: our feet are on the ground, shelves are on the walls, and lamps hang from the ceiling. In space, zero-g not only prevents us from consistently defining an "upright" but, in addition, the ISS environment itself defies our visual expectations, and thus a definition must be imposed. Currently, U.S. NASA engineers define "down" within ISS as pointing towards Earth (NASA 1998), but the solution is not as simple as that.

In microgravity, the "vertical upright" (see Section 2.1), the upright direction that is attributed to a module is defined by what one sees. Unfortunately, the structure and symmetry of the modules makes it difficult to provide consistent visual cues. Physical space in ISS is precious and must be optimized by utilizing all surfaces of the modules. While "walls" define one visual vertical, ceilings and floors define a second, thus there are two different directions that could appear to be upright within a module (dual visual verticals). Dual visual verticals increase the chances of becoming disoriented. Engineers are aware of these issues and are trying to insert visual clues that will help astronauts define a single upright within a module. For example, writing provides a strong visual cue for orientation. A module's labeling, therefore, would be consistent with its vertical upright if it were all orientated in the same direction. However, even if the entire module appears to have a single vertical upright, an astronaut could still be confused by yet a stronger visual cue: e.g. the orientation of a crewmember who could be working or navigating upside down. When this occurs, there might be a visual reorientation illusion (VRI) in which one's perceived orientation relative to the environment changes. In this case, one feels inverted and believes oneself to be, suddenly, upside down (Oman, Lichtenberg et al. 1986; Oman, Young et al. 1988).

If ISS is built as planned, it will consist of up to five nodes and ten to twelve modules, depending on how many Russian docking and stowage modules are launched (NASA 1998) (Figure 1.1). Each module and node built by the U.S., European, or Japanese space agencies will be colored to give orientation cues to the crewmembers (NASA 1996). Each node (Figure 1.2) has a rounded exterior, and a rectangular interior shape with six apertures, one on each face. There is a single point in each node where there is a line of sight for every direction, i.e. into every module connected to that node. Currently, the aft end of Node 1, the one that leads to the Russian Modules, is colored light salmon and its forward end, the one that leads to the U.S. Lab, is off-white. The signs on Node 1 hatches are aligned so that they can be read when "upright" within this node (feet towards nadir). NASA designed a coloring scheme for the hatches of the modules so that every hatch that leads to the center of the station (Node 1) is to be colored and the opposite hatches are off-white. U.S. engineers are proposing that all U.S. built modules and nodes, plus the European and Japanese modules be consistent with this coloring scheme (Johnson_Engineering 2000). The Russian modules at the other end of the station have a different coloring, with dark floors and light ceilings, as established by color scheme in Mir, and their own coordinate system (NASA 1998).

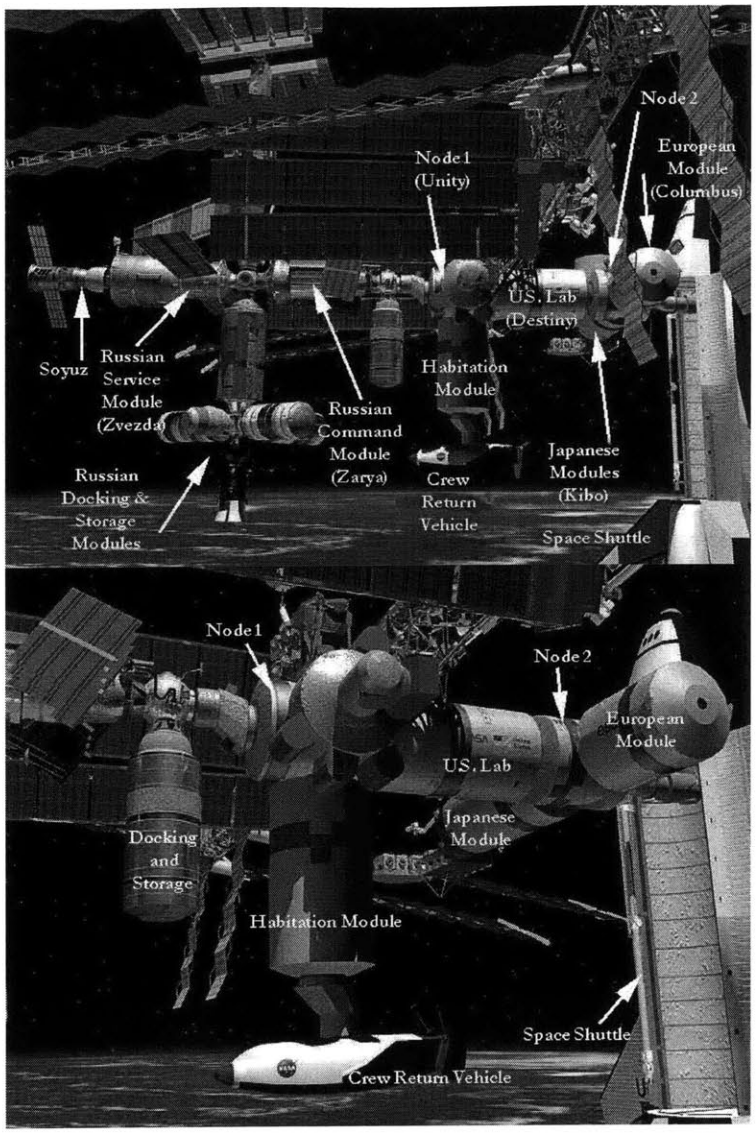


Figure 1.1 International Space Station at Assembly Complete Side Views

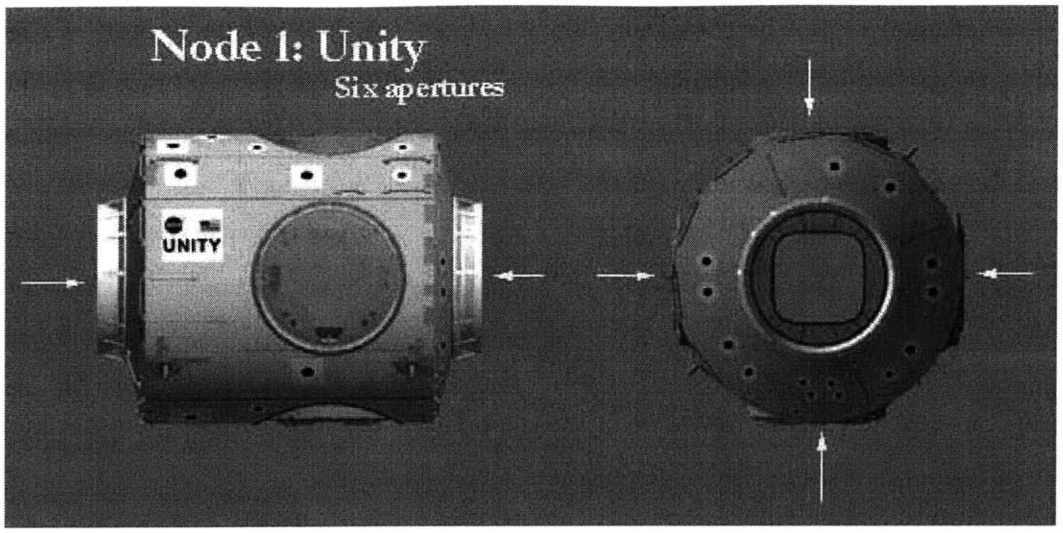


Figure 1.2 Node 1 (Unity) with Six Apertures

As envisioned, ISS is to house six crewmembers and possibly visitors when the shuttle is docked with it. Each person on board is assigned an escape vehicle, which could be the Russian Soyuz, the U.S. Crew Return Vehicle, or the shuttle itself. In an emergency, crewmembers must respond swiftly, either to go to a module with problems or to their assigned return vehicle. It is essential for every person on aboard to be able to react in the proper manner.

Getting from one place to another in microgravity is unlike any ordinary navigation encountered on Earth. In space, one can navigate in any posture, not necessarily just upright. This type of navigation is referred to as navigation with 6 degrees of freedom (6 DOF), i.e. translation in x, y and z coordinates and also rotations about each axis (pitch, yaw, and roll). Navigation in space can further be confounded by the inability to orient oneself spatially with the surrounding environment. Crewmembers on MIR found it particularly difficult to navigate through a node and arrive at a module whose visual vertical upright was inverted with respect to the module they had come from. Beyond the challenges posed by nodes and orientations, crewmembers report being strongly dependent on visual cues as aids to navigate. Emergencies can occur however that obscure vision and thus, navigation cues. For example, during a fire on MIR, "... the smoke (was) so dense that I could not count the fingers in front of my face... I left to get a third fire extinguisher... accidentally bumped into a platform holding a laptop computer." (Richards, Clark et al. 2001)

While disorientation cannot be completely eliminated, we can find ways to improve 6 degrees of freedom navigation under disorientating conditions, e.g. an episode of visual reorientation illusion or an emergency. In order to safeguard astronauts, we believe they should be trained pre-flight in order to reduce their navigation errors during flight.

Our research group studies intra-vehicular virtual reality (VR) training that would decrease the spatial disorientation that leads to navigation errors. The training focuses on helping astronauts develop the best mental representation of the space station they are to encounter. Using it astronauts can explore virtually the inside of the station in its full configuration, and gain experience with the different orientations they might encounter while in station.

This thesis explores two objectives that will support this pre-flight training: the creation of a tool that facilitates intra-vehicular VR training, and the development of an understanding about how humans navigate in an environment where subjects are not always "upright", e.g. 6 degrees of freedom. While trying to validate that tool, "Spacecraft-in-Miniature", we have learned more about what is difficult about wayfinding in microgravity and the type of strategies necessary to navigate successfully in it.

2 BACKGROUND

2.1 ORIENTATION AND VISUAL VERTICALS

On Earth, static perception of orientation with respect to the subjective vertical is thought to be determined by four principal cues: the direction of the gravitational stimulus, a person's headward or footward gravireceptor bias, a person's "idiotropic" tendency to perceive the visual vertical aligned with the body axis (Mittelstaedt 1983; Mittelstaedt 1996), and visual characteristics of the surrounding environment. There are least four visual scene characteristics that define the perceptual vertical on Earth. First, there is the orientation of axes of symmetry of stationary objects and surrounding surfaces. For example, self-tilt can be induced if observing a tilted square against a dark surround. Howard (1982) refers to these scene symmetry cues as "frame" cues (Howard 1982). Second, there is the orientation of "intrinsically polarized objects," familiar objects that are consistently seen in a specific orientation with respect to gravity. These objects (for example, people, tables, sky, earth) have an associated "top" and "bottom" that is recognizable without reference to the surrounding visual environment, and thus provide important cues as to the direction of "up" and "down". Third, there is the position of "extrinsically polarized" objects, defined by the physical relationship one object has with another object and to gravity (for example, objects which are placed "on" or "hang from" other objects). Howard and Hu (2001) have shown that compelling frame and polarity cues can reorient the direction of the subjective vertical by 90 or even 180 degrees with respect to gravity (Howard and Hu 2001). Finally, visual verticals may also be defined through the use of lighting and color. Studies by Howard, Bergstrom et al., and others suggest that humans interpret shading of an object assuming the light source is gravitationally above (Howard, Bergstrom et al. 1990). Ross and Crickman reported divers and pilots use differential brightness for orientation (Ross, Crickman et al. 1969), and Barbour and Coss studies suggest that supine subjects are most likely to feel upright when the upper visual field is illuminated more than the lower visual field (Barbour and Coss 1988).

In space, however, the subjective vertical no longer is influenced by the gravitational stimulus, though the body's gravireceptor bias and idiotropic tendencies remain. Instead, the subjective vertical aligns itself to either the visual vertical or the resultant of the idiotropic vector and the gravireceptor bias vector. For example, an astronaut's subjective vertical aligned itself to the visual vertical of a slanted overhead rack in one of the Spacelab modules; after his work was completed, he felt that this rack was upright and the surface below him was slanted. On the other hand, an astronaut's subjective vertical aligned to the idiotropic and gravireceptor bias vectors while being visually "upside down" could explain visual reorientation illusions (VRIs) (Oman 2000). Spacecrafts generally have lights mounted on a "ceiling" and some (e.g. the Russian Mir station) have modules with dark floors and light ceilings. However, brightness and color cues alone are clearly not sufficient to prevent disorientation, since visual reorientation illusions are commonly reported.

2.2 SPATIAL REPRESENTATIONS

Spatial knowledge, as defined by Golledge, Dougherty, and Bell (1995), consists of three spatial abilities: visualization, orientation, and spatial relations. Spatial visualization is defined as the capability to mentally rotate or twist previous individual visual stimuli (e.g. a place, an object, a building). Spatial orientation is being able to identify a configuration of stimuli from different perspectives. Lastly, spatial relations is the synthesis of many abilities, including estimating distances and angles between landmarks, recalling ordered sequences of cues along a route, and categorizing information into "spatial units" (e.g. nodes and regions) (Golledge, Dougherty et al. 1995). In 1975, Siegel and White proposed that spatial knowledge is acquired in three sequential stages: first, establishing landmarks; second, developing routes that connect the landmarks; and finally, integrating these to form survey or configurational knowledge.

Mental spatial representation of an environment reflects one's spatial knowledge of that environment. Cognitive mental maps refer to a person's spatial representation that integrates and summarizes spatial information. Cognitive maps represent the environment as the mapper believes it to be, and its veracity and complexity depends on the mapper's levels of spatial abilities.

At the most basic level, a mental association of objects with a spatial framework constitutes a mental representation (Franklin and Tversky 1990; Bryant, Lanca et al. 1995; Bryant and Tversky 1999). A spatial framework is a conceptual representation of a local environment that can be defined by the spatial structure as related to a different coordinate system (i.e. our bodies or the world at large). We often use terms such as left or right, in front or behind to describe relationships with respect to our bodies. Gravity is omnipresent in our normal lives, so we also naturally use terms such as above or below to describe spatial relationships. McNamara takes this idea a step further and describes a hierarchical spatial representation as one where "different regions of environment are stored in different branches of a graph-theoretic tree… more detail at the lower levels of the hierarchy"(McNamara 1986). This means that a person would divide, arrange and order knowledge to understand spatial relationships. For example, a person learning where the city of Cambridge could develop a hierarchical mental representation: a country (U.S.) is made up of states (Massachusetts), which have cities (Cambridge). One could envision learning a new environment like the space station as a hierarchical spatial framework: a station, with nodes, lead to modules each of which has objects attached to it.

More complex mental representations encompass landmark, route and/or survey knowledge. Landmark knowledge reflects mental representations that are based solely on the recognition of landmarks, while route knowledge is the mastery of the ordered sequence of legs and turns required to get from one landmark to another. However, the latter does not imply understanding relative positions of the landmarks, i.e. configurational knowledge. Going beyond route knowledge, survey knowledge requires the understanding of Euclidean space (i.e. how far places are from each other) recognizing short cuts, and identifying the environment from different points of view. It is now recognized that many people rely on landmark and route knowledge in unfamiliar environments, and that not everyone is able to acquire survey knowledge of complex environments. Even those who do have survey knowledge often fall back on route or landmark knowledge either out of convenience, or sometimes out of necessity, if their survey knowledge is incomplete.

Each type of knowledge has its corresponding experimental measure. For route knowledge, dependent measures include route retracing, route distance estimation, and retrospective landmark recognition and sequencing tasks(O'Neill 1992) (for example, O'Neill 1992 and Golledge et al. 1995). Examples of measures of configurational knowledge are the ability to perform search tasks, point home or point at known objects (for example, Moeser 1988 and Evans and Pezdek 1980);

also, the capacity to estimate Euclidean distances, and to draw or physically reconstruct a map of the environment. These have been the dependent measures in research on spatial knowledge acquisition, and the independent variables have been the method in which the knowledge is obtained.

Research has focused on how different levels of knowledge are obtained and stored. Two ways of acquiring spatial knowledge are direct navigation and map studying, labeled primary and secondary learning, respectively (Presson and Hazelrigg 1984). Direct navigation refers to actually moving along a route; map study means the study of a route via a map or some other figural display. Experiments have demonstrated that the spatial knowledge acquired from maps is different from that obtained while navigating the environment (Thorndyke and Hayes-Roth 1982; Moeser 1988). For example, Thorndyke and Hayes-Roth (1982) argue that, with moderate exposure, subjects who were map learners did better in judgments of relative location and straight-distance estimation than direct navigation learners, who were better at orienting themselves with respect to unseen objects and estimating route distances.

Thorndyke and Hayes-Roth also contend that through extended exposure direct navigation learners can make their route knowledge into configurational knowledge, and eliminate the advantage that map learners had. However, Moeser (1988) found that in complex spatial environments, survey maps do not develop automatically even in those subjects who traversed the environment extensively.

A second difference in the way spatial knowledge is stored between primary and secondary learning is that map learners create an orientation-specific cognitive map of an environment while direct navigation learners have a more orientation-free mental map (Evans and Pezdek 1980; Levine, Jankovic et al. 1982; Thorndyke and Hayes-Roth 1982; Presson and Hazelrigg 1984). These studies show that map users have larger pointing errors when not aligned to the learned map's orientation as compared to direct navigation learners. Moreover, these pointing errors are linearly proportional to the relative angle between the user's orientation and the map's orientation. Evans et al. (1980) and MacEachren (1992) state, however, that viewing a map in multiple orientations may lead to map learners with more orientation-free cognitive maps.

While all the studies cited focus on terrestrial, 2-D and 3-D, environment navigation in 1-G situations where subjects remain upright, their conclusions have implications for our experiment which is the first to explore navigation in a 3-D environment, e.g., space station, in 0-g, with 6 degrees of freedom. Microgravity makes any space station a complex spatial environment in which astronauts can assume any orientation and navigate in a third dimension (e.g. "walking on walls"). This complexity is confirmed by the accounts of crewmembers that lived on MIR. In-flight, crewmembers felt they could not point from one module to another, even though they had extended exposure to the environment. Instead, they were more dependent on visual landmarks to recall routes (Richards, Clark et al. 2001). Lack of survey knowledge after extended space station exposure corresponds to Moeser's results (1988). It also seems beneficial to provide astronauts a way to acquire orientation-free mental maps of space station. Such configurational knowledge might be critical if, for example, a crewmember's essential visual cues to navigate are obscured. Furthermore, good survey knowledge is essential in coordinating activities outside the space station, e.g. spacewalks and docking maneuvers. Thus, training should emphasize survey knowledge, which to prepares astronauts for work and navigation in station.

2.3 VIRTUAL REALITY

Virtual reality (VR) allows the creation of environments that cannot otherwise be replicated on the Earth. NASA already uses VR, along with neutral buoyancy facilities, to train astronauts for extra-vehicular activities (EVAs), i.e. spacewalks. Currently, in preparation for intra-vehicular activities, astronauts practice on full-size mockups of the individual core modules of International Space Station that are laid out to resemble assembly in space (Figure 2.1). They view each module in its vertical upright orientation, and rely largely on images and maps of the station to understand the overall layout. A virtual space station, however, would allow astronauts to see the station as a whole from the inside, even the modules that are not in the same plane. In addition, astronauts would not be constrained to viewing modules only upright, but could experience being inside the station in any posture, or orientation.

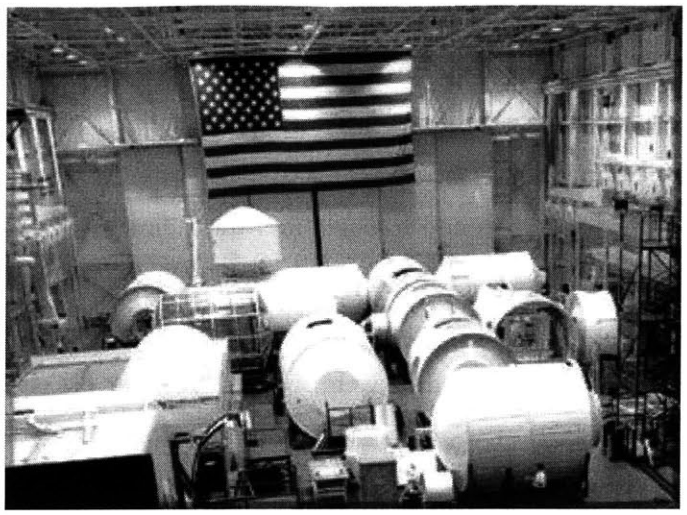


Figure 2.1 NASA Johnson Space Center ISS Mock-Up Facility

While VR gives us the ability to render unique environments, there are some limitations in using this technology. First there are the "real world" constraints, i.e. the user's mobility is restricted to the area that the VR head tracking system covers. VR head tracking systems used today have head tracking volumes typically limited to several cubic meters. A virtual space station is so large and three-dimensional that it is impossible for users to simply "walk around" a virtual station. Users of VR space station simulators must therefore be provided with some artificial (and therefore potentially unrealistic) means to translate their viewpoint within the environment. Finally, astronauts inside a virtual space station cannot see the orientation of other objects and modules that are not in direct view, unless some special provisions are made (see below).

If VR users of a station were artificially moved through the environment (i.e. moved in a way that did not involve physical locomotion), how transferable is the cognitive map learned with a computer-simulation to real-world movement? There is evidence that suggests virtual simulation of direct terrestrial 1-G navigation is transferable, implying that training approximates primary, but not secondary, learning. Several studies of desktop computer navigation simulations have shown that this type of navigation simulation produced elements of cognitive maps that might be expected from direct navigation experience itself (Golledge, Dougherty et al. 1995; Tlauka and Wilson 1996; Bowman, Davis et al. 1999; Richardson, Montello et al. 1999; Rossano, West et al. 1999). Golledge et al. (1995) and Richardson et al. (1999) found that map learning remained better as a way to convey the understanding spatial relations than computer desktop navigation simulation. Unlike terrestrial direct navigation learning, survey maps acquired via desktop navigation are thought by some to be orientation-specific (Richardson, Montello et al. 1999; Rossano, West et al. 1999) rather than orientation-free (Tlauka and Wilson 1996). As for immersive virtual navigation simulations, Witmer et al. (1996) and Bowman et al. (1999) have successfully used VR for the study of route knowledge transfer and the maintenance of spatial orientation, respectively (Witmer, Bailey et al. 1996; Bowman, Davis et al. 1999). Satalich (1995), however, cautions that direct virtual navigation is time-consuming and may not allow sufficient time for the development of survey knowledge (Satalich 1995).

Many of the studies cited have suggested that a reason computer navigation is not equivalent to real navigation is its lack of proprioceptive and vestibular cues which would help VR users keep spatially oriented (e.g. Richardson, Montello et al. 1999). When spatially disoriented, it is hard to acquire a cognitive map of the environment. Aoki et al. (2000) found that when simulating virtual pitch rotations along a route, subjects tended to become disoriented and could not correctly reconstruct the path (Aoki, Yamaguchi et al. 2000). His results can be attributed to VRIs, a subject's inability to redirect the virtual gravity by 90° or 180° because of the effect of the real gravity vector. Many studies (Chance, Gaunet et al. 1998; Klatzky, Loomis et al. 1998; Bakker, Werkhoven et al. 1999) support the conclusion that optic flow alone is insufficient to correctly update angular spatial orientation. Instead, they favor a virtual reality system that includes proprioceptive feedback. More specifically, Lathrop and Kaiser favor immersive systems to maintain spatial orientation (Lathrop and Kaiser 2002, in press). Bakker et al. (1999) contends that kinesthetic and vestibular feedback is the most useful since it allows VR users to move and rotate at their own initiative. Thus, it can be argued that limited cognitive maps can be established using immersive, head-tracking virtual navigation simulation.

Many have addressed the issues presented above in relation to navigation in large, virtual environments. Some proposed solutions involve the use of a VR navigational aid. Darken and Sibert (1993) presented a toolset for navigation in virtual environments. They conclude that real world navigational aids, such as maps, could be applied in VR. More importantly, they believe that if the task had been 3-dimensional, a 2-D map view would have been of little use (Darken and Sibert

1993). In a later study, the same authors write that their "results suggest that users of large-scale virtual worlds require structure (direction indicators, maps, and path restriction) in order to effectively navigate" (Darken and Sibert 1996).

Based on previous work in navigation, Stoakley, Conway and Pausch (1995) devised "Worlds in Miniature" (WIM). WIM (Pausch, Burnette et al. 1995), as described by Pausch, is "a hand-held miniature graphical representation of the virtual environment" (Pausch, Burnette et al. 1995). WIM's two major objectives are to give the user the opportunity to gain a larger context of the environment and to provide the ability to move to locations that are not easily visible or not within walking or grabbing range. The first objective addresses the limitation of traditional VR that allows for only one point of view (his/her own). The second objective is to remove some of the "real world" constraints.

A WIM user holds a miniaturized version of the surrounding virtual environment (VE) referred to as "the model", and can manually manipulate other objects within the model. A WIM user can thus interact with both the VE and the model. For example, objects manipulated in the WIM model will also move in the full size VE and vice versa. Within the WIM model, there is also an iconic representation of the user, called the "doll icon", or, as in this thesis, an avatar. After the avatar is moved, the user can trigger a "fly in" so that the user's viewpoint appears to fly into the model so the user seems to "become" the avatar, and the model now becomes the full size VE. This "fly-in" procedure allows for the substitution of model/avatar rotations and translations in place of user rotations and translations within the surrounding VE (Stoakley, Conway et al. 1995).

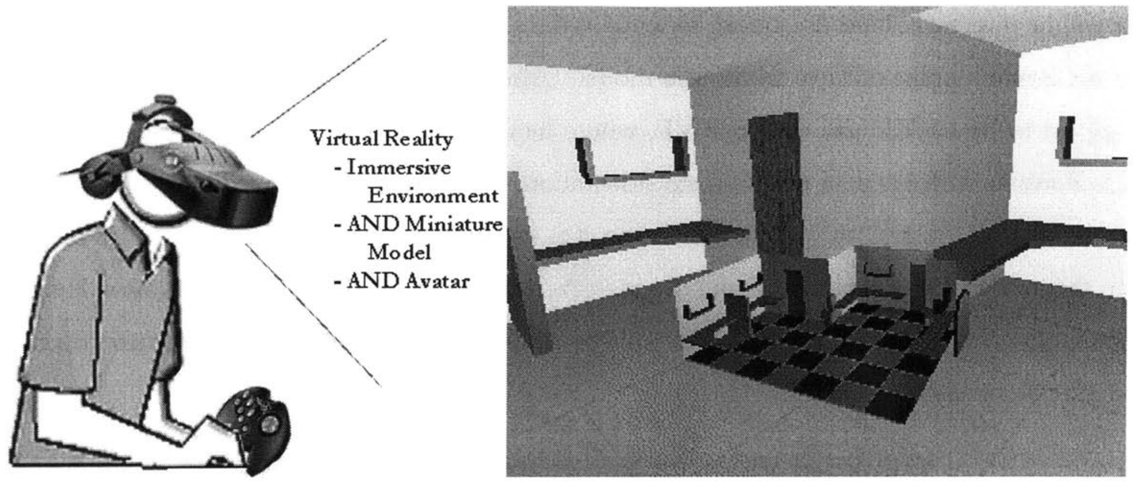


Figure 2.2 Subject Immersed in a Virtual Environment with WIM tool

There are several potential advantages of using a WIM. First, the user can manipulate the model and see the entire VE from different angles. Stoakley et al. 1995 state, "We believe that this interaction technique can establish a new viewpoint more quickly and with less cognitive burden than a technique that requires an explicit 'flight' command and management of the flight path." The cognitive burden would be relating the user's and the avatar's orientation and position, and path planning. WIM overcomes range and occlusion problems by providing a bird's eye view of VE. Finally, the user has the ability to reach and handle objects that could not have otherwise been seen or reached. However, none of these advantages for orientation and navigation have so far been demonstrated in quantitative experiments.

2.4 SPACECRAFT IN MINIATURE

"Spacecraft in Miniature" (SIM) extends the WIM concept to 3-D, six degree of freedom navigation. SIM employs a miniature graphical representation of the surrounding virtual environment (the model), an iconic representation of the user (the avatar), and a fly-in method. Both SIM and WIM substitute potentially disorienting rotations of surrounding virtual environment (VE) with manual model rotations. With SIM, the VE is no longer restricted to one floor/level and an upright user orientation. Users are required to manipulate the model in such a manner so that prior to each fly in, the avatar is positioned and aligned with the user's point of view (Figure 2.5). Essentially, the user will be directly behind the avatar, looking in the same direction. Then, the user will slide, or fly, into the avatar's point of view within the model. After this purely translation motion, the user will be appear to be back in the full scale VE, where he can call upon the miniature model and see an updated avatar with his new position and orientation. This differs slightly from WIM, in which fly-ins can involve rotation as well as translation, i.e. users are not required to align their orientation with the avatar before initiating the fly-in. A rotation while flying into the new location is undesirable because research (cited earlier) shows that virtual rotations without concurrent vestibular cues are potentially disorienting.

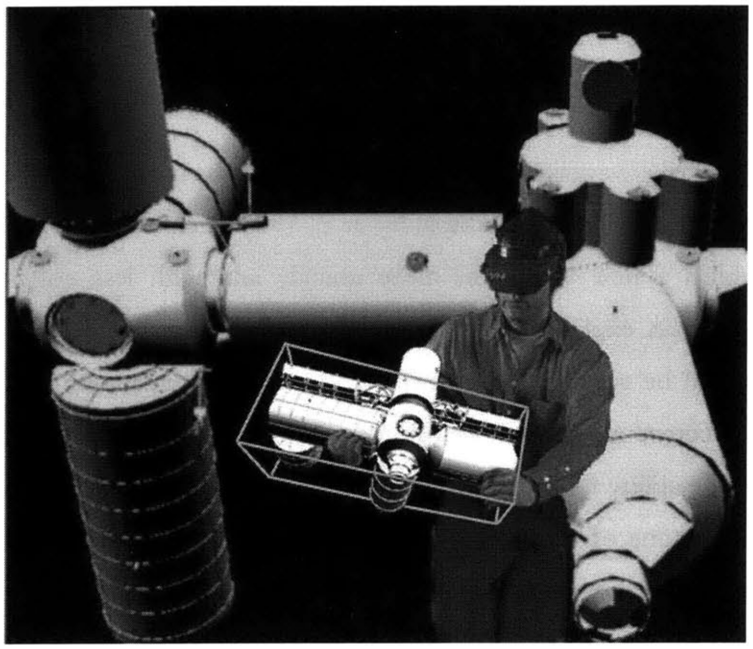


Figure 2.3 Concept Depiction of SIM Virtual Reality User

Our goal was to create a large-scale 3D-virtual environment navigational tool that will make the experience of using VR less disorienting. SIM presents the user with a virtual model of the environment, i.e. a 3D map that provides an exterior ("bird's eye") point of view of the VE. Within the model, the avatar indicates where the user is located within the VE and how he is oriented in it.

The most immediate application to such a navigational tool is astronaut training for the International Space Station, an environment where navigation and orientation is necessary in all three dimensions. The user studies the structure and layout of the space station by looking at the miniature model, rotating and translating it through the use of a six degree of freedom manipulator. The user can see inside the space station because as the model is turned, the model station's walls disappear and reappear to reveal its interior (Figure 2.4) much like looking into a traditional child's dollhouse.

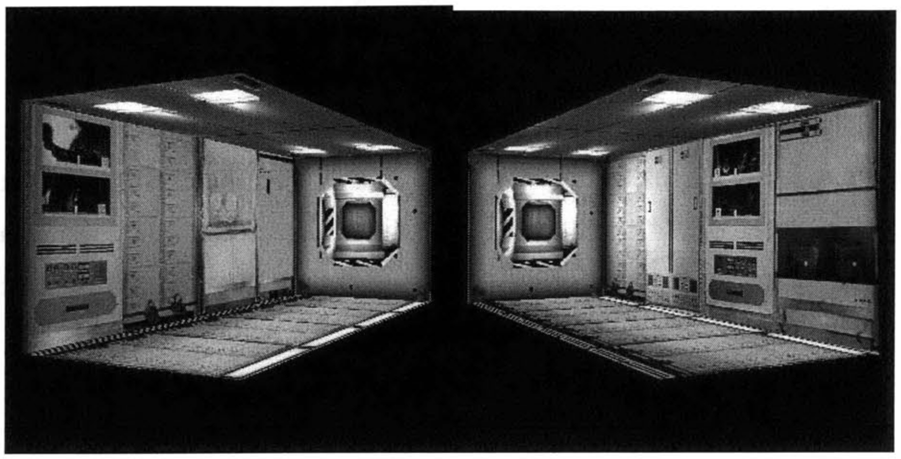


Figure 2.4 Constant Interior View of Virtual Model

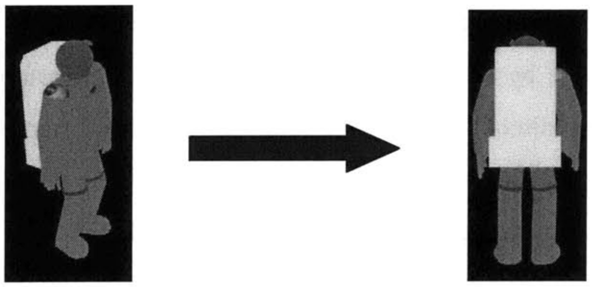


Figure 2.5 Space Station's Avatar Aligned

Our experiment was aimed at demonstrating the potential advantages of training with "Spacecraft in Miniature" over training using direct navigation of a virtual environment without a SIM tool. First, SIM gives the user a map to help them learn a large, complex virtual environment. Second, SIM may allow for the development of an orientation-free cognitive map of the station because it is possible to manually turn the miniature model. Users are required to utilize their hands and to manipulate the miniature model in order to align themselves with the changing posture of the avatar, thus integrating rotations imposed on the model into the subject's interpretations of their implied body rotations. This manipulation allows for haptic and proprioceptive feedback and provides multiple point of views which some researchers say plays a role to orientation-free cognitive maps (Evans and Pezdek 1980; MacEachren 1992). Third, SIM allows for navigation in a large virtual environment and permits body rotations, pitches and rolls that are otherwise impossible on Earth.

Due to hardware and programming constraints, the selection and manipulation of the avatar or other objects in the virtual environment were not implemented in this experiment. For experimental purposes, we also wanted the subjects to move only over specific routes, so all subjects would have the same visual experience. We did not want to give the subjects the ability to manually place the avatar in arbitrary positions. Instead, the avatar was moved to pre-determined locations for SIM users.

2.5 HYPOTHESIS

We set out to show that after training to learn a complex 3-D virtual environment using the navigational tool "Spacecraft in Miniature", users acquire better survey knowledge than a virtual navigation control group who only view the virtual environment directly. A "tour guide" led both groups along the route. The SIM group viewed the station directly at the starting point of a route, and followed the tour guide by calling up the model environment, rotating the model so the tour guide was facing the same direction, and then initiating a "fly-in" translation. The control group viewed the station by being passively translated along the same route and then passively rotated (pitches and yaws) at the route turn point. We hypothesized that SIM users, like terrestrial map learners, should have a better survey knowledge, in terms of their understanding of where modules are located and how they are oriented. They should also be more accurate and faster to respond to questions concerning the locations and orientations of modules after their own orientation was changed, implying a more orientation-free mental representation of the environment. On the other hand, we expected the control subjects might have better landmark and route knowledge (describing routes based on landmarks) than SIM users since their direct navigation training experience required them to be dependent on local visual cues to know their position and orientation. Also, to the extent that it was more difficult to acquire survey knowledge under the control condition, these subjects would have had more practice navigating using only landmark and route-based strategies.

3 METHODS

3.1 PARTICIPANTS

The participants were 11 females and 15 males, ranging in age from 19 to 38[1]. Many were graduate students in the Aeronautics and Astronautics Department at MIT. Those eligible were given compensation for participating. One female participant did not complete the experiment because she experienced initial symptoms of motion sickness during the training phase.

Prior to starting the experiment, subjects completed a cube-rotation test (Witkin, Oltman et al. 1971) and a questionnaire (Appendix C) that was aimed at eliminating subjects with medical conditions that could affect their performance or participation in the experiment. The scores on cube rotation test, which measures ability to mentally visualize 3D objects in rotated positions, were used to balance the subject groups, SIM and Control, for mental rotation ability (mean scores: Control group = 23.3; SIM group = 25.8; only 10% difference between groups in mean test scores). The groups were also approximately balanced by gender.

3.2 EQUIPMENT AND MATERIALS

The virtual environment was a generic virtual space station that was created with this experiment in mind. The virtual space station was made up of seven rectangular modules and two rectangular nodes. The nodes resembled ISS nodes in interior shape and hatch arrangement. Each module was assigned a unique name, listed in Table 3.1, which associated it with the identity of a prominent landmark object located within the module. Modules were given names so that subjects could more

[1] Only one subject was above 29 years of age.

easily remember module names and recall landmark objects within them in a short training session. Salient landmark objects were placed inside each module, not only to help subjects remember the names of the modules, but also to help establish an easily remembered spatial framework. In initial training, the salient landmark object was always on the right wall of the module when the subject was facing the closed end of the module (or in the case of the Control module, which had two open ends, when facing the end colored blue). In later training, subjects learned that when facing the closed/blue end of a module, in an "upright" orientation, the salient object was on the right (and conversely, when the salient object was on their right when facing the closed/blue end, they were "upright").

Label	*Name*	*Object*
0	Experiment Module	Experiment racks
1	EVA Module	Space Suit (EMU)
2	Storage Module	Refrigerator with food
3	Health Fitness Module	Stationary bike and heart monitor
4	Habitation Module	Sleeping bag
5	Centrifuge Module	Centrifuge
6	Control Module	Monitors, computers

Table 3.1 Module Names and Objects

Some aspects of the arrangement of the modules resembled the layout of space stations such as Mir and ISS. The generic space station used in the experiment had a central module with two nodes attached to its ends. Each node led to three other modules. Three modules were oriented similarly (see Figure 3.1, modules labeled 0, 1 and 6). Modules perpendicular to the main block of modules (modules labeled 2 and 3) had vertical uprights that were parallel, or co-aligned. Modules (4 and 5), however, were intentionally inverted (180°) with respect to modules 0, 1, and 6.

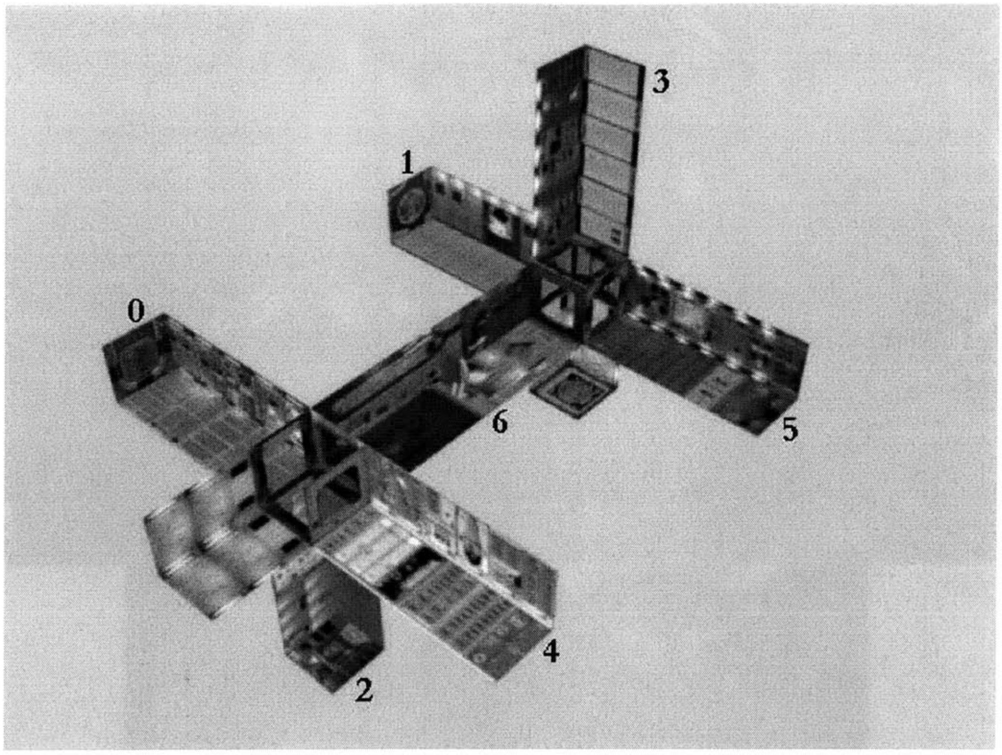
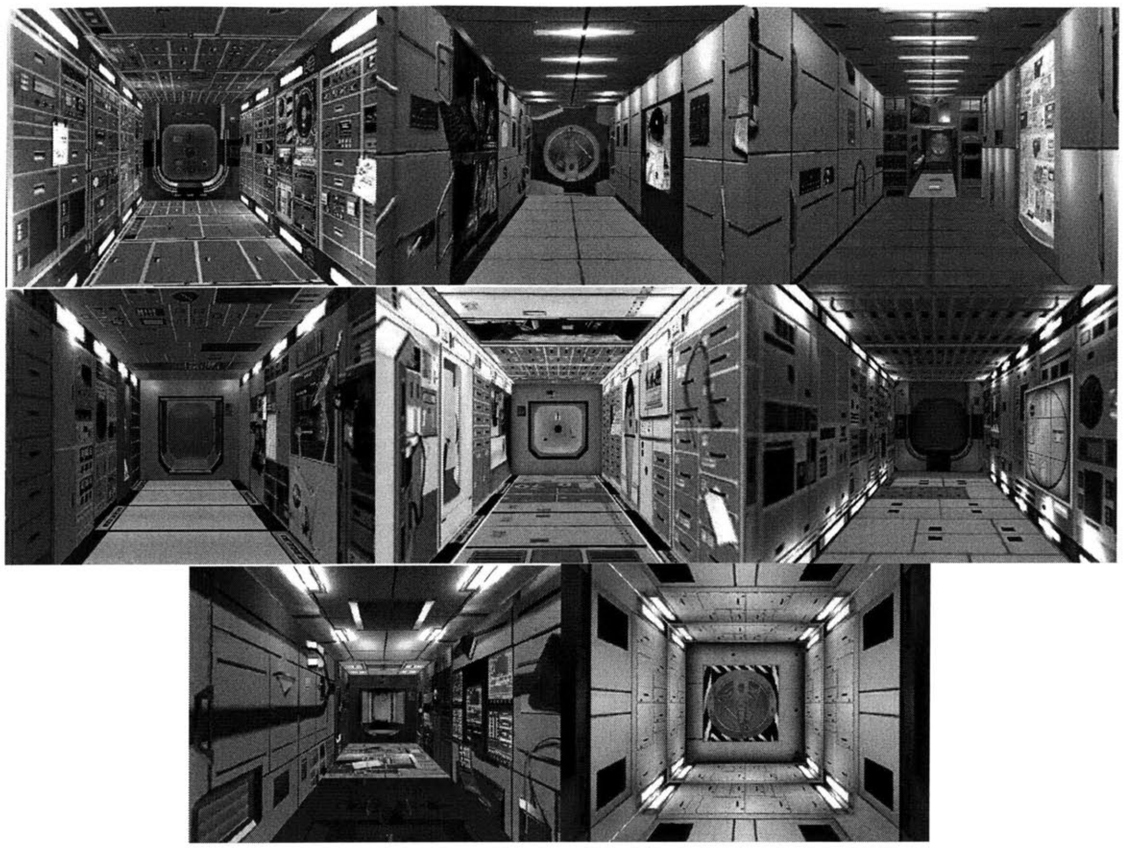


Figure 3.1 Virtual "Generic" Space Station

The station model was created using 3D Studio Max Version 3 with surface textures borrowed from other computer simulated space stations and the world wide web (Figure 3.2). The station was then rendered using Python (version 2.0), the language used for all the programming, and the virtual reality library VRUT (version 2.4) (see Reference for websites). The virtual environment was displayed using a graphics-accelerator-equipped PC workstation, which rendered the graphic images in RGB mode at a 640 by 480 resolution (horizontal and vertical, respectively) in each eye. The signal was converted to VGA (640 x 480 60 Hz) in order to be displayed properly in the Virtual Research V8 head-mounted display (HMD). Images were rendered in stereo.



From left to right, top to bottom: Experiment, EVA, Storage, Health Fitness, Habitation, Centrifuge, Control Module, and a Node.

Figure 3.2 View of Modules' Interiors, Upright and Forward Direction

The HMD had dual 1.3" diagonal Active Matrix Liquid Crystal Displays with a resolution per eye of ((640 x 3) x 480). The field of view was 60° diagonal with 100% binocular overlap. The interpupillary distance, adjusted by the user, was between 52 mm and 74 mm.

Each subject's head was tracked using an IS-600 Mark II (Intersense, Inc., Burlington, MA) hybrid inertial/acoustical/magnetic tracking system. Six degree of freedom tracking was used, with position resolution (X/Y/Z) of 2.5 mm RMS and an angular resolution (P/R/Y) of 0.10° RMS. A second sensor, similar to the one utilized for head tracking, was used to manipulate the miniature virtual model (SIM).

The second sensor was mounted on an Interact Hammerhead game pad through which the user also interacted with the program (e.g. entering answers, activating the "fly in" sequence) using one of its ten buttons or two joysticks. The game pad and sensor was the manipulator tool that allowed users to manipulate the SIM. Once the user activated the miniature model, every rotation and translation on the manipulator would be applied to the miniature model. Since some hand manipulations were physically difficult to do (i.e. turning the tool more than 90°), a function was implemented that would allow users to incrementally rotate the miniature model. Once the user reached the limit of his turn angle, he could deactivate the manipulator to release the model, rotate the manipulator back to the original orientation, reactivate it, and repeat to apply further rotations.

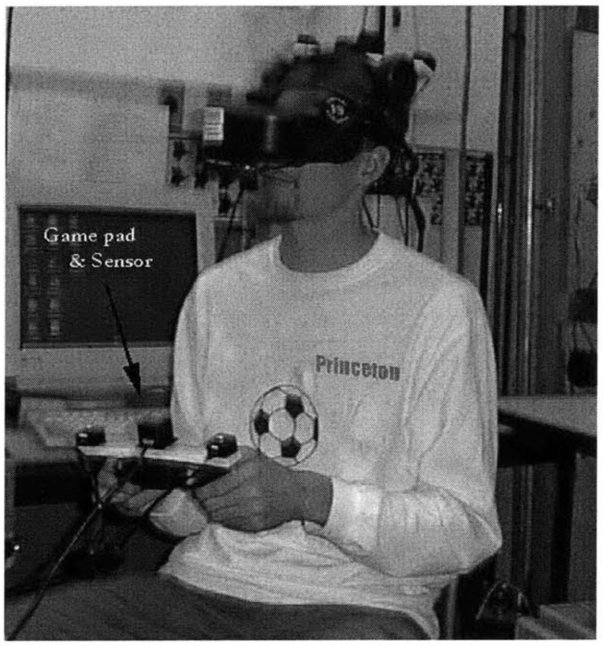


Figure 3.3 User with Head Mounted Display, Game pad, and Sensors

3.3 DESIGN AND PROCEDURE

All subjects were instructed that the goal of the experiment was to learn the layout of the space station. Appendix B contains those instructions. In order to provide an adequate number of breaks, the experiment had three steps. The first and third steps were identical for both experimental groups. In the first step, all subjects were trained to identify the names and landmarks of each module. In the second step, subjects were guided through virtual routes within the station in order to learn the layout. The SIM group subjects were trained on the station's layout using the SIM miniature virtual model (Figure 3.4); the Control group did not have the tool available. The third step, the test section of the experiment, measured subjects' survey and landmark/route knowledge. The former was assessed by asking the subjects to point to a target module from the orientation they were placed in, while the latter was measured using a series of route description questions.

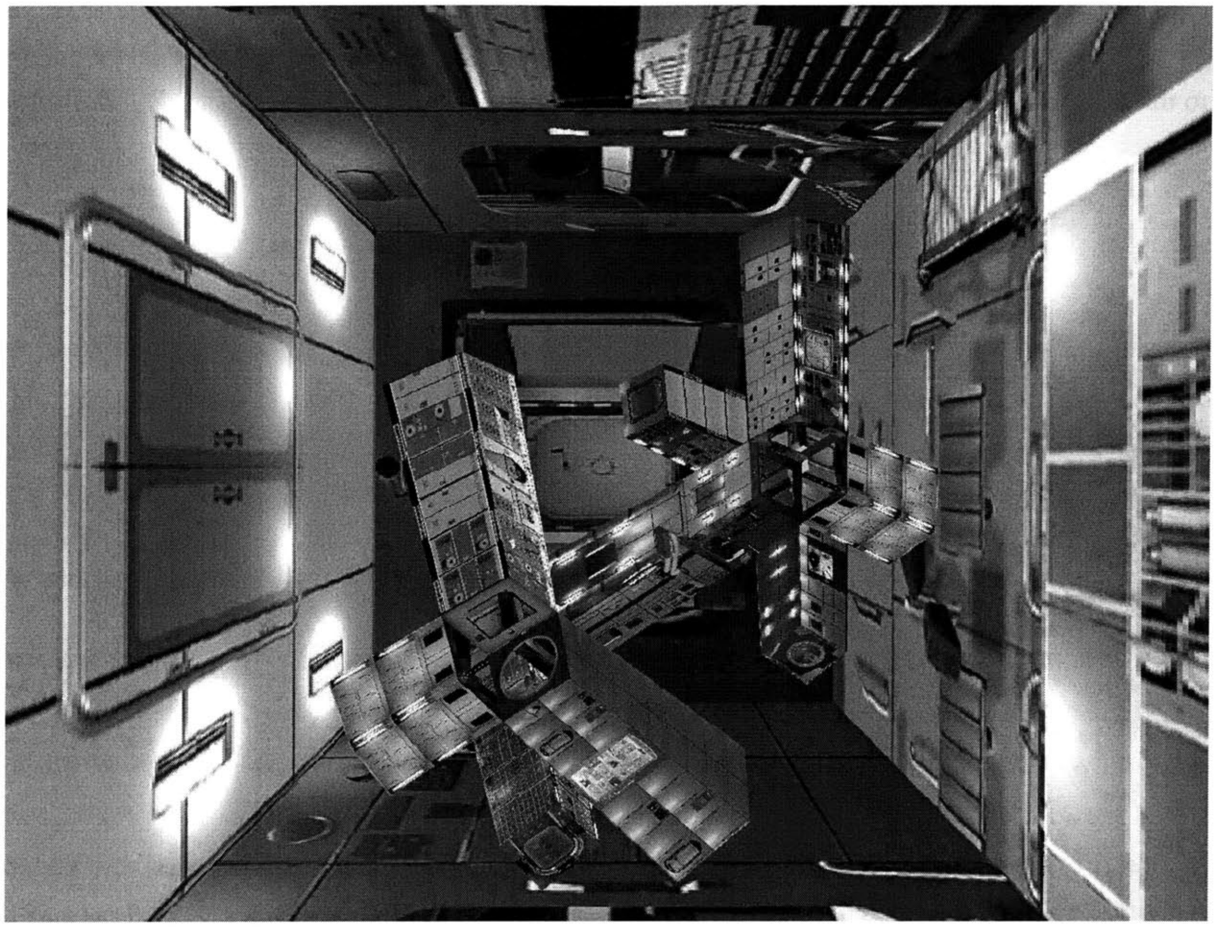


Figure 3.4 View of SIM while Immersed in Virtual Space Station

3.3.1 STEP 1: LEARNING INDIVIDUAL MODULE LANDMARKS

In step 1, subjects learned the name of each individual module and identified the landmark or object associated with it. This section resembled current intra-vehicular activity (IVA) training in that each module was seen individually in a fixed orientation that defined visual vertical. All subjects were told that they would view each module upright and were responsible for learning to recognize and to name each module. They were also given instructions on how to figure out which orientation was upright for a given module (i.e. salient landmark on right wall when facing closed/blue end). Subjects were free to look around the module. They could also move their virtual viewpoint along the length of the module using a one-axis joystick to control their position. Before proceeding to the next step, they were required to correctly identify each module and a node. The sequence of events within Step 1 is delineated in Table 3.2.

	Description	***Comments***
First Viewing	40 sec in each of the 8 module/node, after which the computer placed user in the next module/node (always upright).	Shown the name of each module, moved the length of the module/node, and permitted to look around.
Second Viewing	Repeat of first viewing, but with only 7 sec in each module/node.	Same as above
Test Phase	Shown a module or node upright and required to respond with the correct name of module. Feedback given if correct or incorrect.	User shown a virtual list of possible answers and asked to select one. User could look around, but not traverse the module/node.
Repeat Phase	If user incorrectly identified a module/node, he repeated the viewing phase and was tested again on all modules.	

Table 3.2 Step 1 Time Line

3.3.2 STEP 2: LEARNING LAYOUT OF THE ENTIRE STATION VIA ROUTES

In this step, all subjects were to learn the routes to all the modules relative to the Control Module. They were instructed that their task was to figure out which modules were at which end (Blue or Yellow) of the Control Module, where these modules were located relative to the Control Module, and how each module was placed (oriented) relative to it.

Step 2 was divided into two parts (see also Table 3.3). In the first part, subjects learned two routes beginning in the familiar upright posture. Before being shown a route, subjects were shown the name of the target, destination module. Along the way, they had to answer questions about the route (Table 3.4), intended to help them learn the layout of the station. The questions required them to describe the route to the target module as seen from the initial posture. Subjects were told these questions were ones that also would be used in the third step of the experiment. In addition, once subjects faced the target module, each was asked to name the modules that were radially around him/her. Pilot trials of the experiment had revealed earlier that subjects often failed to look around the environment. These questions were added to encourage subjects to survey the station visually.

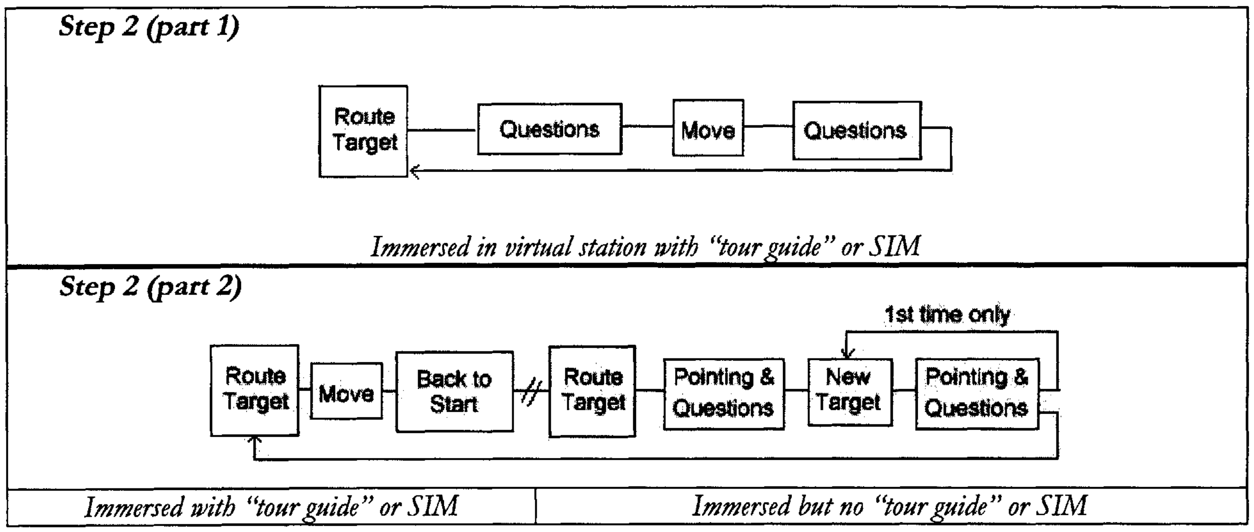


Table 3.3 Step 2 Summary of Sequence

Questions	*Possible Answers*
In the Control Module, facing forward, I am:	Upright, Right-Shoulder Down, Upside-down, Left-Shoulder Down, or "I don't know".
In Control Module, which hatch leads to the target module?	Blue hatch or Yellow hatch.
Once in the node, how would you turn to face the target module?	Pitch 90° forward, Pitch 90° back, Yaw 180°, Yaw 90° left, Yaw 90° right, or "I don't know".
Once in target module, how would you turn to be upright in the target module?	Roll 0° (no turn), Roll 90° left, Roll 90° right, Roll 180°, or "I don't know".

Table 3.4 Route Description Questions and Possible Answers

In the second part of Step 2 (see also Table 3.3), subjects learned four other routes, or destinations, with initial postures that were either upright or right shoulder down. Due to time constraints, it was only feasible to train in one non-upright posture. After completing the route, subjects were virtually transported back to the beginning of the route (posture and position) and asked to complete two tasks: point to the target module just visited and answer the route description questions. Subjects were told that in Step 3 they would have to do these tasks for all modules in four initial postures (upright, right shoulder down, upside down and left shoulder down). To familiarize subjects with the method that would be subsequently used to measure survey and route knowledge in Step 3, subjects were required to practice the tasks (pointing and route description) not only for the target module, but also for two other surrounding modules. When a subject completed answering a set of questions, he/she was given feedback on their answers. If incorrect, they were told the correct answers.

The Control group were told that they would have an astronaut "tour guide" that was facing in the direction of the target module. Their task was to find the "tour guide" within the virtual environment and study his position and orientation (see Figure 3.5). The subjects were allowed to look around and investigate that section of the station for as long as they chose to. Subjects were then allowed to push a button that initiated a passive movement into the guide's position and orientation. The computer simulated a single translation into the appropriate node and then one rotation (i.e. yaw left, yaw right, pitch forward, or pitch back) in order to have the subject face the target module. Subjects were allowed to visually familiarize themselves within the node.

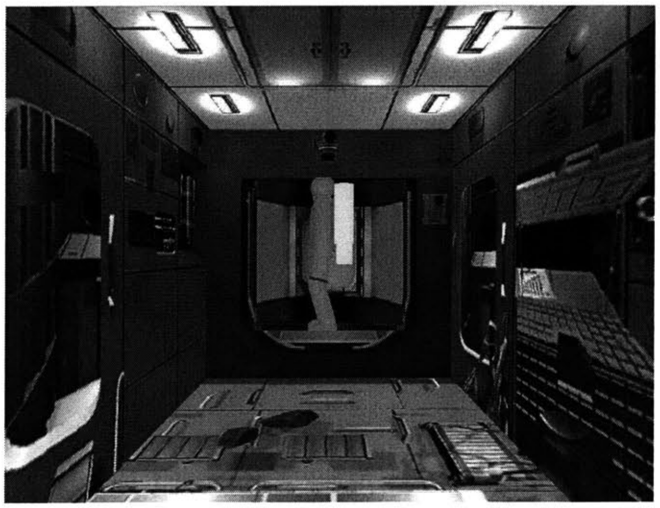


Figure 3.5 Control Subject Looking at "Tour Guide" Astronaut

Subjects in the SIM group had a similar task. They were told that they had a miniature station model they could manipulate. Within the miniature model, there were two astronauts, one that represented the subject (in initial position and posture) and another who was a "tour guide" that was facing the target module. Subjects were instructed to study the model and the "tour guide's" position and orientation. Once they decided to "fly" into the guide's position, they were instructed to manipulate the model so that they were aligned with and looking directly at the backpack of the "tour guide" astronaut (see Figure 3.6). (Before starting Step 2, SIM subjects were allowed to practice until they were comfortable manipulating the model with the 6-DOF sensor). The subject pushed a button, and the computer simulated a single "fly in" translation into the model, making the subject adopt the point of view of the "tour guide" inside the larger virtual environment. The participants were also allowed to visually explore in that section of the station.

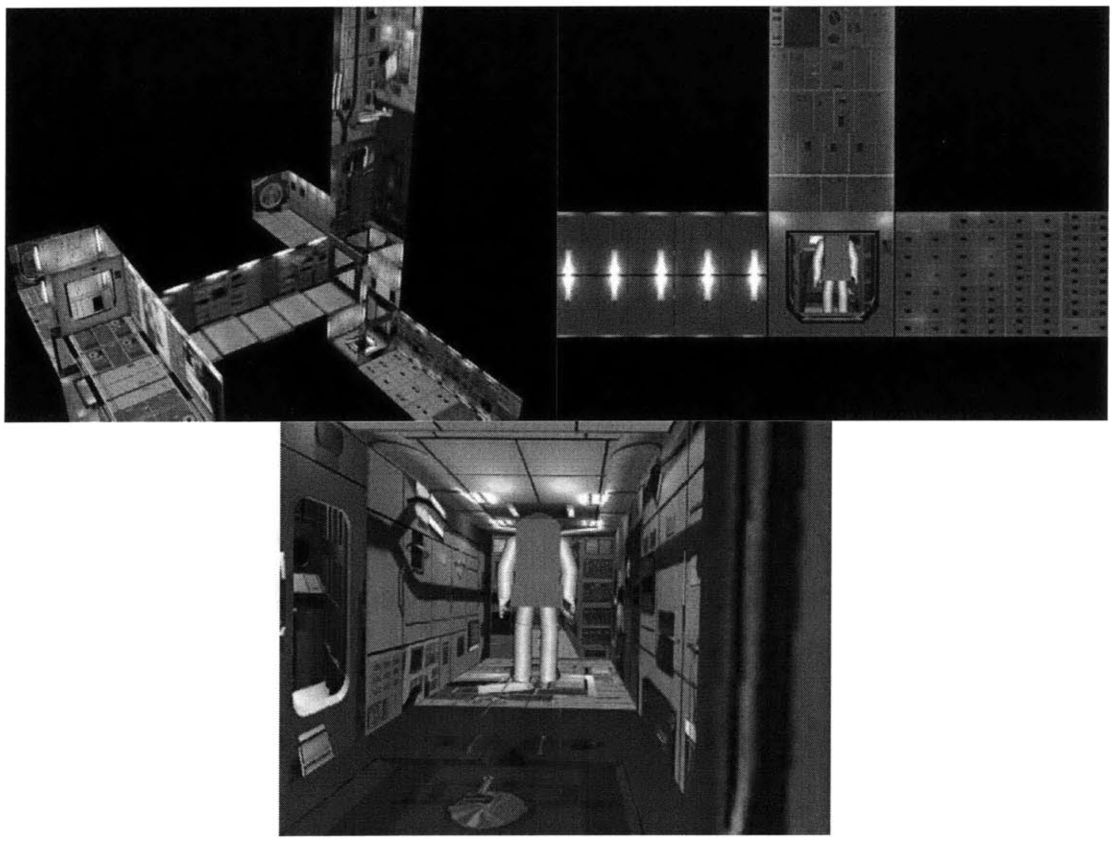


Figure 3.6 SIM Subject through a "Fly" in Sequence (Order: top left, right, bottom)

All subjects were presented the same routes in the same order, regardless of which group they were in. They were also told exactly how the test step was to be conducted and that they would not have the "tour guides" or the miniature model in Step 3.

3.3.3 Step 3: Measuring 3-D Knowledge of Station Layout and Routes

All subjects performed the same 24 trials: 4 initial orientations x 6 target modules, in the same order. The trials always started in the Control Module in one of the 4 possible initial postures: upright, right-shoulder down, upside-down, or left-shoulder down (Figure 3.7). Subjects were then given the name of the target module, and asked to point to it. They were not required to point to a specific place in the target module. Subjects pointed by moving a virtual crosshair that moved as their head moved (Figure 3.8), and then pushed a button to enter their answer. In essence, participants pointed with their head, and the pointing direction was inferred from the rotation applied to the head tracker. After a subject had indicated his pointing answer, he then answered the route description questions that revealed what he knew about how to reach the target module from that initial position and posture (Figure 3.9). Subjects had no time constraints, and they received no feedback.

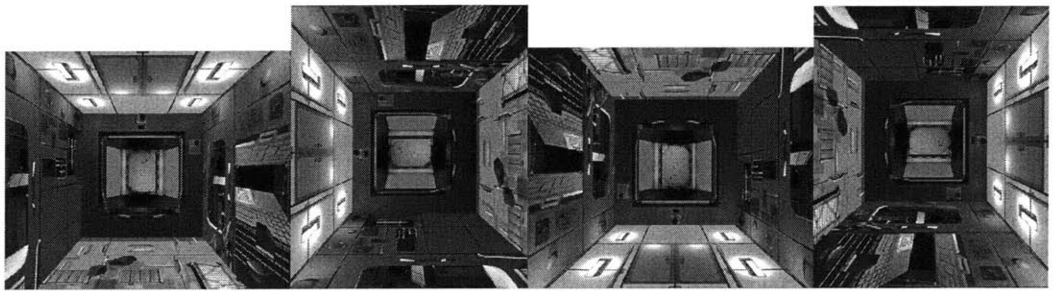


Figure 3.7 Four Possible Subject View Orientations in the Control Module, from left to right: Upright, Right Shoulder Down, Upside Down, & Left Shoulder Down

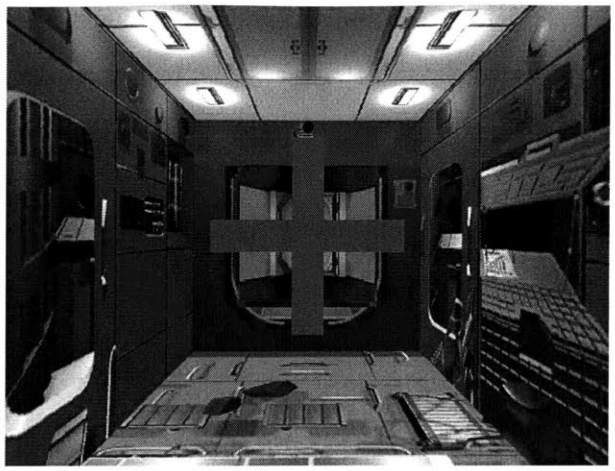
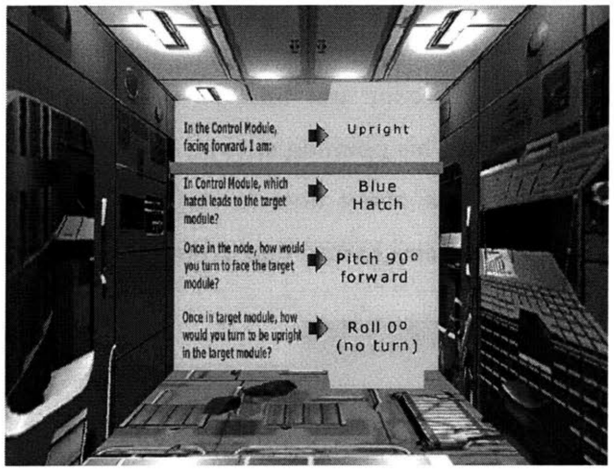


Figure 3.8 Pointing Crosshair as seen by All Subjects



In the Control Module, facing forward, I am:	Upright
In Control Module, which hatch leads to the target module?	Blue Hatch
Once in the node, how would you turn to face the target module?	Pitch 90° forward
Once in target module, how would you turn to be upright in the target module?	Roll 0° (no turn)

Figure 3.9 Route Description Questions Task as seen by All Subjects

Each task and trial was scored "1" if the answer was <u>incorrect</u>, and "0" otherwise. For the pointing task, since subjects were instructed to simply point to the target module's general location, and not to any particular place on it, only absolute pointing error angles greater than 40° were used to score an incorrect.

Finally, all subjects were asked to fill out a post-experiment-questionnaire concerning their experience and were asked to reconstruct a physical model of the station with rectangular blocks (Figure 3.10). Each rectangular block, or module, made of transparent plexi-glass, had a picture of the module's salient object on one interior wall and a picture of the closed hatch on another. The experimenter documented the final arrangement of the modules.

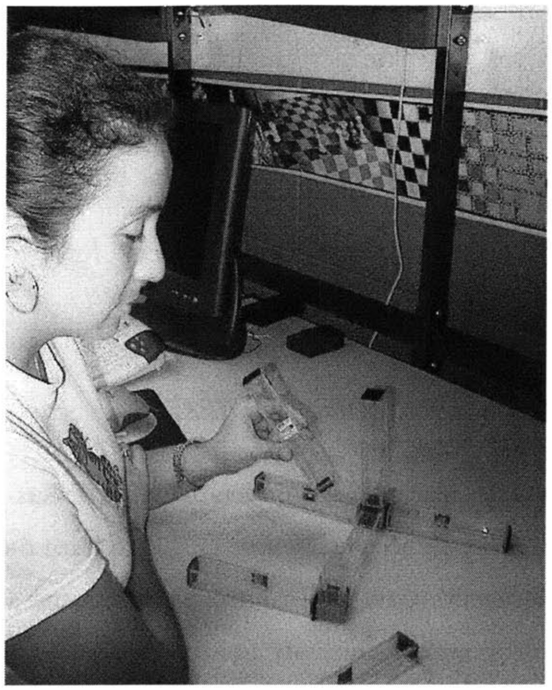


Figure 3.10 Subject Constructing Physical Model of Station

4 Data and Results

Twelve subjects were tested with the Control version of the experiment, while fourteen were tested with the SIM version. Data and response times were collected for the two tasks: pointing to target module and answering route description questions.

4.1 Determining Trained Subjects

Since the study's object was to examine the effects of SIM versus Control treatments among successfully trained subjects, those subjects who were unable to learn the task in the artificially short time available for the experiment were omitted from subsequent analysis. Individual subject performances were analyzed by task, particularly for the direction pointing task. These analyses determined which subjects had not succeeded in learning much about the station layout. Figure 4.1 and Figure 4.2 show the mean fraction of incorrect pointing errors for all Control and SIM subjects, ordered by error fraction. The excluded subjects (Control subjects numbered 1 - 3, and SIM subjects 1 – 4) were those whose mean fraction incorrect were greater than 50%, and had correspondingly high median pointing error angles. These subjects are subjects 1, 6, 9, 15, 23, 26 and 27 in the Appendices. This reduced the Control group to 9 subjects and the SIM group to 10.

Though the subject groups were balanced using cube rotation test scores, no correlation was found between cube rotation test scores and a subject's quality of pointing or route description. Hence cube rotation test scores did not predict which subjects would train to the 50% mean error fraction criterion in the time allowed.

4.2 DIRECTION POINTING TASK

Figure 4.1 and Figure 4.2 show that the SIM and Control groups (all 26 subjects)[2] had similar distributions of pointing errors. The SIM group, however, had more subjects with nearly perfect (less than 10% incorrect) scores than the Control group. SIM subjects also seemed to be either poor or very good[3], while the Control group had subjects at several levels of performance. These are consistent with the hypothesis that SIM improves configurational knowledge acquisition to the very good level except for the previously identified group of subjects who were weak at the pointing task.

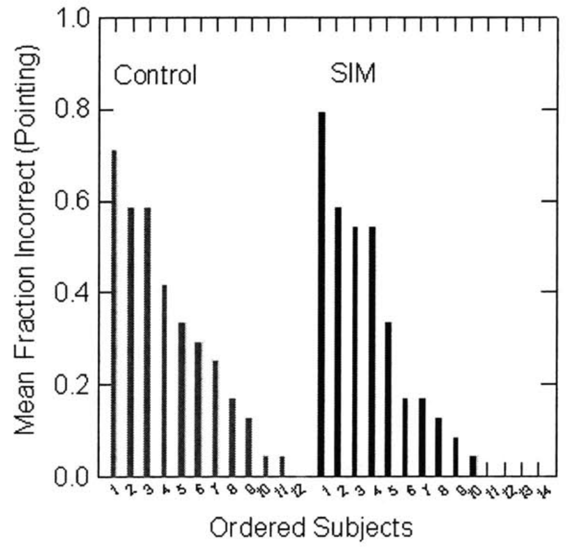


Figure 4.1 Mean Fraction Incorrect for Direction Pointing Task per Group

[2] Figure 4.1 and 4.2 are the only figures to show all 26 subjects performance. All other figures are of properly trained subjects.

[3] Only SIM subject labeled 5 in Figure 4.1 would be considered to have intermediate performance. His errors (see Subject 17, Appendix E) cannot be attributed to misplacement of module or poor mental rotation.

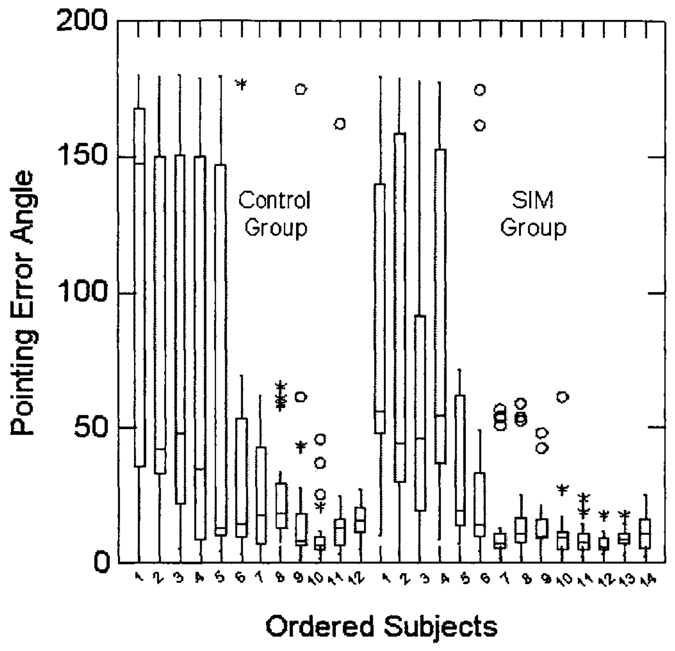


Figure 4.2 Distribution of Pointing Error Angle per Subject

Pointing errors were categorized by their largest rotation component in a particular direction. Pointing error types were pitch, yaw, and roll errors, which were not used to determine accuracy in the pointing task. In general, both SIM and Control groups had similar distributions of pointing error types (Figure 4.3 and Figure 4.4). Though most pointing errors were not pure yaws, pitches, or rolls, they did have a largest component in a particular direction, which was then used to classify the pointing error type. Figure 4.4 is a 2-D projection of a 3-D plot of the yaw, pitch, and roll components of the error vector. Since the roll components were small in magnitude (relative to the other components), only the yaw and pitch components of the error vector were used to construct Figure 4.5. Figure 4.5 was made by plotting pitch versus yaw components, each multiplied by its corresponding error angle. This figure shows that all large errors were yaws, corresponding to pointing to the wrong side of the station. Clusters around $\pm 50°$ indicate errors that were on the correct side of the station but in the wrong place. Again, both groups have similar distributions.

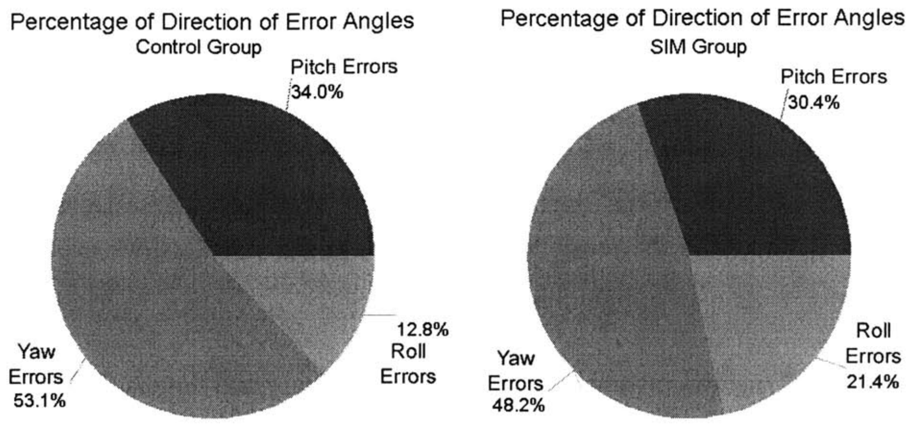


Figure 4.3 Distribution of Direction of Error Angle for Control and SIM Groups

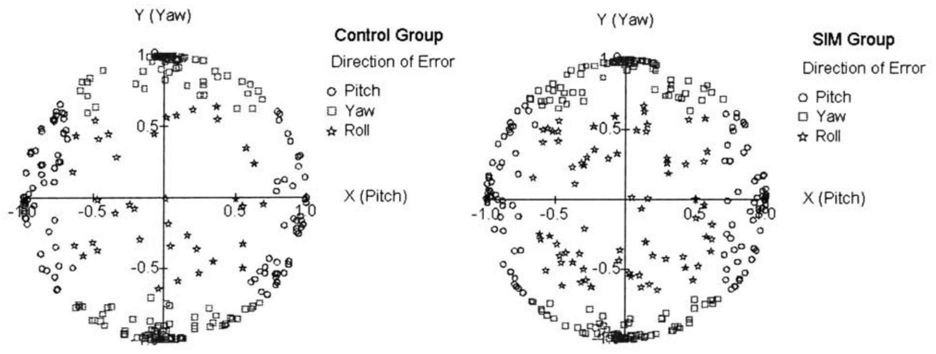


Figure 4.4 Distribution of Error Vector Components for Control and SIM Groups

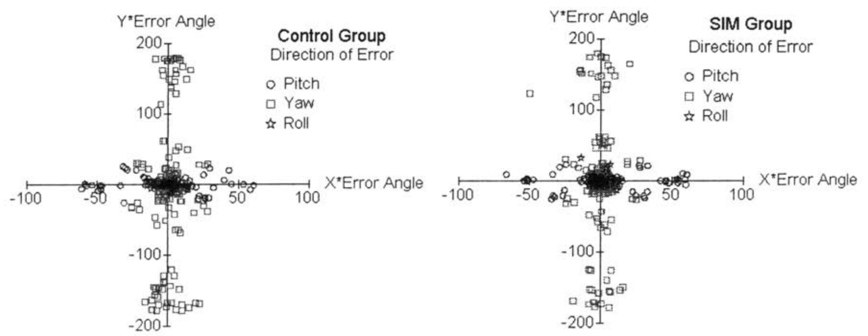


Figure 4.5 Distribution of (Direction * Angle of Errors) for Control and SIM Groups

4.2.1 Effect of Posture

It was expected that initial posture, or orientation, would affect a subject's ability to point correctly to the target module. In Step 3, all subjects were tested on four initial postures: upright, left shoulder down, upside down, and right shoulder down. Since most of the training was done in an upright posture and since it is the most "natural" of the initial postures, we expected subjects to perform their best on trials with upright initial condition.

Mean error angles and fraction incorrect were plotted by posture and targets (Figure 4.6, Figure 4.7 and Figure 4.8). Over all postures, the properly trained SIM subjects had smaller mean error angles and mean fraction incorrect for the direction pointing task as compared to the properly trained Control subjects.

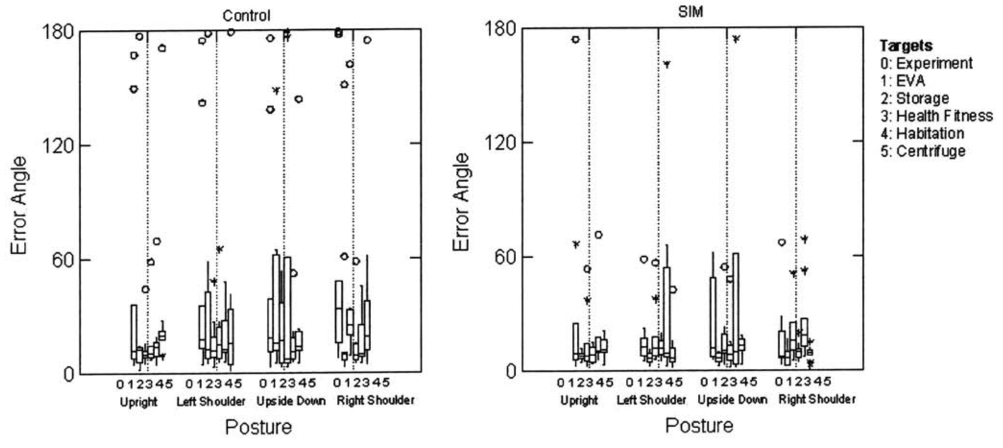


Figure 4.6 Distribution of Error Angle by Posture and Target per Group

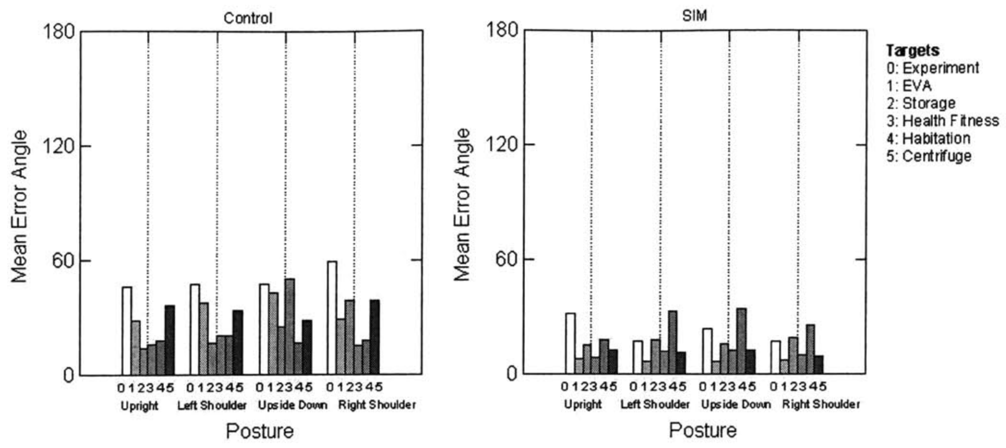


Figure 4.7 Mean Error Angle by Posture and Target per Group

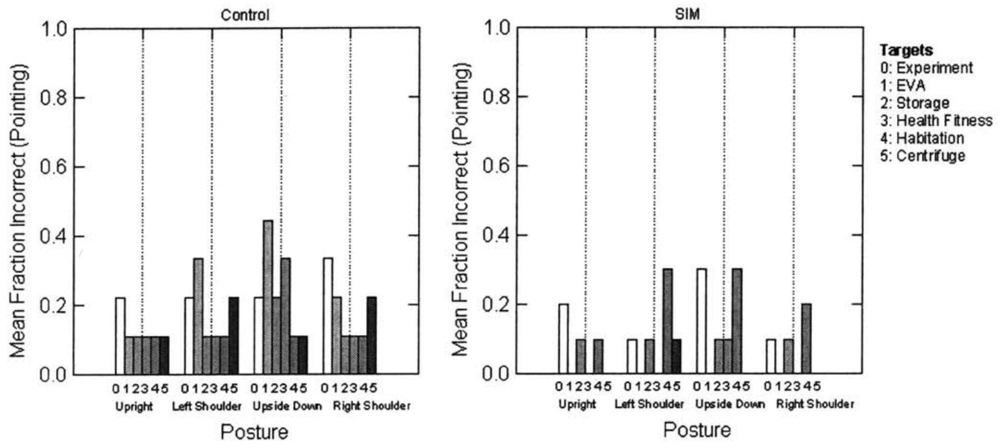


Figure 4.8 Mean Fraction Incorrect by Posture and Target per Group

Table 4.1 and Table 4.2 shows results of the Kruskal-Wallis non-parametric tests (Conover 1999) done on the means for error angle and fraction incorrect by group for all trials and for each initial posture. The SIM group produced significantly smaller ($p < 0.05$) mean error angles over all conditions and for right shoulder down initial posture. For the SIM group, the mean fraction incorrect in right shoulder down initial posture was comparable to that in the upright initial condition. For both measures, the initial upside down posture produced the highest mean error rate among the groups.

Error Angle	*Control Group*		*SIM Group*		
Initial Posture	Mean	Std Err Mean	Mean	Std Err Mean	P-value
Upright	26.267	5.713	15.602	3.192	0.165
Left-shoulder down	29.293	5.799	16.176	3.060	0.086
Upside down	35.084	6.566	17.426	3.378	0.086
Right-shoulder down	33.185	6.321	14.607	1.800	0.003
All Postures	30.957	3.043	15.953	1.454	0.041

Table 4.1 Direction Pointing Mean Error Angle Posture Comparisons by Groups (Kruskal-Wallis Non-parametric Test)

Direction Pointing	*Control Group*		*SIM Group*		
Initial Posture	Mean	Std Err Mean	Mean	Std Err Mean	P-value
Upright	0.130	0.046	0.067	0.032	0.259
Left-shoulder down	0.185	0.053	0.100	0.039	0.275
Upside down	0.241	0.059	0.133	0.044	0.305
Right-shoulder down	0.185	0.053	0.067	0.032	0.056
All Postures	0.185	0.026	0.092	0.019	0.136

Table 4.2 Mean Fraction Incorrect for Direction Pointing Posture Comparisons by Groups (Kruskal-Wallis Non-parametric Test)

A Friedman rank test based on error angle (Table 4.3) showed a statistically significant ranking only for the Control group, which performed their best in the upright posture. Ranking by the fraction incorrect data (Table 4.4) suggested a trend that performance was worst in the upside down posture for both groups, though not statistically significant for either group.

Error Angle			
Control Group		*SIM Group*	
Posture	Rank	Posture	Rank
Upright	13.0	Upside Dwn	22.0
Upside Dwn	24.0	Upright	24.0
Left Shldr	25.0	Left Shldr	26.0
Right Shldr	28.0	Right Shldr	28.0
P-Value	*0.035*		*0.753*

Table 4.3 Friedman Ranking of Postures by Mean Error Angle for Groups

Percent Incorrect				
	Control Group		*SIM Group*	
	Posture	Rank	Posture	Rank
	Upright	17.5	Upright	22.0
	Left Shldr	22.5	Right Shldr	22.5
	Right Shldr	23.5	Left Shldr	26.0
	Upside Dwn	26.5	Upside Dwn	29.5
P-Value		*0.104*		*0.073*

Table 4.4 Friedman Ranking of Postures by Mean Fraction Incorrect for Groups

In summary, initial posture had some effects in the pointing task results. While the SIM group had overall significantly lower mean error angle for the pointing task than the Control group, they had it only for the right shoulder down initial posture (Table 4.1). Both groups had the highest error rate when in the upside down initial condition.

4.2.2 EFFECT OF TARGET

If subjects made a consistently larger percentage of errors, or larger average error angles, when pointing to particular targets, then it would have indicated that there was something about the targets or the training procedure that made it difficult to remember how to localize them. There was no significant cross-effect of target and group on mean error angle and fraction incorrect except that the SIM subjects had significantly smaller error angles than the Control for the EVA and the Centrifuge modules by Kruskal-Wallace non-parametric test ($p < 0.05$).

Nevertheless, subjects' performance by target apparently depended on which end of the station the route required them to go: forwards (through the blue hatch) and backwards (through the yellow hatch). Subjects normally faced the blue hatch at the start of all trials, and the first routes learned were also at this end of the station. Modules at the two ends of the station may have been remembered in hierarchical fashion, with those on the blue side "chunked" together. Analysis of error angles and fraction incorrect by hatch color (blue and yellow hatch modules)[4] revealed that SIM subjects did particularly well when making judgments concerning the blue end of the station.

[4] EVA, Centrifuge, and Health Fitness are blue hatch modules, while Experiment, Habitation, and Storage are yellow hatch modules.

They had significantly lower mean error angles (p-value < 0.05, Kruskal-Wallis tests) for the blue hatch modules than the Control subjects, but the difference for the yellow hatch modules was not as large nor statistically significant (Table 4.5). This could be evidence of SIM subjects' preference for facing the blue hatch while performing the pointing task.

Error Angle				***Fraction***	***Incorrect***	
	Mean	Mean		Mean	Mean	
	Control	*SIM*	*P-value*	*Control*	*SIM*	*P-value*
Blue Hatch Modules	31.380	9.612	0.009	0.204	0.017	0.009
Yellow Hatch Modules	30.535	22.293	0.191	0.167	0.167	0.639
Difference (Blue-Yellow)	0.845	-12.681	0.191	0.037	-0.150	0.153

Table 4.5 Angle and Direction Comparisons by Blue/Yellow hatch per Subgroup (Kruskal-Wallis Non-parametric Test)

A Friedman rank test of mean error angle showed a significant effect of target for SIM group (Table 4.6). The easiest modules to point to according to SIM subjects were the EVA and Centrifuge and Health Fitness, which were located through the blue hatch of the station; the hardest modules, Storage, Habitation and Experiment, were those located through the yellow hatch. Friedman ranking of fraction incorrect by target did not yield significant results for either group.

Error Angle			
Control Group		*SIM Group*	
Module	Rank	Module	Rank
EVA	25.0	EVA	14.0
Habitation	27.0	Centrifuge	31.0
Health Fitness	28.0	Health Fitness	32.0
Storage	35.0	Storage	42.0
Experiment	36.0	Experiment	43.0
Centrifuge	38.0	Habitation	48.0
P-value	*0.448*	*P-value*	*0.001*

Table 4.6 Friedman Ranking of Targets by Mean Error Angle

A similar result was not seen for the Control group whose Friedman ranking of targets was not significantly concordant over its subjects (Table 4.6). The Control group's overall trend appears to be like the SIM group's, to separate modules into blue/yellow sides, if it were not for the Habitation and the Centrifuge modules. Control subjects performed better when pointing to the Habitation

Module than the Centrifuge Module. The Centrifuge Module had a few high error angles that were skewing its mean, but even when these were suppressed, neither the ranking order nor its significance changed. It can be speculated that the low mean error angle for the Habitation Module, as compared to those for Centrifuge Module, is due to Habitation being the last module shown in training, and thus better remembered by Control subjects.

In summary, the SIM group appeared to have developed a different way of accessing information about the layout of the station than the Control group. SIM subjects seemed to have "chunked" the station into two sections, while evidence for this occurring within the Control subjects was not strong. The SIM group performed significantly better in the pointing task than Control subjects for targets that were located through the blue hatch of the station, which indicated that subjects training with SIM developed a preference for that side of the station.

4.2.3 RESPONSE TIMES FOR DIRECTION POINTING

Figure 4.9 and Figure 4.10 show the distribution of response times (RT) for the direction pointing task for all properly trained subjects. Analysis of means was also repeated suppressing outlying response times (RT > 90 seconds) but this did not result in different conclusions. Figure 4.10 shows that the SIM group had a smaller standard deviation for response times as compared to the Control group.

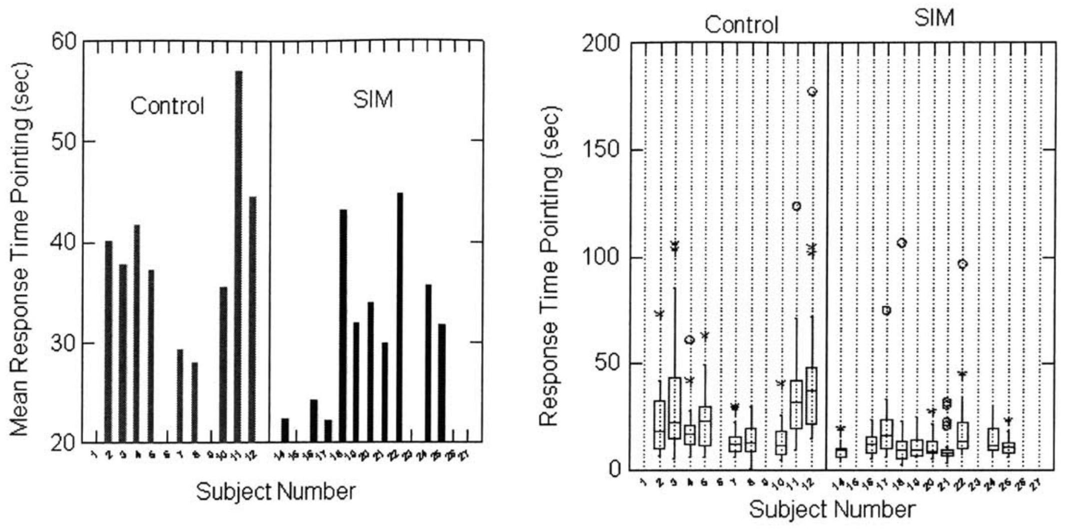


Figure 4.9 Response times for Direction Pointing Task per Group by Subject

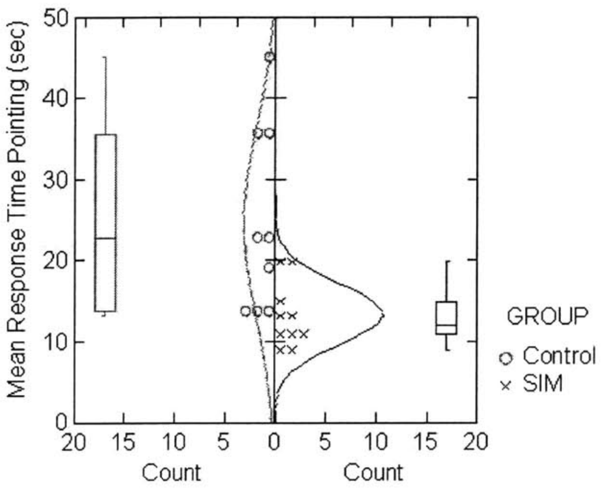


Figure 4.10 Distribution of Mean Response Time for Direction Pointing Task by Group

Overall, the SIM group was significantly faster at the pointing task, regardless of initial posture (Table 4.7, $p < 0.05$, Kruskal-Wallis non-parametric test)[5]. Response time is another measure of

[5] P-values for t-test were not all significant for each initial posture. However, Kruskal-Wallis p-values are reported because it is a more robust test since it does not rely on the data being normally distributed and is not sensitive to outliers as the t-test.

performance since lower response times may imply lower mental loads. Thus, based on RT, SIM subjects, overall, performed significantly better than Control subjects in the pointing task, which complements the significantly lower SIM mean error angle as compared to Control mean (Table 4.1).

It was expected the lowest average response times for both groups would be the trials that had upright initial posture. While true for the Control group, the shortest average RT for the SIM group is the right shoulder down initial posture (Table 4.7). It is possible that since response times decreased over trials, SIM subjects' response times were quicker in the right shoulder down condition because more of these trials occurred in the latter half of the experiment (Figure 4.11). The experiment was not large enough to balance for this possible effect in addition to initial posture, target, and type of turn. However, this does not explain SIM subjects' greater quickness as compared with Control subjects over all trials. The order of trials, therefore, cannot account for the overall lower mean RT among SIM subjects.

RT Pointing	*Control Group*		*SIM Group*		*K-W*	*T-test*
Initial Posture	Mean	SD	Mean	SD	P-value	P-value
Upright	18.473	8.752	12.135	4.423	0.018	0.075
Left-shoulder down	28.020	14.587	13.145	5.402	0.006	0.016
Upside down	29.745	16.769	16.930	7.220	0.027	0.058
Right-shoulder down	22.348	11.895	10.895	3.160	0.014	0.021
All Postures	24.647	11.556	13.276	3.723	0.009	0.019

Table 4.7 Response Time for Direction Pointing Posture Comparisons by Group (Kruskal-Wallis Non-parametric Test and T-test)

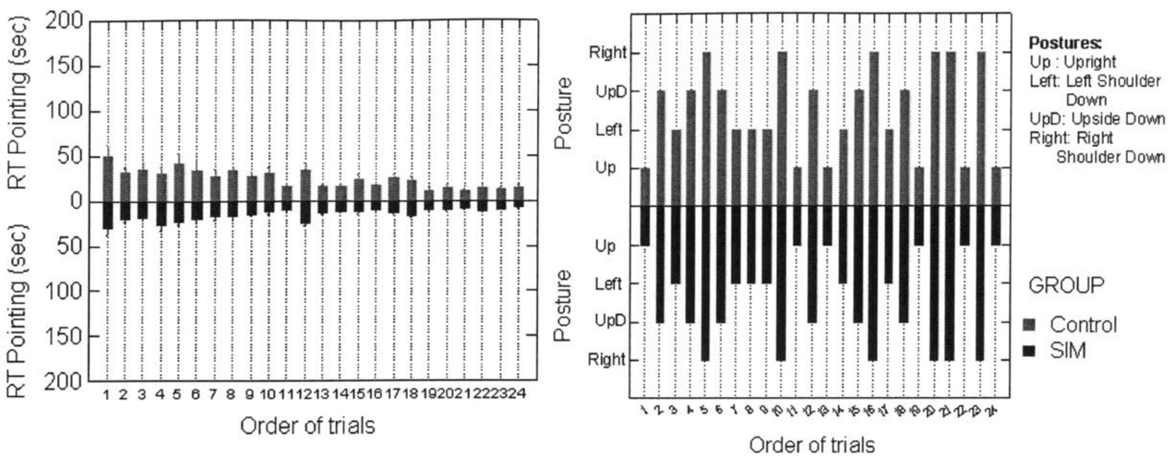


Figure 4.11 Direction Pointing Response Time and Postures over Order of Trials

The Friedman ranking of initial postures by response times (Table 4.8) confirms that SIM subjects had the shortest RT for the right shoulder down initial posture. Apart from the later appearance of right shoulder down trials in Step 3, there is another alternative reason why the right shoulder down initial posture resulted in shorter response times: subjects trained (Step 2) in the right shoulder down initial posture, which did not occur for the other non-upright postures. In addition, SIM subjects had significantly lower mean error angle for the right shoulder down initial posture as compared to Control subjects (Table 4.1). Both reasonings imply that exposure to non-upright conditions could improve performance for tasks that involve those postures.

RT Pointing				
Initial Posture	*Control*	*Rank*	*SIM*	*Rank*
	Upright	14.0	Right Shldr Dwn	15.0
	Right Shldr Dwn	18.0	Upright	21.0
	Upside Dwn	28.0	Left Shldr Dwn	26.0
	Left Shldr Dwn	30.0	Upside Dwn	38.0
P-value		*0.008*		*0.001*

Table 4.8 Friedman Ranking of Postures by RT Pointing by Groups

Table 4.8 shows that the upside down condition resulted in longer RT for the SIM group. The Control subjects seem to have short RT for upright and right shoulder down initial conditions as compared to the other postures (upside down and left shoulder down). This appears to be the same trend seen in Figure 4.11.

Based on response times by targets (Table 4.9), the SIM group had significantly lower RT for the pointing task over all targets, except the EVA Module ($p < 0.05$ for Kruskal-Wallis non-parametric test). Subjects within groups agreed on the relative quickness of their ability to point to a module (Table 4.10). Similarly as in mean error angles (Table 4.6), SIM subjects (in this case, Control subjects as well) had shorter RT for modules that were found through the blue hatch than those found through the yellow hatch. However, this could be an effect of having to physically turn around to point to the "yellow side" modules. While SIM subjects had significantly smaller mean error angles than Control subjects for only "blue side" modules (Table 4.5), they had significantly shorter RT for both "blue side" and "yellow side" modules (Table 4.11).

RT Pointing	*Control Group*		*SIM Group*		*K-W*	*T-test*
Target	Mean	SD	Mean	SD	P-value	P-value
Experiment	28.900	14.395	18.502	7.416	0.041	0.076
EVA	18.378	11.839	11.727	7.584	0.086	0.173
Storage	24.035	16.132	11.972	4.389	0.022	0.058
Health Fitness	21.573	14.131	11.356	5.716	0.034	0.069
Habitation	30.399	13.172	16.101	4.814	0.003	0.012
Centrifuge	24.594	15.605	9.999	4.107	0.011	0.024

Table 4.9 Response Time for Pointing Comparisons by Target per Group (Kruskal-Wallis Non-parametric Test and T-test)

RT Pointing				
Target	*Control*	*Rank*	*SIM*	*Rank*
	EVA	19.0	Centrifuge	23.0
	Centrifuge	27.0	EVA	28.0
	Health Fitness	28.0	Health Fitness	28.0
	Storage	31.0	Storage	30.0
	Experiment	41.0	Habitation	47.0
	Habitation	43.0	Experiment	54.0
P-value		*0.023*		*0.001*

Table 4.10 Friedman Ranking of Targets by RT Pointing per Group

RT Pointing	*Control*	*Group*	*SIM*	*Group*	*K-W*	*T-test*
	Mean	SD	Mean	SD	P-value	P-value
Blue Hatch Modules	21.515	12.226	11.027	4.940	0.014	0.036
Yellow Hatch Modules	27.778	11.837	15.525	3.577	0.011	0.015

Table 4.11 Response Time Comparisons by Blue/Yellow hatch per Group (Kruskal-Wallis Non-parametric test and T-test)

4.2.4 Summary of Direction Pointing Results

Percent incorrect, error angle and response time in the direction pointing task were used as dependent measures of survey knowledge. SIM subjects had a significantly lower mean error angle and pointing response times as compared to Control subjects, implying that the SIM subjects had a better survey knowledge of the station than the Control subjects. SIM subjects significantly outperformed Control subjects for the right shoulder down initial posture. SIM subjects were also faster at pointing to a target when their initial posture was changed, which implies a more orientation-free mental representation of the station. However, the SIM group also preferred modules on the blue side of the station, as analysis of the effect of targets on pointing accuracy indicates some orientation-specificity. We suspect that SIM subjects separated the modules in the station into two "chunks". With this framework, SIM subjects had significantly lower mean error angles for the blue/front modules.

4.3 Route Description Questions

4.3.1 Question 1: What is Your Initial Posture?

Most subjects had little difficulty answering the first route description question and thus determining their initial posture (upright, left shoulder down, upside down, or right shoulder down). In Figure 4.12, one SIM subjects stood out as poor at understanding their posture. Upon closer examination, it appeared that this subject misunderstood the definition of right and left shoulder down, but not necessarily the posture he was in. Thus, most pointing and route description mistakes cannot be attributed to misperception of initial posture.

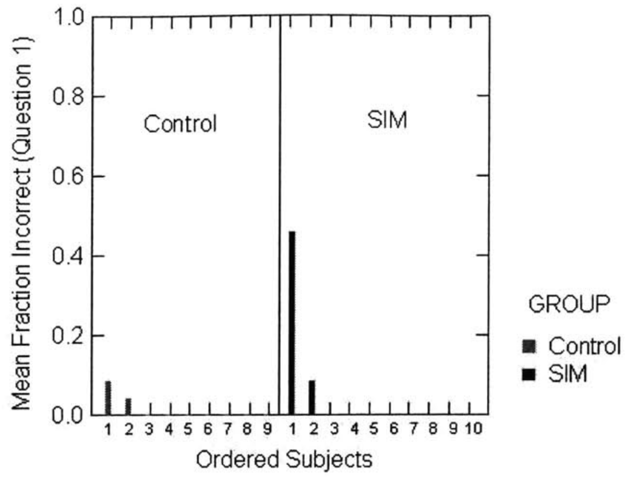


Figure 4.12 Mean Fraction Incorrect for Question 1 (Initial Orientation) per Subject

4.3.2 QUESTION 2: TURN TOWARDS BLUE OR YELLOW HATCH?

The second route description question asked the subject to tell through which hatch the target module was to be found. There were only a few subjects that made any mistakes in Question 2 (Figure 4.13). The rest of the subjects were able to divide the six modules into their corresponding sides (through the blue or yellow hatch) of the station. Figure 4.14 illustrates that the groups did not agree on which modules were easier to place in the correct side.

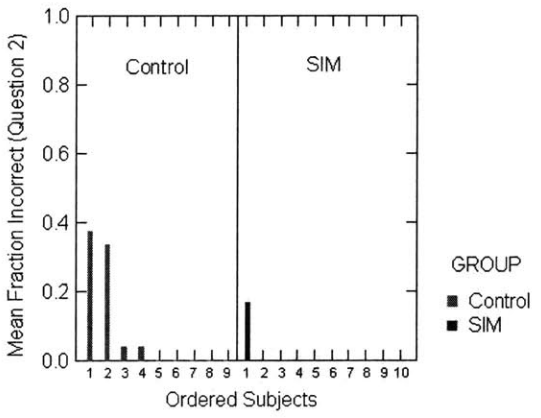


Figure 4.13 Mean Fraction Incorrect (Question 2: Blue/Yellow Hatch) per Group

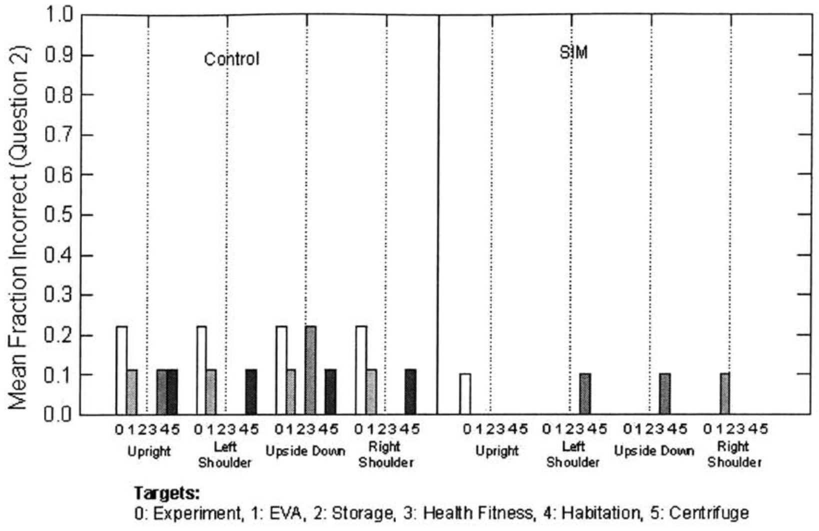


Figure 4.14 Mean Fraction Incorrect (Question 2) by Posture and Target per Group

4.3.3 QUESTION 3: WHICH WAY TO TURN IN NODE TO FACE TARGET?

The third route description question asked what turn was necessary in the node (yaw or pitch) to face the target module. A wrong answer to Question 3 was equivalent to making a wrong turn along a route that was being physically traversed. Figure 4.15 shows that SIM subjects appear to have a lower mean fraction incorrect for Question 3, but the difference does not appear to be very large.

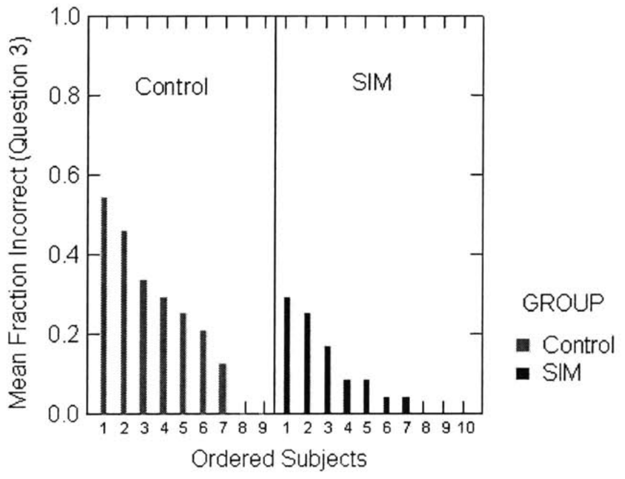


Figure 4.15 Mean Fraction Incorrect (Question 3: Turn in Node) per Group

Initial posture had some effect on the performance of subjects when answering Question 3. Overall, both groups did best in the upright initial posture. The trend was for the SIM group subjects to make fewer mistakes in Question 3 than the Control group subjects, though the difference was significant only for the right shoulder down initial posture (p-value < 0.05, Kruskal-Wallis non-parametric test) (Table 4.12). While the SIM subjects had equal mean fractions incorrect for upright and right shoulder down initial postures, the Control subjects had their highest mean percent incorrect for the right shoulder down condition. A Friedman ranking of postures based on fraction incorrect was not significant for either group.

Question 3	*Control Group*		*SIM Group*		
Initial Posture	Mean	Std Err Mean	Mean	Std Err Mean	P-value
Upright	0.148	0.033	0.067	0.037	0.079
Left-shoulder down	0.241	0.093	0.117	0.050	0.295
Upside down	0.278	0.073	0.133	0.042	0.126
Right-shoulder down	0.315	0.090	0.067	0.037	0.019
All Postures	0.245	0.062	0.096	0.033	0.083

Table 4.12 Mean Fraction Incorrect for Question 3 Comparisons by Posture per Group (Kruskal-Wallis Non-parametric Test)

Unfortunately, analysis by targets did not reveal any significant differences nor provide insight into the learning methods used.

4.3.4 QUESTION 4: WHICH WAY TO TURN TO BECOME UPRIGHT?

In order to have correctly answered the last route description question (how to turn within the target module to be upright), subjects had to know the orientation of the target's vertical upright relative to the Control Module and incorporate the rotation due to their initial posture.

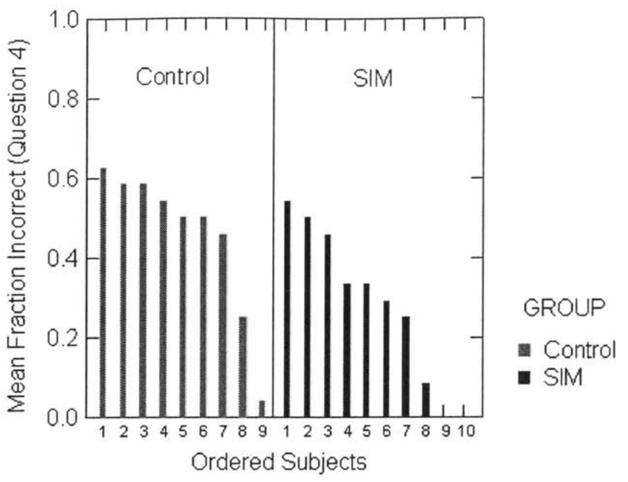


Figure 4.16 Mean Fraction Incorrect (Question 4: Turn in Target Module) per Group

It was thought that SIM subjects might make fewer mistakes answering this route description question because the global point of view of the entire model station, which SIM provided, would make it easier to notice which modules had differently oriented vertical uprights. The graph (Figure 4.16) shows little difference between group average fractions incorrect, but the corresponding Kruskal-Wallis non-parametric tests gave a significant result: SIM subjects had more correct answers than Control subjects (Table 4.13). SIM group had significantly lower mean fraction incorrect for Question 4 than the Control group in three initial postures (upright, upside down, and right shoulder down, ($p < 0.05$, Kruskal-Wallis test).

Question 4	*Control Group*		*SIM Group*		
Initial Posture	Mean	Std Err Mean	Mean	Std Err Mean	P-value
Upright	0.389	0.073	0.200	0.054	0.035
Left-shoulder down	0.426	0.097	0.350	0.094	0.525
Upside down	0.481	0.081	0.250	0.062	0.030
Right-shoulder down	0.519	0.059	0.317	0.076	0.053
All Postures	0.454	0.063	0.279	0.062	0.045

Table 4.13 Mean Fraction Incorrect for Question 4 by Posture Comparisons per Group (Kruskal-Wallis Non-parametric Test)

The effect of initial posture was analyzed for Question 4. As expected, upright orientation showed the lowest mean fraction incorrect. SIM subjects found left and right shoulder down the hardest postures to answer Question 4. Friedman rankings of postures by mean fraction incorrect for Question 4 were not significant for either group.

Figure 4.17 shows the distribution of the errors (90° roll left/right and roll 180°) made when answering Question 4 for each target. Most mistakes were roll 180°. It appears both the SIM and Control group had the fewest mistakes with the Experiment and EVA Modules, modules that share a vertical upright with the Control Module. SIM subjects appear to have made fewer mistakes than Control subjects for the modules that were perpendicular to Control Module, the Storage and Health Fitness Modules. Finally, both groups had the most mistakes for the Centrifuge and Habitation Modules, which were the ones that had vertical uprights inverted relative to the Control Module. It seems that inverted vertical uprights, regardless of treatment, were the hardest to learn.

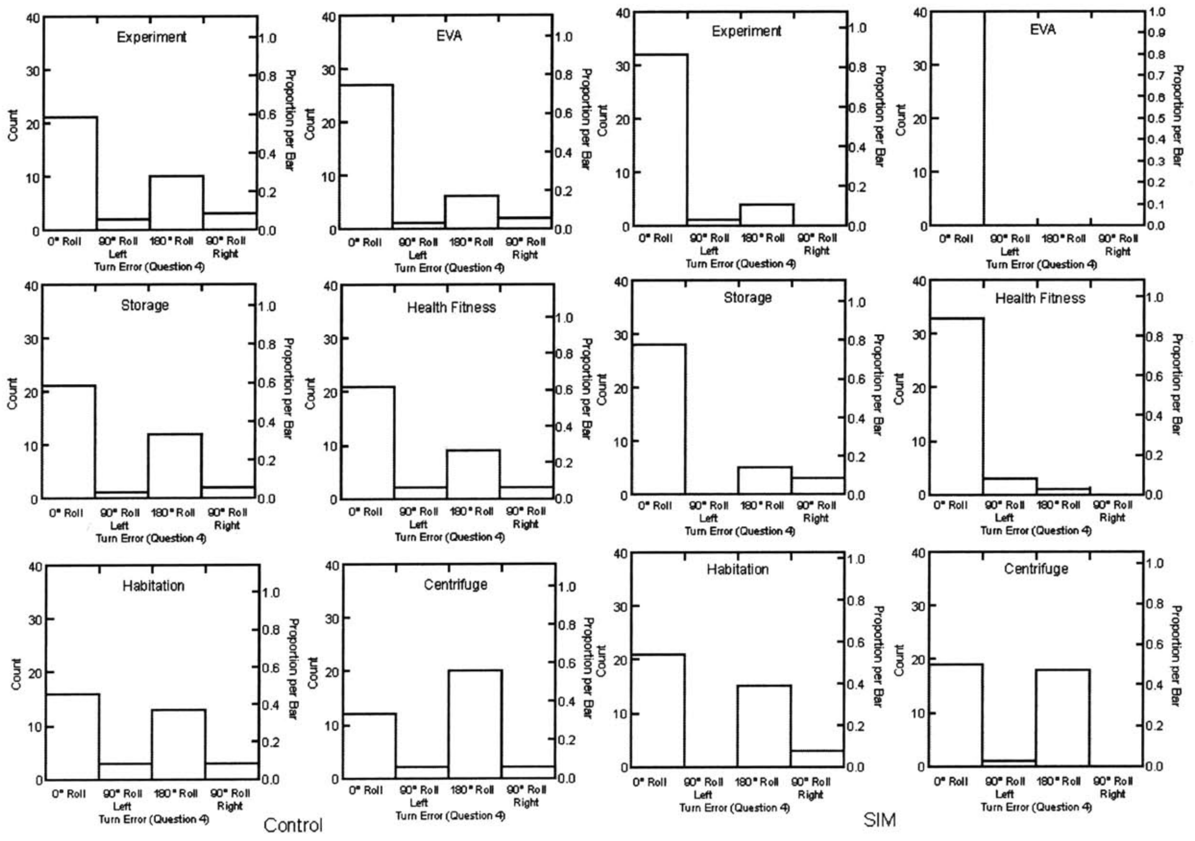


Figure 4.17 Distribution of Error Code for Question 4 by Target per Group

Comparing mean fraction incorrect by target for Question 4 did not reveal significant results except for the EVA Module ($p < 0.05$, Kruskal-Wallis non-parametric test), which was the target with the most correct answers for both groups. Based on mean fraction incorrect (see Table 4.14), the SIM subjects ranked the EVA and Health Fitness (blue side modules) best, followed by the Experiment and Storage (yellow). These modules have vertical uprights that were either the same or pitched relative to the Control's vertical upright. The Habitation and Centrifuge Modules, inverted 180° from the Control Module, ranked the lowest (worst performance). This result may indicate that learning modules that require roll, or more than one rotation, to reorient posture are harder than modules requiring one rotation, pitch or yaw.

Question 4			
Control Group		*SIM Group*	
Posture	Rank	Posture	Rank
EVA	22.5	EVA	21.5
Health Fitness	28.5	Health	29.5
Experiment	28.5	Experiment	32.5
Storage	29.5	Storage	38.0
Centrifuge	39.0	Centrifuge	43.5
Habitation	41.0	Habitation	45.0
P-Value	*0.161*		*0.044*

Table 4.14 Friedman Ranking of Targets by Mean Fraction Incorrect for Question 4 per Group

4.3.5 Response Time for Route Description Questions

Response times for route description questions were higher than RT for direction pointing because subjects had to answer a set of four questions (Figure 4.18 and Figure 4.19). On average, SIM subjects answered the route description questions slightly faster than Control subjects, and only significantly faster in the upright condition (Table 4.15, $p < 0.05$, Kruskal-Wallis and t-test). Friedman rankings of the postures by mean RT for questions showed a significant preference among all subjects for the upright posture (Table 4.16).

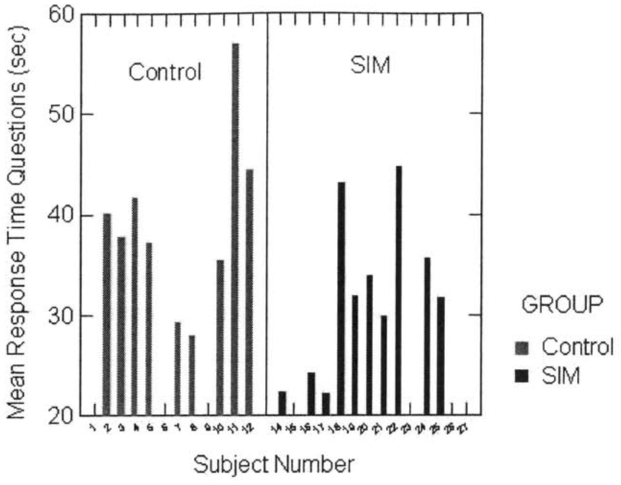


Figure 4.18 Mean Response Times for Route Description Questions per Group

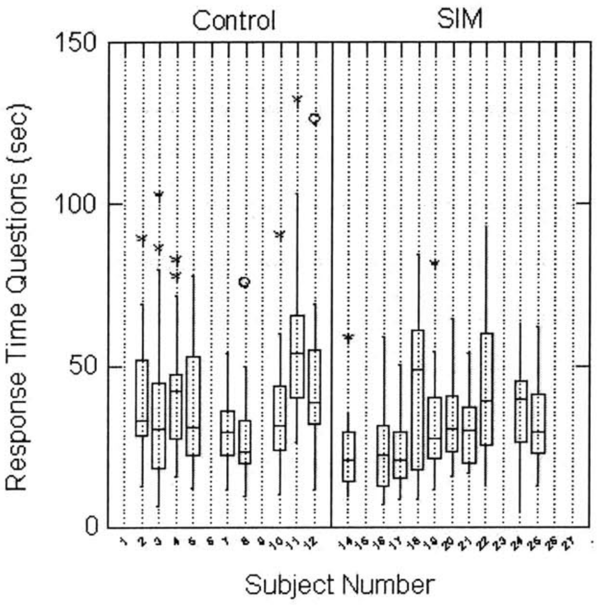


Figure 4.19 Distribution of RT for Route Description Questions per Group

RT All Questions	*Control Group*		*SIM Group*		*K-W*	*T-test*
Initial Posture	Mean	SD	Mean	SD	P-value	P-value
Upright	29.448	9.710	18.860	3.322	0.001	0.011
Left-shoulder down	43.066	10.083	38.739	11.823	0.327	0.401
Upside down	41.610	12.546	35.625	11.568	0.221	0.297
Right-shoulder down	41.774	10.851	34.674	8.488	0.165	0.136
All Postures	38.974	8.586	31.975	7.867	0.121	0.083

Table 4.15 Response Time for Questions by Posture Comparisons per Group (Kruskal-Wallis Non-parametric Test and T-test)

RT Questions				
Initial Posture	*Control*	*Rank*	*SIM*	*Rank*
	Upright	11.0	Upright	10.0
	Right Shldr Dwn	24.0	Right Shldr Dwn	27.0
	Left Shldr Dwn	27.0	Upside Dwn	29.0
	Upside Dwn	28.0	Left Shldr Dwn	34.0
P-value		*0.006*		*0.000*

Table 4.16 Friedman Ranking of Postures by RT for Questions per Group

Average response times by targets did not reveal significant differences between groups, but a significant Friedman ranking of RT by targets was found for SIM group alone. However, these results may be biased. Targets found through the blue (forward) hatch had shorter RT than targets found yellow hatch (back side of the station). Subjects would usually turn around, some a couple of times, to look at the yellow hatch when answering questions that related to its modules, which slowed their responses for these modules.

4.3.6 Summary of Route Description Task Results

The route description task was the measure of landmark and route knowledge. There were four route description questions (Table 3.4) that illustrate the identification of landmarks and turns necessary to reach a target module upright from the initial posture. Based on responses for each question, we can conclude: 1) most errors for the presented tasks were not made because subjects had misunderstood their initial posture; 2) most subjects were able to determine which modules were on which side (blue/yellow hatch) of the station; 3) there was no significant difference between SIM and Control group for making correct turns within a node to face target modules (except for trials that had right shoulder down initial posture); and 4) SIM group had significantly fewer mistakes than the Control when determining the correct turn to become upright within the target module, which indicated SIM subjects had more knowledge of the relationships of vertical uprights between modules.

4.4 EXIT QUESTIONNAIRES

All subjects were asked to fill out an exit questionnaire, in which they rated their experience and strategies used for the experiment. They could answer each statement in the questionnaire as "Very True" (score 0) through "Very False" (score 5). The questionnaire and mean answers for each statement can be found in Appendix D.

In comparing all subjects in both groups, only one statement produced a significant difference (t-test, $p < 0.05$, SIM = 1.86, Control = 3.21), and it favored the SIM training: "I felt confident I knew the layout of the station after Step 2". There were two other statements that we expected a difference for: "I had a mental map of the entire station with me inside" and "I had a miniature mental map of the entire station." Each described the hypothesized type of strategy each group, Control and SIM, respectively, would adopt. Instead, we found no uniformity within subjects in either group. Some in each group had an outside point of view and an inside point of view.

4.4.1 CONSTRUCTION OF STATION LAYOUT

All subjects were asked to construct a model of the station's layout. Figure 4.20 shows the distribution of trained subjects categorized by the accuracy of the location and orientation of the modules in the station. Only one Control subject was able to perfectly reproduce the layout of the station, while three SIM subjects succeeded at the task. It appears from the figure that SIM subjects had a more accurate mental representation of the locations and orientations of the modules as compared to Control subjects, though the results were not significant based on Kruskal-Wallis test analysis.

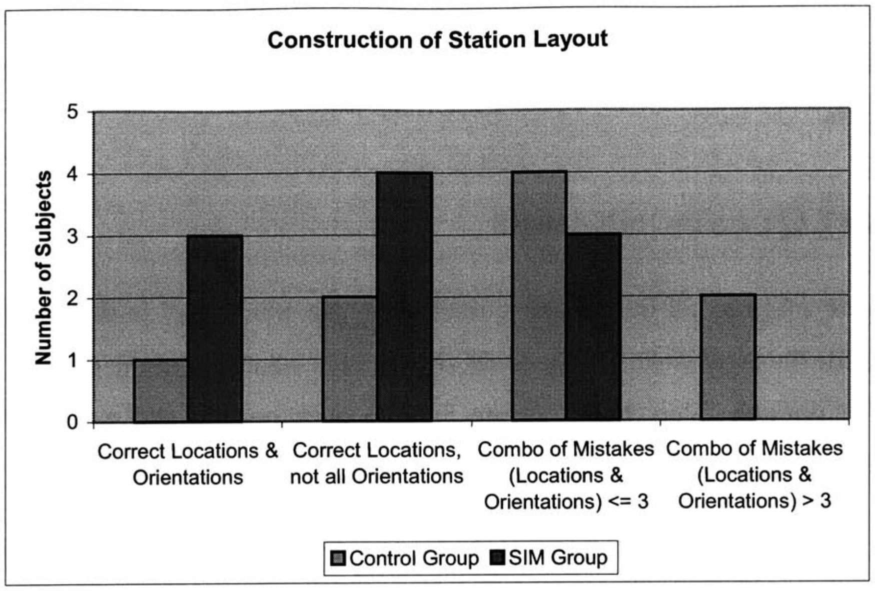


Figure 4.20 Distribution Subjects for Construction of Station Layout per Group

5 DISCUSSION AND CONCLUSIONS

Since about a third of the subjects in each group did not succeed at the training, they were omitted from the final analysis. While it was surprising to find that so many subjects found the task of learning the station's layout so difficult, another study that involved three-dimensional mental rotations found its task, similarly, too difficult for a third of the subject population (Richards 2000).

Based on previous research on map learning, we hypothesized SIM subjects would have better survey knowledge than the Control subjects, and the Control subjects would have better landmark and route knowledge. We also expected that map learning would result in an orientation-specific cognitive representation of the station. The results, however, support only the first hypothesis.

For the pointing task, the mean error angle and average response times were significantly lower for SIM subjects than Control subjects. Thus, subjects that were given the SIM as a tool to learn and navigate within the space station did better in pointing to modules than subjects who did not use SIM, indicating better survey knowledge for the SIM group. The pointing data and results also revealed that SIM subjects had a division of performance. They pointed more accurately to those targets that were in the forward direction (blue hatch modules) than those that were behind them (yellow hatch modules). During training instructions, description of the station as six modules, three of which were through the blue hatch, and three, through the yellow hatch, implied a "chunking" strategy that divided modules by hatch color. We suggest training with SIM helped users develop this "chunking" strategy to store their survey knowledge. There is weak evidence that Control subjects fully adopted this strategy.

Subjects that were given the SIM in the learning phase of the experiment did slightly better in describing routes to modules than subjects who did not use SIM, which contradicted our original landmark/route knowledge expectations. While there was no difference in performance between groups for route description questions 1 through 3, SIM subjects did perform significantly better than Control subjects for Question 4. We expected SIM subjects would be less dependent on landmark and routes, but their performance in the route description questions suggested that they were good at deducing routes based on their survey knowledge. It is possible also that the training SIM subjects received better prepared the SIM users to deduce answers to route description questions. This training included: 1) practicing the given tasks while viewing the miniature model of the station while still immersed in the space station environment; and 2) examining and "looking around" in the nodes after a "fly in", like the Control subjects did.

For the last route description question, which required the subjects to remember the relationship between target modules' visual vertical and the Control Module's, the SIM group's mean fraction incorrect was significantly lower than the Control groups'. By posture, means were all lower for the SIM group, and significantly different for every posture save the left shoulder down condition. The answer to this question could be considered the hardest task since it required the integration of up to three 90° rotations to determine the correct turn that would place the subject upright within the target module. Aside from understanding the initial orientation and the position of the target module, the orientation of the target relative to the Control Module must also be known. Trials that had target modules that were inverted relative to the Control Module were the hardest for both groups. However, SIM subjects understood the orientations of the pitched modules while the Control subjects did not seem as confident about those. Thus, SIM was effective at teaching relationships between the vertical uprights of many modules.

We suspected SIM subjects would have a mental representation of the station that was orientation-specific, but the results were mixed. Traditionally, orientation-specific mental maps result in response times that are linearly proportional to angle of rotation of initial posture (for example, Evans and Pezdek 1980 and Levine et al. 1982). Response times for SIM subjects did not exhibit that trait (Table 4.7), and furthermore, they were significantly shorter than the Control subjects' response times. Thus, based on traditional interpretations of response times, SIM subjects had an orientation-free mental map. However, based on pointing performance, SIM subjects seem to have a preference for facing the blue hatch when pointing, suggesting an orientation-specific mental representation. If subjects had no preference for initial posture, we would have expected to see equal response times for every initial posture but this was not seen for the SIM group. We must conclude that SIM subjects may not have necessarily acquired an orientation-free cognitive representation of the station, but it did allow for multiple point of views.

We speculate that one of the reasons SIM subjects had a mental representation that allowed for multiple point of views is that in the training step, subjects were able to view the miniature model in multiple orientations, which can contribute to orientation-free mental maps (MacEachren 1992; Evans and Pezdek 1980). The converse is also true. SIM subjects were orientation-specific towards the blue hatch of the station since they almost always studied the miniature model with only the blue hatch in sight. We can speculate then that while training, SIM subjects applied more 90° rolls rotations on the miniature model than 180° yaws, since there seems to be a lack of orientation-specificity in the roll direction.

Another result that supports the multiple point of views conclusion is the significantly more accurate performance (both in the pointing task and in Question 3) of SIM subjects for trials with right shoulder down initial posture. Visual exposure in Step 3 and training with the SIM in this non-upright initial posture in Step 2 gave SIM subjects an advantage over Control subjects in this initial posture, even though both groups received similar visual experiences. . In the third step of the experiment, there were more right shoulder down trials in the latter half, allowing subjects to visually familiarize themselves with non-upright postures. In the second step of the experiment, all subjects had experience with two postures: upright and right shoulder down. The right shoulder down training was incorporated into the second step in order to allow for some exposure to non-upright postures before testing. The task performance for other non-upright postures was not as significantly better than Control, conceivably due to the lack of training in those postures. For example, trained subjects seemed to find the upside down initial posture difficult, with highest mean error angles and longest response times.

A positive effect due to the use of SIM for learning the station's layout is remarkable, considering that SIM users had to also contend with the physical obstacle of successfully manipulating the sensor and game pad. The interface between the SIM and the user was not as natural ("user friendly") as anticipated, and thus SIM subjects had to concentrate on properly manipulating the SIM. Exit questions revealed that SIM subjects felt "the SIM tool was essential in learning the layout" (mean score: 0.29 on 0 to 5 scale, Appendix D) even though they found it moderately difficult to use.

The exit questionnaire also indicated SIM subjects were more confident about their acquired spatial representation of the station as compared to Control subjects. The model construction of their mental map suggested that more SIM subjects had a more accurate representation, with respect to locations and orientations of the modules. The Control subjects' inability to acquire a more accurate mental representation was hindered by the limited time available for training. Overall, these results parallel Satalich's (1995), in which she found that large-scale VR navigation can sometimes be hindered if moving about is too difficult and time-consuming.

The biggest constraint posed by the present experiment was its length. Training sessions accounted for most of the subject's time (about 1.5 hours) while the testing phase of the experiment was only 20 minutes, with time for only one repetition. Pilot trials showed that even with lengthier training sessions, not all subjects would be able to acquire a minimum knowledge of the layout. The experiment was limited in obtaining survey and route knowledge measurements relative to one module, the Control Module. While subjects felt they could "get from any module in the station to any other module", as their exit questionnaire revealed, it would be useful to understand if this training allowed subjects to acquire the level of survey and route knowledge that would let them navigate from any one module to another. Investigating this would necessitate testing with the pointing and route description tasks between all of the seven modules. This could be considered for follow on experiments.

Another future testing scenario for SIM could include the implementation of the ability to manipulate the avatar. Subjects, within this thesis experiment, differed in their preferred strategies, which posed a disadvantage to those who did not like to explore new environments by learning routes. If user-initiated avatar manipulation was to be implemented, subjects could adopt their own individual method of learning new environments. While this scenario encourages additional strategies for the acquisition of mental representations, it also makes each group population diverse in strategies, possibly too diverse to compare between groups.

While this experiment was aimed at investigating if the training with SIM was effective, further experiments are needed to study which aspects of SIM are the most useful: the avatar, the ability to rotate the miniature model, the ability to look inside the model, and/or the "fly in". For example, would SIM be as effective if the avatar did not show the user's orientation or if the avatar was not even there? Is the "fly in" necessary? Is it important that the SIM model be rotatable? Is it enough to have SIM as a stationary 3D map, and if so, how do these factors affect the acquisition of the mental representation?

This experiment has shown that in a couple of hours of training with SIM, many college students were able to learn to reliably identify seven individual modules. About two thirds of them were able to learn their spatial relationships when assembled into a space station, and then point and describe routes to them. Simple direct experience in complex environments is not always sufficient to develop the survey knowledge, as demonstrated by Mir crewmembers and in Moeser's 1-G experiments (1988). Though landmark/route knowledge is probably adequate for many routine day-to-day navigation tasks on ISS, some degree of survey knowledge of the ISS is arguably essential for dealing with emergency escape situations under conditions of reduced visibility, and/or in emergency situations, requiring crewmembers to make relative spatial judgments between modules. "Spacecraft in Miniature" could potentially be a useful tool in such training because it not only provides an interface to virtually navigate large, 3D virtual environments, but also is useful for the pre-flight acquisition of a relatively orientation-free mental representation of the large, complex 3D space station environment.

REFERENCES

Python, www.python.org. **Ver. 2.**

VRUT (Virtual Reality Utilities), www.recveb.ucsb.edu/vrut/tree/. **Ver. 2.4.**

Aoki, H., T. Yamaguchi and R. Ohno (2000). A Study of Orientation in a Zero Gravity Environment by means of Virtual Reality Simulation. Japan, Annual Meeting of Architectural Institute of Japan.

Bakker, N. H., P. J. Werkhoven and P. O. Passenier (1999). "The Effects of Proprioceptive and Visual Feedback on Geographical Orientation in Virtual Environments." Presence **8**(1): 36 - 53.

Barbour, C. G. and R. G. Coss (1988). "Differential Color Brightness as a Body Orientation Cue." Human Factors **30**: 713 - 717.

Bowman, D. A., E. T. Davis, L. F. Hodges and A. N. Badre (1999). "Maintaining Spatial Orientation during Travel in an Immersive Virtual Environment." Presence **8**(6): 618 - 631.

Bryant, D. J., M. Lanca and B. Tversky (1995). "Spatial Concepts and Perception of Physical and Diagrammed Scenes." Perceptual and Motor Skills **81**: 531 - 546.

Bryant, D. J. and B. Tversky (1999). "Mental Representations of Perspective and Spatial Relations From Diagrams and Models." Journal of Experimental Psychology: Learning, Memory, and Cognition **25**(1): 137 - 156.

Chance, S. S., F. Gaunet, A. C. Beall and J. M. Loomis (1998). "Locomotion Mode Affects the Updating of Objects Encountered during Travel: The Contribution of Vestibular and Proprioceptive Inputs to Path Integration." Presence **7**(2): 168 - 178.

Conover, W. J. (1999). Practical Nonparametric Statistics. New York, John Wiley & Sons, Inc.

Darken, R. P. and J. L. Sibert (1993). A Toolset for Navigation in Virtual Environments. ACM User Interface Software & Technology.

Darken, R. P. and J. L. Sibert (1996). "Navigating Large Virtual Spaces." International Journal of Human-Computer Studies **8**(1): 49 - 72.

Evans, G. W. and K. Pezdek (1980). "Cognitive Mapping: Knowledge of Real-World Distance and Location Information." Journal of Experimental Psychology: Human Learning and Memory **6**(1): 13 - 24.

Franklin, N. and B. Tversky (1990). "Searching Imagined Environments." Journal of Experimental Psychology: General **119**(1): 63 - 76.

Golledge, R. G., V. Dougherty and S. Bell (1995). "Acquiring Spatial Knowledge: Survey Versus Route-Based Knowledge in Unfamiliar Environments." Annals of the Association of American Geographers **85**(1): 135 - 158.

Howard, I. P. (1982). Human Visual Orientation. Chichester, Sussex, John Wiley.

Howard, I. P., S. S. Bergstrom and M. Ohmi (1990). "Space From Shading in Different Frames of Reference." Perception **23**: 523 - 530.

Howard, I. P. and G. Hu (2001). "Visually Induced Reorientation Illusions." Perception **30**: 583 - 600.

Johnson_Engineering (2000). Proposal for ISS IVA Colors. G. Finney. Houston, TX, Habitability Design Center.

Klatzky, R. L., J. M. Loomis, A. C. Beall, S. S. Chance and R. G. Golledge (1998). "Spatial Updating of Self-Position and Orientation during Real, Imagined, and Virtual Locomotion." Psychological Science **9**(4): 293 - 298.

Lathan, C. and G. Clement (1997). Response of the Neurovestibular System to Spaceflight. Fundamentals of Space Life Sciences. S. E. Churchill. Malabar, FL, Krieger Publishing Co. **2**: 65 - 82.

Lathrop, W. B. and M. K. Kaiser (2002, in press). "Perceived Orientation in Physical and Virtual Environments: Changes in Perceived Orientation as a Function of Idiothetic Information Available." Presence.

Levine, M., I. N. Jankovic and M. Palij (1982). "Principles of Spatial Problem Solving." Journal of Environmental Psychology: General **111**(2): 157 - 175.

MacEachren, A. M. (1992). "Learning Spatial Information from Maps: Can Orientation-Specificity Be Overcome?" Professional Geographer **44**(4): 431 - 443.

McNamara, T. P. (1986). "Mental Representations of Spatial Relations." Cognitive Psychology **18**: 87 -121.

Mittelstaedt, H. (1983). "A New Solution to the Problem of the Subjective Vertical." Naturwissenschaften **70**: 272 - 281.

Mittelstaedt, H. (1996). Inflight and Postflight Results on the Causation of Inversion Illusions and Space Sickness:Scientific Results of the German Spacelab Mission D1. Wissenshaftliche Projecktfuhrung D1/DFVLR. Koln, Germany.

Moeser, S. D. (1988). "Cognitive Mappping in a Complex Building." Environment and Behavior **20**(1): 21 - 49.

NASA (1996). International Space Station Interior Color Scheme: SSP 50008, Revision B. Houston, TX, Space Station Program Office.

NASA (1998). International Space Station Familiarization. Houston, TX, Web: www.spaceflight.nasa.gov/spacenews/factsheets/pdfs/td9702.pdf.

Oman, C. M. (2000). Human Visual Orientation in Weightlessness. York Conference 2001: Levels of Perception, Springer Verlag.

Oman, C. M., B. K. Lichtenberg, K. E. Money and R. K. McCoy (1986). "MIT/Canadian Vestibular Experiments on the Spacelab-1 Mission: 4. Space Motion Sickness: Symptoms, Stimuli, and Predictability." Experimental Brain Research **64**: 316 - 334.

Oman, C. M., L. R. Young, D. G. D. Watt, K. E. Money, B. K. Lichtenberg, R. V. Kenyon and A. P. Arrott (1988). MIT/Canadian Spacelab Experiments on Vestibular Adaptation and Space Motion Sickness. Basic and Applied Aspects of Vestibular Function. J. C. Hwang, N. G. Daunton and V. J. Wilson. Hong Kong, Hong Kong University Press.

O'Neill, M. J. (1992). "Effects of Familiarity and Plan Complexity on Wayfinding in Simulated Buildings." Journal of Environmental Psychology **12**: 319 - 327.

Pausch, R., T. Burnette, D. Brockway and M. E. Weiblen (1995). Navigation and Locomotion in Virtual Worlds via Flight into Hand-Held Miniatures, SIGGRAPH Conferences. **1999**.

Presson, C. C. and M. D. Hazelrigg (1984). "Building Spatial Representations Through Primary and Secondary Learning." Journal of Experimental Psychology: Learning, Memory, and Cognition **10**(4): 716 - 722.

Richards, J. T. (2000). Three-dimensional Spatial Learning in a Virtual Space Station Node. Dept. of Aeronautics and Astronautics. Cambridge, MA, Massachusetts Institute of Technology: 1 - 144.

Richards, J. T., J. B. Clark, C. M. Oman and T. H. Marshburn (2001). Neurovestibular Effects of Long-Duration Spaceflight: A Summary of Mir Phase 1 Experiences: 1-33.

Richardson, A., D. R. Montello and M. Hegarty (1999). "Spatial Knowledge Acquisition from Maps and from Navigation in Real and Virtual Environments." Memory and Cognition **27**(4): 741 - 750.

Ross, H. E., S. D. Crickman, N. V. Sills and E. P. Owen (1969). "Orientation to the Vertical in Free Divers." Aerospace Medicine **40**: 485 - 494.

Rossano, M. J., S. O. West, T. J. Robertson, M. C. Wayne and R. B. Chase (1999). "The Acquisition of Route and Survey Knowledge from Computer Models." Journal of Environmental Psychology **19**: 101 - 115.

Satalich, G. A. (1995). Navigation and Wayfinding in Virtual Reality: Finding Proper Tools and Cues to Enhance Navigation Awareness, University of Washington.

Stoakley, R., M. J. Conway and R. Pausch (1995). Virtual Reality on a WIM: Interactive Worlds in Miniature, CHI Conference. **1999**.

Thorndyke, P. W. and B. Hayes-Roth (1982). "Differences in Spatial Knowledge Acquired from Maps and Navigation." Cognitive Psychology **14**: 560 - 589.

Tlauka, M. and P. N. Wilson (1996). "Orientation-free Representations from Navigation through a Computer-Simulated Environment." Environment and Behavior **28**(5): 647 - 664.

Witkin, H. A., P. Oltman, E. Raskin and S. Karp (1971). Manual for the Embedded Figures Tests. Palo Alto, CA, Consulting Psychologists Press.

Witmer, B. G., J. H. Bailey and B. W. Knerr (1996). "Virtual Spaces and Real World Places: Transfer of Route Knowledge." International Journal of Human-Computer Studies **45**: 413 - 428.

World Wide Web:

www.spaceflight.nasa.gov

www.nasa.gov.

APPENDIX A: PROGRAMMING CODE

Included in this appendix are programming for the set up, Step 1, Step 2 (SIM and Control), and Step 3 programs.

Code also in CD version copy of this thesis, which will be stored in the Man-Vehicle Lab.

Set-Up Program

Used to familiarize subject with environment and practice ratcheting

```
import vrut
import sid

vrut.go(vrut.HMD | vrut.STEREO)
#vrut.framerate()
module          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\PracticeModule.wrl')
lilmod          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\PracticeModule.wrl', vrut.HEAD)
lilmod.scale(.05,.05,.05)
lilmod.translate(0, -.1, .745)
teapot          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Teapotgreen.wrl', vrut.HEAD)
teapot.scale(.05,.05,.05)
teapot.translate(0, -.1, .5)
teapot.curtain(vrut.CLOSE)

# Sensor              SELECT THE RIGHT HEAD TRACKER!
tracker = vrut.addsensor('isensemulti_beta2') # just rotations??
vrut.tracker()
tracker.command(6) # RESET1    ... see multi.py in AndyL
tracker.reset()

wand = vrut.addsensor('isensemulti_beta2')
tracker.command(7)
tracker.reset()

on = 1
off = 0

global movflag, first
movflag = off
first = on

def myKeyboard(key):
   global first, movflag

   if key == ' ':
      if movflag == on:
         movflag = off
         tracker.command(5)
         if first == on:
            first = off
            tracker.command(4)
      elif movflag == off:
         tracker.command(7)
         tracker.reset()
         movflag = on
         vrut.starttimer(0.001)

   elif key == 'r':
      tracker.command(6) # RESET1    ... see multi.py in AndyL
      tracker.reset()

def mytimer(timernum):
   global movflag, first

   if timernum == 1:
      if movflag == on:
         data = wand.get()
         lilmod.translate(data[0],-.1+data[1],.5+data[2])
         lilmod.rotate(-data[3],-data[4],data[5],data[6]*57.296)
         vrut.starttimer(1,0.001)
   elif timernum == 2:
      if sid.buttons() == sid.BUTTON8 and movflag == off:
         tracker.command(7)
         tracker.reset()
         movflag = on
         vrut.starttimer(1,0.001)
      elif sid.buttons() == sid.BUTTON8 and movflag == on:
         movflag = off
         tracker.command(5)
         if first == on:
            first = off
            tracker.command(4)
      vrut.starttimer(2,0.3)


vrut.callback(vrut.KEYBOARD_EVENT, 'myKeyboard')
vrut.callback(vrut.TIMER_EVENT,'mytimer')
vrut.starttimer(2,0.001)
```

Step 1 : Learning Individual Module Landmarks

This program was the same for Control and SIM groups

```
import vrut
import time
import math
import types
import string
import sid

# 4/1/2001 this program will allow user to view each module.
# 5/27/2001 implementing the wand into the viewing

#
*****************************************************************
*********************
# KEY:
#
#  8 (up) == travel up a module
#  2 (down) == travel down a module
#  4 (left) == scroll left names
#  6 (right) == scroll right names
#  5 == answer button
#  0 == pop up module name
#
*****************************************************************
*********************

#    1                          direction nomenclature
#    |
# 2 ----- 0 into the page --  4
#    |                 out of the page -- 5
#    3
#
# Module labelling:
```

```
#           0: Columbus, European Module
#           1: Stowage Module
#           2: Zarya, Russian Control Module
#           3: Destiny, U.S. Module
#           4: Habitation Module
#           5: Kibo, Japanese Module
#           6: Zvezda, Russian Service Module
#  7: Node (generic)

#                                                      Constants
**************************************************************
************


on = 1
off = 0
scfac = 0.075
speed = 0.25      # flyin speed in the z
rate = 0.05

# times
time1 = 40   #40
time2 = 7   #7
time3 = 3

RAD2DEG = 57.295779513082323
DEG2RAD = 1/57.295779513082323

# Timernums
starting = 1
next = 2
test = 3
place = 4
flip = 5
sound = 6
waitENTER = 7
moving = 8

global count,testing, index, right, arrow, viewing, bflag
global namepops0, namepops1 ,namepops2 ,namepops3 ,namepops4
,namepops5, namepops6, namepops7
count = 1
testing = 0
index = 0
right = 0
arrow = 0
bflag = [0,off]
viewing = on   # flag
namepops0 = 0
namepops1 = 0
namepops2 = 0
namepops3 = 0
namepops4 = 0
namepops5 = 0
namepops6 = 0
namepops7 = 0

#
**************************************************************
*********************

#                                                      Variables
**************************************************************
************

vrut.go(vrut.HMD | vrut.STEREO)
#vrut.go()

# Sensor              SELECT THE RIGHT HEAD TRACKER!
tracker = vrut.addsensor('isensemulti_beta')   # just rotations??
vrut.tracker()
tracker.command(6)  # RESET1    ... see multi.py in AndyL
tracker.reset()

# Objects
# for testing purposes:  need to update station2.wrl  4/18/2001
station            =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Station2_node.wrl')
#station           =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Station_astr_obj.wrl')
columbus           =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Columbus.wrl',vrut.HEAD)
stowage            =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Stowage.wrl',vrut.HEAD)
zarya              =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Zarya.wrl',vrut.HEAD)
zvezda             =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Zvezda.wrl',vrut.HEAD)
habitation         =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Habitation.wrl',vrut.HEAD)
kibo               =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Kibo.wrl',vrut.HEAD)
destiny            =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Destiny.wrl',vrut.HEAD)
node               =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Node.wrl',vrut.HEAD)
correct            =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Correct.wrl',vrut.HEAD)
incorrect          =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Incorrect.wrl',vrut.HEAD)
testphase          =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Testphase.wrl',vrut.HEAD)
notest             =              vrut.addchild('n:\Jessica\Thesis
SIM\Models\Notest.wrl',vrut.HEAD)
columbus.scale(0.25,.25,.25)
stowage.scale(0.25,.25,.25)
zarya.scale(0.25,.25,.25)
zvezda.scale(0.25,.25,.25)
habitation.scale(0.25,.25,.25)
kibo.scale(0.25,.25,.25)
destiny.scale(0.25,.25,.25)
node.scale(0.25,.25,.25)
correct.scale(0.25,.25,.25)
incorrect.scale(0.25,.25,.25)
columbus.translate(0,-.1,.75)
stowage.translate(0,-.1,.75)
zarya.translate(0,-.1,.75)
zvezda.translate(0,-.1,.75)
habitation.translate(0,-.1,.75)
kibo.translate(0,-.1,.75)
destiny.translate(0,-.1,.75)
node.translate(0,-.1,.75)
correct.translate(0,-.1,.5)
incorrect.translate(0,-.1,.5)
testphase.translate(0,-.4,1.25)
notest.translate(0,-.4,1.25)
columbus.curtain(vrut.CLOSE)
stowage.curtain(vrut.CLOSE)
zarya.curtain(vrut.CLOSE)
zvezda.curtain(vrut.CLOSE)
habitation.curtain(vrut.CLOSE)
kibo.curtain(vrut.CLOSE)
destiny.curtain(vrut.CLOSE)
node.curtain(vrut.CLOSE)
correct.curtain(vrut.CLOSE)
incorrect.curtain(vrut.CLOSE)
testphase.curtain(vrut.CLOSE)
notest.curtain(vrut.CLOSE)
```

```
# Trial number
global module, curtain
module = 0

# Init location for each module: [x,y,z] coordinates relative to center
(which is mid Zvezda)
# Init orientation for floor/ceiling are "correct" (nose,feet)
# Max length of module to be traversed from start
startpos     =     [[-1,0,-5.25],[-1,0,5.25],[0,-1,-5.25],[0,1,5.25],[1,0,-
5.25],[1,0,5.25],[0,0,-4.25],[0,0,7.30]]
startori = [[2,3],[2,3],[3,4],[1,4],[0,1],[0,1],[4,3],[4,3]]
length = [[-5,0,0],[-5,0,0],[0,-5,0],[0,5,0],[5,0,0],[5,0,0],[0,0,7.5],[0,0,4]]

testord                                                          =
[[2,5,7,1,6,0,4,3],[3,4,0,6,1,7,5,2],[7,0,6,5,3,1,4,2],[2,4,1,3,5,6,0,7],[5,2,1,
4,6,0,7,3],[3,7,0,6,4,1,2,5]]

#
*****************************************************************
**********************

def closeAll():
   columbus.curtain(vrut.CLOSE)
   stowage.curtain(vrut.CLOSE)
   zarya.curtain(vrut.CLOSE)
   zvezda.curtain(vrut.CLOSE)
   habitation.curtain(vrut.CLOSE)
   kibo.curtain(vrut.CLOSE)
   destiny.curtain(vrut.CLOSE)
   node.curtain(vrut.CLOSE)

def defAngles(nose,feet):
   # print(nose,feet)
   # these are defined as if starting from original orientation
   # 4/1/2001 I had to edit this for viewmodules program...
   if nose == 2 or nose == 0:
      pitch = 0
      if nose == 2: yaw = -90
      else: yaw = 90
      if feet == 4:
         if nose == 2:
            roll = 90
         else: roll = -90
      if feet == 1:
         roll = 180
      if feet == 5:
         if nose == 2:
            roll = -90
         else: roll = 90
      if feet == 3:
         roll = 0

   if nose == 3 or nose == 1:
      yaw = 0
      if nose == 3: pitch = 90      # changed the sign of the pitch
4/1/2001
      else: pitch = -90
      if feet == 5:
         if nose == 3:
            roll = 0
         else: roll = 180
      if feet == 0:
         roll = 90
      if feet == 4:
         if nose == 3:
            roll = 180
         else: roll = 0
      if feet == 2:
         roll = -90
   if nose == 4 or nose == 5:
      pitch = 0
      if nose == 4: yaw = 0
      else: yaw = 180
      if feet == 3:
         roll = 0
      if feet == 1:
         roll = 180
      if feet == 0:
         if nose == 4:
            roll = 90
         else: roll = -90
      if feet == 2:
         if nose == 4:
            roll = -90
         else: roll = 90
   return [yaw,pitch,roll]

def showTag(module):
   global curtain
   global   namepops0,   namepops1   ,namepops2   ,namepops3
,namepops4 ,namepops5, namepops6, namepops7
   if module == 0:
      if curtain == off:
         columbus.curtain(vrut.OPEN)
         curtain = on
         namepops0 = namepops0 + 1
      else:
         columbus.curtain(vrut.CLOSE)
         curtain = off
   if module == 1:
      if curtain == off:
         stowage.curtain(vrut.OPEN)
         curtain = on
         namepops1 = namepops1 + 1
      else:
         stowage.curtain(vrut.CLOSE)
         curtain = off
   if module == 2:
      if curtain == off:
         zarya.curtain(vrut.OPEN)
         curtain = on
         namepops2 = namepops2 + 1
      else:
         zarya.curtain(vrut.CLOSE)
         curtain = off
   if module == 3:
      if curtain == off:
         destiny.curtain(vrut.OPEN)
         curtain = on
         namepops3 = namepops3 + 1
      else:
         destiny.curtain(vrut.CLOSE)
         curtain = off
   if module == 4:
      if curtain == off:
         habitation.curtain(vrut.OPEN)
         curtain = on
         namepops4 = namepops4 + 1
      else:
         habitation.curtain(vrut.CLOSE)
         curtain = off
   if module == 5:
      if curtain == off:
         kibo.curtain(vrut.OPEN)
         curtain = on
         namepops5 = namepops5 + 1
      else:
            kibo.curtain(vrut.CLOSE)
            curtain = off
    if module == 6:
        if curtain == off:
            zvezda.curtain(vrut.OPEN)
            curtain = on
            namepops6 = namepops6 + 1
        else:
            zvezda.curtain(vrut.CLOSE)
            curtain = off
    if module == 7:
        if curtain == off:
            node.curtain(vrut.OPEN)
            curtain = on
            namepops7 = namepops7 + 1
        else:
            node.curtain(vrut.CLOSE)
            curtain = off

def myKeyboard(key):
    if key == ' ':
        vrut.starttimer(waitENTER,rate)
        vrut.starttimer(starting,0.001)
        station.curtain(vrut.OPEN)
    if key == 'r':
        tracker.command(6) # RESET1
        tracker.reset()

def mytimer(timernum):
    global module, curtain, count, testing, index, arrow, right
    global dist, xflag, yflag, zflag, viewing
    global  namepops0,  namepops1  ,namepops2  ,namepops3
,namepops4 ,namepops5, namepops6, namepops7

    if timernum == starting:
        viewing = on
        notest.curtain(vrut.CLOSE)
        vrut.reset(vrut.BODY_ORI)
        vrut.reset(vrut.HEAD_POS)
        ori = defAngles(startori[module][0],startori[module][1])

vrut.translate(vrut.HEAD_POS,startpos[module][0],startpos[module]
[1],startpos[module][2])
        vrut.rotate(vrut.BODY_ORI, ori[0],ori[1],ori[2])
        dist = 0
        curtain = off
        closeAll()
        xflag = off; yflag = off; zflag = off
        if length[module][0] <> 0: xflag = on
        elif length[module][1] <> 0: yflag = on
        elif length[module][2] <> 0: zflag = on
        if count == 1:
            vrut.starttimer(sound,time1)
            vrut.starttimer(next,time1)  # should be time1
        elif count == 2:
            vrut.starttimer(sound,time2)
            vrut.starttimer(next,time2)  # should be time2
        elif count == 3:
            vrut.starttimer(test,0.001)
        elif count == 4:
            vrut.starttimer(sound,time2)
            vrut.starttimer(next,time2)  # should be time2
        elif count == 5:
            vrut.starttimer(sound,time3)
            vrut.starttimer(next,time3)
        elif count == 6:
            vrut.starttimer(test,0.001)

    elif timernum == next:
        module = module + 1
        if module < 8:
            vrut.starttimer(starting,0.001)
        else:
            if count < 7:
                print('starting again')
                module = 0
                count = count + 1
                vrut.starttimer(starting,0.001)
            else:
                print('the end')
                vrut.translate(vrut.HEAD_POS, 0,100,0)

    elif timernum == test:
        curtain = off
        closeAll()
        testphase.curtain(vrut.OPEN)
        #viewing = off
        index = 0
        right = 0
        out              =            '\n\n'            +
'Test'+'\t'+str(testord[testing][0])+'\t'+str(testord[testing][1])+'\t'+st
r(testord[testing][2])+'\t'+str(testord[testing][3])+'\t'+str(testord[testi
ng][4])+'\t'+str(testord[testing][5])+'\t'+str(testord[testing][6])+
'\t'+str(testord[testing][7])+ '\t'+'Correct'+'\n'
        file.write(out)
        file.flush()
        out = str(testing+1) + '\t'
        file.write(out)
        file.flush()
        vrut.starttimer(place,2)

    elif timernum == place:
        testphase.curtain(vrut.CLOSE)
        viewing = off
        correct.curtain(vrut.CLOSE)
        incorrect.curtain(vrut.CLOSE)
        print('index',index)
        print('right',right)
        if index < 8:
            where = testord[testing][index]
            vrut.reset(vrut.BODY_ORI)
            vrut.reset(vrut.HEAD_POS)
            ori = defAngles(startori[where][0],startori[where][1])

vrut.translate(vrut.HEAD_POS,startpos[where][0],startpos[where][1],
startpos[where][2])
            vrut.rotate(vrut.BODY_ORI, ori[0],ori[1],ori[2])
            index = index + 1
        else:
            if right == 8:
                print('the end')
                out = str(right) + '\n'
                file.write(out)
                file.flush()
                out                                              =
str(namepops0)+'\n'+str(namepops1)+'\n'+str(namepops2)+'\n'+st
r(namepops3)+'\n'+str(namepops4)+'\n'+str(namepops5)+'\n'+str(
namepops6)+'\n'+str(namepops7)+'\n'
                file.write(out)
                file.flush()
                file.close()
                vrut.translate(vrut.HEAD_POS,0,100,0)
            else:
                out = str(right) + '\n'
                file.write(out)
                file.flush()
                testing = testing + 1
                count = 4
            closeAll()
            notest.curtain(vrut.OPEN)
            vrut.starttimer(starting,2)

    elif timernum == sound:
          vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\Reminder.wav')

    elif timernum == waitENTER:
        #print('waiting for keys')
        if count == 1 or count == 2 or count == 4 or count == 5:
            if viewing == on:  # ??
                x,y,ang = sid.get()
                if sid.buttons() != 0:
                    bflag[0] = sid.buttons()
                    bflag[1] = on
                elif sid.buttons() == 0 and bflag[1] == on:
                    bflag[1] = off
                    if bflag[0] == sid.BUTTON9:
                        showTag(module)
                if round(y,2) == 1.0:
                    if xflag == on:
                        if length[module][0] > 0 and dist < length[module][0]:
                            vrut.translate(vrut.HEAD_POS, 0.1, 0, 0)
                            dist = dist + 0.1
                        elif    length[module][0]    <    0    and    dist    >
length[module][0]:
                            vrut.translate(vrut.HEAD_POS, -0.1, 0, 0)
                            dist = dist - 0.1
                    if yflag == on:
                        if length[module][1] > 0 and dist < length[module][1]:
                            vrut.translate(vrut.HEAD_POS, 0, 0.1, 0)
                            dist = dist + 0.1
                        if length[module][1] < 0 and dist > length[module][1]:
                            vrut.translate(vrut.HEAD_POS, 0, -0.1, 0)
                            dist = dist - 0.1
                    if zflag == on:
                        if length[module][2] > 0 and dist < length[module][2]:
                            vrut.translate(vrut.HEAD_POS, 0, 0, 0.1)
                            dist = dist + 0.1
                        if length[module][2] < 0 and dist > length[module][2]:
                            vrut.translate(vrut.HEAD_POS, 0, 0, -0.1)
                            dist = dist - 0.1
                if round(y,2) == -1.0:
                    if xflag == on:
                        if length[module][0] > 0 and dist > 0:
                            vrut.translate(vrut.HEAD_POS, -0.1, 0, 0)
                            dist = dist - 0.1
                        if length[module][0] < 0 and dist < 0:
                            vrut.translate(vrut.HEAD_POS, 0.1, 0, 0)
                            dist = dist + 0.1
                    if yflag == on:
                        if length[module][1] > 0 and dist > 0:
                            vrut.translate(vrut.HEAD_POS, 0, -0.1, 0)
                            dist = dist - 0.1
                        if length[module][1] < 0 and dist < 0:
                            vrut.translate(vrut.HEAD_POS, 0, 0.1, 0)
                            dist = dist + 0.1
                    if zflag == on:
                        if length[module][2] > 0 and dist > 0:
                            vrut.translate(vrut.HEAD_POS, 0, 0, -0.1)
                            dist = dist - 0.1
                        if length[module][2] < 0 and dist < 0:
                            vrut.translate(vrut.HEAD_POS, 0, 0, 0.1)
                            dist = dist + 0.1
        elif count == 3 or count == 6:
            if viewing == off:
                x,y,ang = sid.get()
                if sid.buttons() != 0:
                    bflag[0] = sid.buttons()
                    bflag[1] = on
                elif sid.buttons() == 0 and bflag[1] == on:
                    print('bflag',bflag)
                    bflag[1] = off
                    if bflag[0] == sid.BUTTON9:
                        bflag[0] = sid.buttons()
                        closeAll()
                        print(arrow, testord[testing][index-1])
                        if arrow == testord[testing][index-1]:
                            right = right + 1
                            correct.curtain(vrut.OPEN)
                        else:
                            incorrect.curtain(vrut.OPEN)
                        out = str(arrow)+'\t'
                        file.write(out)
                        file.flush()
                        vrut.starttimer(place,1)
                elif bflag[0] == sid.BUTTON1:
                    bflag[0] = sid.buttons()
                    if arrow == 7:
                        arrow = 0
                    else:
                        arrow = arrow + 1
                    closeAll()
                    if arrow == 0:
                        columbus.curtain(vrut.OPEN)
                    if arrow == 1:
                        stowage.curtain(vrut.OPEN)
                    if arrow == 2:
                        zarya.curtain(vrut.OPEN)
                    if arrow == 3:
                        destiny.curtain(vrut.OPEN)
                    if arrow == 4:
                        habitation.curtain(vrut.OPEN)
                    if arrow == 5:
                        kibo.curtain(vrut.OPEN)
                    if arrow == 6:
                        zvezda.curtain(vrut.OPEN)
                    if arrow == 7:
                        node.curtain(vrut.OPEN)
                elif bflag[0] == sid.BUTTON4:
                    bflag[0] = sid.buttons()
                    if arrow == 0:
                        arrow = 7
                    else:
                        arrow = arrow - 1
                    closeAll()
                    if arrow == 0:
                        columbus.curtain(vrut.OPEN)
                    if arrow == 1:
                        stowage.curtain(vrut.OPEN)
                    if arrow == 2:
                        zarya.curtain(vrut.OPEN)
                    if arrow == 3:
                        destiny.curtain(vrut.OPEN)
                    if arrow == 4:
                        habitation.curtain(vrut.OPEN)
                    if arrow == 5:
                        kibo.curtain(vrut.OPEN)
                    if arrow == 6:
                        zvezda.curtain(vrut.OPEN)
                    if arrow == 7:
                        node.curtain(vrut.OPEN)
    vrut.starttimer(waitENTER,rate)
```

```
#
*****************************************************************************************

vrut.translate(vrut.HEAD_POS,0,-1.82, 0) # not needed after head
tracker is connected

#
*****************************************************************************************

vrut.callback(vrut.KEYBOARD_EVENT, 'myKeyboard')
vrut.callback(vrut.TIMER_EVENT,'mytimer')

# Setting up output files
exp_time = time.localtime(time.time())
timestring = time.strftime('%b%d_%H%M',exp_time)
filename          =          'N:\Jessica\Thesis
SIM\Data\Sim\Phase1\Phase1_SIM'+str(timestring)+'.dat'
#filename         =          'N:\Jessica\Thesis
SIM\Data\Cntrl\Phase1\Phase1_Cntrl'+str(timestring)+'.dat'
file = open(filename,'w')

#vrut.starttimer(starting,0.001)
station.curtain(vrut.CLOSE)
```

Step 2: Learning Layout of the Entire Station via Routes

Control Group (Parts A & B)

```
import vrut
import time
import math
import types
import string
import sid

# 6/4/2001 This is adapting trainingCntrl_ver3 to the new protocol.
# 6/6/2001 done
# 6/23/2001 new protocol implementation. deviding phase 2 into
two parts.

#
*****************************************************************************************
# KEY:
#
*****************************************************************************************

#   1                       direction nomenclature
#   |
# 2 ----- 0 into the page --  4
#   |               out of the page -- 5
#   3
#
# Target difficulty:
#       0: modules 0 or 1
#       1: modules 4 or 5
#       2: modules 2 or 3
# FarNode:      I  MIGHT  HAVE  GOTTEN  THIS
BACKWARDS... 4/2/2001
#    0: modules 1, 3, or 5
#       1: modules 0, 2, or 4
# Posture:  relative to module 6
#       0: roll, with nose at 4 & feet at 1
#       1: pitch, (positive) with nose at 1 & feet at 4
#       2: roll, with nose at 4 & feet at 2
#       3: pitch, (negative) with nose at 3 & feet at 5
#
# Module labelling:
#       0: Columbus, European Module      == Experiment
#       1: Stowage Module                 == EVA
#       2: Zarya, Russian Control Module  == Storage
#       3: Destiny, U.S. Module           == Health Fitness
#       4: Habitation Module              == Habitation
#       5: Kibo, Japanese Module          == Centrifuge
#       6: Zvezda, Russian Service Module == Control
#       Nodes
#       7: FarNode 1, hatch at end of zvezda (blue)        #
6/4/2001 had these reversed
#       8: FarNode 0, hatch at other end of zvezda (yellow)
#
#                                              Constants
****************************************************************************
***********

on = 1
off = 0
flytime = 100
speed = 1
maxtrials = 1   # double check!
rate = 0.3

qax = .3
qay = .435
qby = .15
qcy = -.15
qdy = -.435
qaz = 2.05
lay = .275
lby = 0
lcy =-.275
ldy = -.57
lz  = 2.0
ansscfac = 4
scfac = 2.25

HMD = on

RAD2DEG = 57.295779513082323
DEG2RAD = 1/57.295779513082323

#
*****************************************************************************************

if HMD == on:
   vrut.go(vrut.HMD | vrut.STEREO)
else:
   vrut.go(vrut.CONSOLE)
   vrut.translate(vrut.HEAD_POS, 0,-1.82,0)

#
*****************************************************************************************

# Sensor
tracker = vrut.addsensor('isensemulti_beta')   # just rotations??
vrut.tracker()
tracker.command(6)  # RESET1    ... see multi.py in AndyL
tracker.reset()
```

```
#
****************************************************************
**********************
# Loading objects and their initial positions

station = vrut.addchild('n:\Jessica\Thesis SIM\Models\Station2.wrl')
astronaut = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Astronaut.wrl')
astronaut.curtain(vrut.CLOSE)
arrow = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Cross.wrl',vrut.HEAD)
arrow.translate(0,0,.75)
arrow.curtain(vrut.CLOSE)

firstQuestion = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Firstquestion.wrl',vrut.HEAD)
firstQuestion.translate(0,0,1.5)
firstQuestion.scale(scfac,scfac,scfac)
firstQuestion.curtain(vrut.CLOSE)
secondQuestion = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Secondquestion.wrl',vrut.HEAD)
secondQuestion.translate(0,0,1.5)
secondQuestion.scale(scfac,scfac,scfac)
secondQuestion.curtain(vrut.CLOSE)
line = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Underline.wrl',vrut.HEAD)
line.translate(0,lay,lz)
line.scale(scfac,scfac,scfac)
line.curtain(vrut.CLOSE)

ansblue = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansbluehatch.wrl',vrut.HEAD)
ansyellw = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyellowhatch.wrl',vrut.HEAD)
ansblue.translate(qax,qby,qaz)
ansblue.scale(ansscfac,ansscfac,ansscfac)
ansyellw.translate(qax,qby,qaz)
ansyellw.scale(ansscfac,ansscfac,ansscfac)
ansblue.curtain(vrut.CLOSE)
ansyellw.curtain(vrut.CLOSE)

ansupright = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansupright.wrl',vrut.HEAD)
ansrdown = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansrightdown.wrl',vrut.HEAD)
ansupdown = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansupsidedown.wrl',vrut.HEAD)
ansldown = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansleftdown.wrl',vrut.HEAD)
ansnoknow = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansupright.translate(qax,qay,qaz)
ansupright.scale(ansscfac,ansscfac,ansscfac)
ansrdown.translate(qax,qay,qaz)
ansrdown.scale(ansscfac,ansscfac,ansscfac)
ansupdown.translate(qax,qay,qaz)
ansupdown.scale(ansscfac,ansscfac,ansscfac)
ansldown.translate(qax,qay,qaz)
ansldown.scale(ansscfac,ansscfac,ansscfac)
ansnoknow.translate(qax,qay,qaz)
ansnoknow.scale(ansscfac,ansscfac,ansscfac)
ansupright.curtain(vrut.CLOSE)
ansrdown.curtain(vrut.CLOSE)
ansupdown.curtain(vrut.CLOSE)
ansldown.curtain(vrut.CLOSE)
ansnoknow.curtain(vrut.CLOSE)
ans3pit90f = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Anspitch90f.wrl',vrut.HEAD)
ans3pit90b = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Anspitch90b.wrl',vrut.HEAD)
ans3yaw180 = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyaw180.wrl',vrut.HEAD)
ans3yaw90l = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyaw90l.wrl',vrut.HEAD)
ans3yaw90r = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyaw90r.wrl',vrut.HEAD)
ans3norot = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansnorotate.wrl',vrut.HEAD)
ans3noknow = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans3pit90f.translate(qax,qcy,qaz)
ans3pit90f.scale(ansscfac,ansscfac,ansscfac)
ans3pit90b.translate(qax,qcy,qaz)
ans3pit90b.scale(ansscfac,ansscfac,ansscfac)
ans3yaw180.translate(qax,qcy,qaz)
ans3yaw180.scale(ansscfac,ansscfac,ansscfac)
ans3yaw90l.translate(qax,qcy,qaz)
ans3yaw90l.scale(ansscfac,ansscfac,ansscfac)
ans3yaw90r.translate(qax,qcy,qaz)
ans3yaw90r.scale(ansscfac,ansscfac,ansscfac)
ans3norot.translate(qax,qcy,qaz)
ans3norot.scale(ansscfac,ansscfac,ansscfac)
ans3noknow.translate(qax,qcy,qaz)
ans3noknow.scale(ansscfac,ansscfac,ansscfac)
ans3pit90f.curtain(vrut.CLOSE)
ans3pit90b.curtain(vrut.CLOSE)
ans3yaw180.curtain(vrut.CLOSE)
ans3yaw90l.curtain(vrut.CLOSE)
ans3yaw90r.curtain(vrut.CLOSE)
ans3norot.curtain(vrut.CLOSE)
ans3noknow.curtain(vrut.CLOSE)

ansr180 = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll180.wrl',vrut.HEAD)
ansr90l = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll90l.wrl',vrut.HEAD)
ansr90r = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll90r.wrl',vrut.HEAD)
ansr0 = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll0.wrl',vrut.HEAD)
ans4noknow = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansr0.translate(qax,qdy,qaz)
ansr0.scale(ansscfac,ansscfac,ansscfac)
ansr90l.translate(qax,qdy,qaz)
ansr90l.scale(ansscfac,ansscfac,ansscfac)
ansr90r.translate(qax,qdy,qaz)
ansr90r.scale(ansscfac,ansscfac,ansscfac)
ansr180.translate(qax,qdy,qaz)
ansr180.scale(ansscfac,ansscfac,ansscfac)
ans4noknow.translate(qax,qdy,qaz)
ans4noknow.scale(ansscfac,ansscfac,ansscfac)
ansr180.curtain(vrut.CLOSE)
ansr90l.curtain(vrut.CLOSE)
ansr90r.curtain(vrut.CLOSE)
ansr0.curtain(vrut.CLOSE)
ans4noknow.curtain(vrut.CLOSE)

anshealth = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
anshab = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ansexp = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
```

```
anseva = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
anscont = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ansclose = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ansstore = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
anscentr = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans5noknow = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansexp.translate(qax,qay,qaz)
ansexp.scale(ansscfac,ansscfac,ansscfac)
anseva.translate(qax,qay,qaz)
anseva.scale(ansscfac,ansscfac,ansscfac)
ansstore.translate(qax,qay,qaz)
ansstore.scale(ansscfac,ansscfac,ansscfac)
anshealth.translate(qax,qay,qaz)
anshealth.scale(ansscfac,ansscfac,ansscfac)
anshab.translate(qax,qay,qaz)
anshab.scale(ansscfac,ansscfac,ansscfac)
anscentr.translate(qax,qay,qaz)
anscentr.scale(ansscfac,ansscfac,ansscfac)
anscont.translate(qax,qay,qaz)
anscont.scale(ansscfac,ansscfac,ansscfac)
ansclose.translate(qax,qay,qaz)
ansclose.scale(ansscfac,ansscfac,ansscfac)
ans5noknow.translate(qax,qay,qaz)
ans5noknow.scale(ansscfac,ansscfac,ansscfac)
anshealth.curtain(vrut.CLOSE)
anshab.curtain(vrut.CLOSE)
ansexp.curtain(vrut.CLOSE)
anseva.curtain(vrut.CLOSE)
anscont.curtain(vrut.CLOSE)
ansclose.curtain(vrut.CLOSE)
ansstore.curtain(vrut.CLOSE)
anscentr.curtain(vrut.CLOSE)
ans5noknow.curtain(vrut.CLOSE)

ans2health = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
ans2hab = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans2exp = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
ans2eva = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
ans2cont = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans2close = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans2store = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
ans2centr = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans6noknow = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans2exp.translate(qax,qby,qaz)
ans2exp.scale(ansscfac,ansscfac,ansscfac)
ans2eva.translate(qax,qby,qaz)
ans2eva.scale(ansscfac,ansscfac,ansscfac)
ans2store.translate(qax,qby,qaz)
ans2store.scale(ansscfac,ansscfac,ansscfac)
ans2health.translate(qax,qby,qaz)
ans2health.scale(ansscfac,ansscfac,ansscfac)
ans2hab.translate(qax,qby,qaz)
ans2hab.scale(ansscfac,ansscfac,ansscfac)
ans2centr.translate(qax,qby,qaz)
ans2centr.scale(ansscfac,ansscfac,ansscfac)
ans2cont.translate(qax,qby,qaz)
ans2cont.scale(ansscfac,ansscfac,ansscfac)
ans2close.translate(qax,qby,qaz)
ans2close.scale(ansscfac,ansscfac,ansscfac)
ans6noknow.translate(qax,qby,qaz)
ans6noknow.scale(ansscfac,ansscfac,ansscfac)
ans2health.curtain(vrut.CLOSE)
ans2hab.curtain(vrut.CLOSE)
ans2exp.curtain(vrut.CLOSE)
ans2eva.curtain(vrut.CLOSE)
ans2cont.curtain(vrut.CLOSE)
ans2close.curtain(vrut.CLOSE)
ans2store.curtain(vrut.CLOSE)
ans2centr.curtain(vrut.CLOSE)
ans6noknow.curtain(vrut.CLOSE)

ans3health = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
ans3hab = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans3exp = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
ans3eva = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
ans3cont = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans3close = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans3store = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
ans3centr = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans7noknow = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans3exp.translate(qax,qcy,qaz)
ans3exp.scale(ansscfac,ansscfac,ansscfac)
ans3eva.translate(qax,qcy,qaz)
ans3eva.scale(ansscfac,ansscfac,ansscfac)
ans3store.translate(qax,qcy,qaz)
ans3store.scale(ansscfac,ansscfac,ansscfac)
ans3health.translate(qax,qcy,qaz)
ans3health.scale(ansscfac,ansscfac,ansscfac)
ans3hab.translate(qax,qcy,qaz)
ans3hab.scale(ansscfac,ansscfac,ansscfac)
ans3centr.translate(qax,qcy,qaz)
ans3centr.scale(ansscfac,ansscfac,ansscfac)
ans3cont.translate(qax,qcy,qaz)
ans3cont.scale(ansscfac,ansscfac,ansscfac)
ans3close.translate(qax,qcy,qaz)
ans3close.scale(ansscfac,ansscfac,ansscfac)
ans7noknow.translate(qax,qcy,qaz)
ans7noknow.scale(ansscfac,ansscfac,ansscfac)
ans3health.curtain(vrut.CLOSE)
ans3hab.curtain(vrut.CLOSE)
ans3exp.curtain(vrut.CLOSE)
ans3eva.curtain(vrut.CLOSE)
ans3cont.curtain(vrut.CLOSE)
ans3close.curtain(vrut.CLOSE)
ans3store.curtain(vrut.CLOSE)
ans3centr.curtain(vrut.CLOSE)
ans7noknow.curtain(vrut.CLOSE)

ans4health = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
ans4hab = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans4exp = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
```

```
ans4eva = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
ans4cont = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans4close = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans4store = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
ans4centr = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans8noknow = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans4exp.translate(qax,qdy,qaz)
ans4exp.scale(ansscfac,ansscfac,ansscfac)
ans4eva.translate(qax,qdy,qaz)
ans4eva.scale(ansscfac,ansscfac,ansscfac)
ans4store.translate(qax,qdy,qaz)
ans4store.scale(ansscfac,ansscfac,ansscfac)
ans4health.translate(qax,qdy,qaz)
ans4health.scale(ansscfac,ansscfac,ansscfac)
ans4hab.translate(qax,qdy,qaz)
ans4hab.scale(ansscfac,ansscfac,ansscfac)
ans4centr.translate(qax,qdy,qaz)
ans4centr.scale(ansscfac,ansscfac,ansscfac)
ans4cont.translate(qax,qdy,qaz)
ans4cont.scale(ansscfac,ansscfac,ansscfac)
ans4close.translate(qax,qdy,qaz)
ans4close.scale(ansscfac,ansscfac,ansscfac)
ans8noknow.translate(qax,qdy,qaz)
ans8noknow.scale(ansscfac,ansscfac,ansscfac)
ans4health.curtain(vrut.CLOSE)
ans4hab.curtain(vrut.CLOSE)
ans4exp.curtain(vrut.CLOSE)
ans4eva.curtain(vrut.CLOSE)
ans4cont.curtain(vrut.CLOSE)
ans4close.curtain(vrut.CLOSE)
ans4store.curtain(vrut.CLOSE)
ans4centr.curtain(vrut.CLOSE)
ans8noknow.curtain(vrut.CLOSE)

columbus = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Columbus.wrl',vrut.HEAD)
stowage = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Stowage.wrl',vrut.HEAD)
zarya = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Zarya.wrl',vrut.HEAD)
destiny = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Destiny.wrl',vrut.HEAD)
habitation = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Habitation.wrl',vrut.HEAD)
kibo = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Kibo.wrl',vrut.HEAD)
reminder = vrut.addchild('n:\Jessica\Thesis
SIM\Models\Reminder.wrl', vrut.HEAD)
columbus.scale(0.25,.25,.25)
stowage.scale(0.25,.25,.25)
zarya.scale(0.25,.25,.25)
destiny.scale(0.25,.25,.25)
habitation.scale(0.25,.25,.25)
kibo.scale(0.25,.25,.25)
columbus.translate(0,-.1,.5)
stowage.translate(0,-.1,.5)
zarya.translate(0,-.1,.5)
destiny.translate(0,-.1,.5)
habitation.translate(0,-.1,.5)
kibo.translate(0,-.1,.5)
reminder.translate(0,-.4,2)
columbus.curtain(vrut.CLOSE)
stowage.curtain(vrut.CLOSE)
zarya.curtain(vrut.CLOSE)
destiny.curtain(vrut.CLOSE)
habitation.curtain(vrut.CLOSE)
kibo.curtain(vrut.CLOSE)
reminder.curtain(vrut.CLOSE)


#
*****************************************************************
*********************


# geographic location of each denoted point relative to center of
station
# geoxyz[endpoint] = x,y,z       endpoint can be module or at
nodes
# interesting note: you need to make this floating numbers!
# 6/4/2001 this is changed to just the nodes!
#geoxyz = [[-5.0,0.0,-5.25],[-5.0,0.0,5.25],[0.0,-5.0,-5.25],
#        [0.0,5.0,5.25],[5.0,0.0,-5.25],[5.0,0.0,5.25],
#        [0.0,0.0,0.0],[0.0,0.0,5.25],[0.0,0.0,-5.25]]

geoxyz = [[0.0,0.0,-5.25],[0.0,0.0,5.25],[0.0,0.0,-5.25],
        [0.0,0.0,5.25],[0.0,0.0,-5.25],[0.0,0.0,5.25],
        [0.0,0.0,0.0],[0.0,0.0,5.25],[0.0,0.0,-5.25]]

trials = [1,0,3,2,5,4]
initpost = [0,0,0,0,1,1]
# initpost[module] = 0 if in canonical, 1 if right shldr down; posture
you learn "route"

# segments[module] = endpoints passing necessary to get there from
center of station
#segments = [[8,0],[7,1],[8,2],[7,3],[8,4],[7,5]]
# segments is unnecessary because all you need is to do
geoxyz[module][2] !!!!

# segori[module] = orientation (relative to reset body_ori)
# orientation given as (nose,feet)
segori = [[2,3],[2,3],[3,4],[1,4],[0,5],[0,4]]

# other[count] = [other module, other module] within that hatch
other = [[3,1],[0,2],[5,1],[0,4],[5,3],[2,4]]

global    nextflag,module,count,routeflag,    showing,
set1flag,moremodflag
global    q1flag,q2flag,q3flag,q4flag,q5flag,q6flag,q7flag,q8flag,
begintime
global
q1index,q2index,q3index,q4index,q5index,q6index,q7index,q8index
nextflag = off
module = 0
count = 0
routeflag = on
set1flag = on
showing = off
moremodflag = on
q1flag = off
q2flag = off
q3flag = off
q4flag = off
q5flag = off
q6flag = off
q7flag = off
q8flag = off
q1index = 0
q2index = 0
q3index = 0
q4index = 0
q5index = 0
```

```
q6index = 0
q7index = 0
q8index = 0

# Timernums
layout = 1
closeTarget = 2
closeAstr = 3
placeAstrnt = 4
waitPOP = 5
waitFLYIN = 6
waitENTER = 7
showFirstQuest = 8
showSecondQuest = 9
scroller = 10
move2node = 11
move = 12
rotateSetup = 13
rotating = 14
taskpoint = 15
next = 16
waitNEXT = 17

#
*****************************************************************
**********************

def initVariables():
   global module,count, routeflag, showing, q1flag, set1flag, nextflag
   global
q1index,q2index,q3index,q4index,q5index,q6index,q7index,q8index
   q1index = 0
   q2index = 0
   q3index = 0
   q4index = 0
   q5index = 0
   q6index = 0
   q7index = 0
   q8index = 0
   module = trials[count]
   # increase count??? where is it?
   if nextflag == on:
      nextflag = off
      routeflag = on
   showing = off
   q1flag = on
   line.translate(0,lay,lz)
   set1flag = on
   initpos = initpost[module]
   vrut.reset(vrut.BODY_ORI)
   vrut.reset(vrut.HEAD_POS)
   if initpos == 1:
      vrut.rotate(vrut.BODY_ORI, 0,0,90)

def showTarget(modulenum, tclose):
   if modulenum == 0:
      columbus.curtain(vrut.OPEN)
   elif modulenum == 1:
      stowage.curtain(vrut.OPEN)
   elif modulenum == 2:
      zarya.curtain(vrut.OPEN)
   elif modulenum == 3:
      destiny.curtain(vrut.OPEN)
   elif modulenum == 4:
      habitation.curtain(vrut.OPEN)
   elif modulenum == 5:
      kibo.curtain(vrut.OPEN)
   vrut.starttimer(closeTarget,tclose)

def closeAll():
   columbus.curtain(vrut.CLOSE)
   stowage.curtain(vrut.CLOSE)
   zarya.curtain(vrut.CLOSE)
   destiny.curtain(vrut.CLOSE)
   habitation.curtain(vrut.CLOSE)
   kibo.curtain(vrut.CLOSE)

def defAngles(nose,feet):
   # print(nose,feet)
   # these are defined as if starting from original orientation
   # 4/25/2001 editted nose 3 or 1
   print('nose,feet',nose,feet)
   if nose == 2 or nose == 0:
      pitch = 0
      if nose == 2: yaw = -90
      else: yaw = 90
      if feet == 4:
         if nose == 2:
            roll = -90
         else: roll = 90
      elif feet == 1:
         roll = 180
      elif feet == 5:
         if nose == 2:
            roll = 90
         else: roll = -90
      elif feet == 3:
         roll = 0

   if nose == 3 or nose == 1:
      if feet == 5:
         if nose == 3:
            pitch = -90
            yaw = 0
            roll = 0
         else:
            yaw = 180
            pitch = 90 # ?
            roll = 0
      elif feet == 0:
         if nose == 3:
            yaw = 90
            pitch = -90
            roll = 0
         else:
            yaw = -90
            pitch = 90
            roll = 0
      elif feet == 4:
         if nose == 3:
            yaw = 180
            pitch = -90
            roll = 0
         else:
            yaw = 0
            pitch = 90
            roll = 0
      elif feet == 2:
         if nose == 3:
            yaw = -90
            pitch = -90
            roll = 0
         else:
            yaw = 90
            pitch = 90
            roll = 0

   if nose == 4 or nose == 5:
    pitch = 0
    if nose == 4: yaw = 0
    else: yaw = 180
    if feet == 3:
      roll = 0
    elif feet == 1:
      roll = 180
    elif feet == 0:
      if nose == 4:
        roll = 90
      else: roll = -90
    elif feet == 2:
      if nose == 4:
        roll = -90
      else: roll = 90
  return [yaw,pitch,roll]

def Eul2Quats(psi,theta,phi):
  # psi == yaw; theta == pitch; phi == roll
  # Mode of implementation : model.avatar.rotate(-q2,q3,-q1,2*math.acos(q0)*RAD2DEG)
  cr = math.cos(phi*DEG2RAD/2)
  cp = math.cos(theta*DEG2RAD/2)
  cy = math.cos(psi*DEG2RAD/2)
  sr = math.sin(phi*DEG2RAD/2)
  sp = math.sin(theta*DEG2RAD/2)
  sy = math.sin(psi*DEG2RAD/2)
  cpcy = cp*cy
  spsy = sp*sy
  q0 = cr*cpcy + sr*spsy
  q1 = sr*cpcy - cr*spsy
  q2 = cr*sp*cy + sr*cp*sy
  q3 = cr*cp*sy - sr*sp*cy
  quat0 = -q2
  quat1 = q3
  quat2 = -q1
  quat3 = 2*math.acos(q0)*RAD2DEG
  return [quat0,quat1,quat2,quat3]


def defLearnTurns(modulenum):
  if modulenum == 0:
    yaw = 90
    pitch = 0
    roll = 0
  elif modulenum == 1:
    yaw = -90
    pitch = 0
    roll = 0
  elif modulenum == 2:
    yaw = 0
    pitch = 90
    roll = 0
  elif modulenum == 3:
    yaw = 0
    pitch = 90
    roll = 0
  elif modulenum == 4:
    yaw = -90
    pitch = 0
    roll = 0
  elif modulenum == 5:
    yaw = 90
    pitch = 0
    roll = 0

  return [yaw,pitch,roll]

def closeOthers(flag):
  #global q1flag,q2flag,q3flag,q4flag,q5flag,q6flag,q7flag,q8flag
  if flag == 'q2flag':
    ansblue.curtain(vrut.CLOSE)
    ansyellw.curtain(vrut.CLOSE)
  elif flag == 'q1flag':
    ansupright.curtain(vrut.CLOSE)
    ansrdown.curtain(vrut.CLOSE)
    ansupdown.curtain(vrut.CLOSE)
    ansldown.curtain(vrut.CLOSE)
    ansnoknow.curtain(vrut.CLOSE)
  elif flag == 'q3flag':
    ans3pit90f.curtain(vrut.CLOSE)
    ans3pit90b.curtain(vrut.CLOSE)
    ans3yaw180.curtain(vrut.CLOSE)
    ans3yaw90l.curtain(vrut.CLOSE)
    ans3yaw90r.curtain(vrut.CLOSE)
    ans3norot.curtain(vrut.CLOSE)
    ans3noknow.curtain(vrut.CLOSE)
  elif flag == 'q4flag':
    ansr180.curtain(vrut.CLOSE)
    ansr90l.curtain(vrut.CLOSE)
    ansr90r.curtain(vrut.CLOSE)
    ansr0.curtain(vrut.CLOSE)
    ans4noknow.curtain(vrut.CLOSE)
  elif flag == 'q5flag':
    anshealth.curtain(vrut.CLOSE)
    anshab.curtain(vrut.CLOSE)
    ansexp.curtain(vrut.CLOSE)
    anseva.curtain(vrut.CLOSE)
    anscont.curtain(vrut.CLOSE)
    ansclose.curtain(vrut.CLOSE)
    ansstore.curtain(vrut.CLOSE)
    anscentr.curtain(vrut.CLOSE)
    ans5noknow.curtain(vrut.CLOSE)
  elif flag == 'q6flag':
    ans2health.curtain(vrut.CLOSE)
    ans2hab.curtain(vrut.CLOSE)
    ans2exp.curtain(vrut.CLOSE)
    ans2eva.curtain(vrut.CLOSE)
    ans2cont.curtain(vrut.CLOSE)
    ans2close.curtain(vrut.CLOSE)
    ans2store.curtain(vrut.CLOSE)
    ans2centr.curtain(vrut.CLOSE)
    ans6noknow.curtain(vrut.CLOSE)
  elif flag == 'q7flag':
    ans3health.curtain(vrut.CLOSE)
    ans3hab.curtain(vrut.CLOSE)
    ans3exp.curtain(vrut.CLOSE)
    ans3eva.curtain(vrut.CLOSE)
    ans3cont.curtain(vrut.CLOSE)
    ans3close.curtain(vrut.CLOSE)
    ans3store.curtain(vrut.CLOSE)
    ans3centr.curtain(vrut.CLOSE)
    ans7noknow.curtain(vrut.CLOSE)
  elif flag == 'q8flag':
    ans4health.curtain(vrut.CLOSE)
    ans4hab.curtain(vrut.CLOSE)
    ans4exp.curtain(vrut.CLOSE)
    ans4eva.curtain(vrut.CLOSE)
    ans4cont.curtain(vrut.CLOSE)
    ans4close.curtain(vrut.CLOSE)
    ans4store.curtain(vrut.CLOSE)
    ans4centr.curtain(vrut.CLOSE)
    ans8noknow.curtain(vrut.CLOSE)

def scrollAns(flag,which):
  if flag == 'q2flag':
    print(which)
    closeOthers('q2flag')
    if which == 0:
      ansblue.curtain(vrut.OPEN)
    elif which == 1:
      ansyellw.curtain(vrut.OPEN)
    else:
      print('error in q2flag')
  elif flag == 'q1flag':
    closeOthers('q1flag')
    if which == 0:
      ansupright.curtain(vrut.OPEN)
    elif which == 1:
      ansrdown.curtain(vrut.OPEN)
    elif which == 2:
      ansupdown.curtain(vrut.OPEN)
    elif which == 3:
      ansldown.curtain(vrut.OPEN)
    elif which == 4:
      ansnoknow.curtain(vrut.OPEN)
    else:
      print('error in q1flag')
  elif flag == 'q3flag':
    closeOthers('q3flag')
    if which == 0:
      ans3pit90f.curtain(vrut.OPEN)
    elif which == 1:
      ans3pit90b.curtain(vrut.OPEN)
    elif which == 2:
      ans3yaw180.curtain(vrut.OPEN)
    elif which == 3:
      ans3yaw90l.curtain(vrut.OPEN)
    elif which == 4:
      ans3yaw90r.curtain(vrut.OPEN)
    elif which == 5:
      ans3norot.curtain(vrut.OPEN)
    elif which == 6:
      ans3noknow.curtain(vrut.OPEN)
    else:
      print('error in q3flag')
  elif flag == 'q4flag':
    closeOthers('q4flag')
    if which == 0:
      ansr0.curtain(vrut.OPEN)
    elif which == 1:
      ansr90l.curtain(vrut.OPEN)
    elif which == 2:
      ansr90r.curtain(vrut.OPEN)
    elif which == 3:
      ansr180.curtain(vrut.OPEN)
    elif which == 4:
      ans4noknow.curtain(vrut.OPEN)
    else:
      print('error in q4flag')
  elif flag == 'q5flag':
    closeOthers('q5flag')
    if which == 0:
      ansexp.curtain(vrut.OPEN)
    elif which == 1:
      anseva.curtain(vrut.OPEN)
    elif which == 2:
      ansstore.curtain(vrut.OPEN)
    elif which == 3:
      anshealth.curtain(vrut.OPEN)
    elif which == 4:
      anshab.curtain(vrut.OPEN)
    elif which == 5:
      anscentr.curtain(vrut.OPEN)
    elif which == 6:
      anscont.curtain(vrut.OPEN)
    elif which == 7:
      ansclose.curtain(vrut.OPEN)
    elif which == 8:
      ans5noknow.curtain(vrut.OPEN)
    else:
      print('error in q5flag')
  elif flag == 'q6flag':
    closeOthers('q6flag')
    print('which', which)
    if which == 0:
      ans2exp.curtain(vrut.OPEN)
    elif which == 1:
      ans2eva.curtain(vrut.OPEN)
    elif which == 2:
      ans2store.curtain(vrut.OPEN)
    elif which == 3:
      ans2health.curtain(vrut.OPEN)
    elif which == 4:
      ans2hab.curtain(vrut.OPEN)
    elif which == 5:
      ans2centr.curtain(vrut.OPEN)
    elif which == 6:
      ans2cont.curtain(vrut.OPEN)
    elif which == 7:
      ans2close.curtain(vrut.OPEN)
    elif which == 8:
      ans6noknow.curtain(vrut.OPEN)
  elif flag == 'q7flag':
    closeOthers('q7flag')
    if which == 0:
      ans3exp.curtain(vrut.OPEN)
    elif which == 1:
      ans3eva.curtain(vrut.OPEN)
    elif which == 2:
      ans3store.curtain(vrut.OPEN)
    elif which == 3:
      ans3health.curtain(vrut.OPEN)
    elif which == 4:
      ans3hab.curtain(vrut.OPEN)
    elif which == 5:
      ans3centr.curtain(vrut.OPEN)
    elif which == 6:
      ans3cont.curtain(vrut.OPEN)
    elif which == 7:
      ans3close.curtain(vrut.OPEN)
    elif which == 8:
      ans7noknow.curtain(vrut.OPEN)
    else:
      print('error in q7flag')
  elif flag == 'q8flag':
    closeOthers('q8flag')
    if which == 0:
      ans4exp.curtain(vrut.OPEN)
    elif which == 1:
      ans4eva.curtain(vrut.OPEN)
    elif which == 2:
      ans4store.curtain(vrut.OPEN)
    elif which == 3:
      ans4health.curtain(vrut.OPEN)
    elif which == 4:
      ans4hab.curtain(vrut.OPEN)
    elif which == 5:
      ans4centr.curtain(vrut.OPEN)
    elif which == 6:
      ans4cont.curtain(vrut.OPEN)
    elif which == 7:
      ans4close.curtain(vrut.OPEN)
    elif which == 8:
      ans8noknow.curtain(vrut.OPEN)
    else:
        print('error in q8flag')


def closeEverything():
    global set1flag
    firstQuestion.curtain(vrut.CLOSE)
    secondQuestion.curtain(vrut.CLOSE)
    line.curtain(vrut.CLOSE)
    if set1flag == on:
        ansblue.curtain(vrut.CLOSE)
        ansyellw.curtain(vrut.CLOSE)
        ansupright.curtain(vrut.CLOSE)
        ansrdown.curtain(vrut.CLOSE)
        ansupdown.curtain(vrut.CLOSE)
        ansldown.curtain(vrut.CLOSE)
        ansnoknow.curtain(vrut.CLOSE)
        ans3pit90f.curtain(vrut.CLOSE)
        ans3pit90b.curtain(vrut.CLOSE)
        ans3yaw180.curtain(vrut.CLOSE)
        ans3yaw90l.curtain(vrut.CLOSE)
        ans3yaw90r.curtain(vrut.CLOSE)
        ans3norot.curtain(vrut.CLOSE)
        ans3noknow.curtain(vrut.CLOSE)
        ansr180.curtain(vrut.CLOSE)
        ansr90l.curtain(vrut.CLOSE)
        ansr90r.curtain(vrut.CLOSE)
        ansr0.curtain(vrut.CLOSE)
        ans4noknow.curtain(vrut.CLOSE)
    elif set1flag == off:
        anshealth.curtain(vrut.CLOSE)
        anshab.curtain(vrut.CLOSE)
        ansexp.curtain(vrut.CLOSE)
        anseva.curtain(vrut.CLOSE)
        anscont.curtain(vrut.CLOSE)
        ansclose.curtain(vrut.CLOSE)
        ansstore.curtain(vrut.CLOSE)
        anscentr.curtain(vrut.CLOSE)
        ans5noknow.curtain(vrut.CLOSE)
        ans2health.curtain(vrut.CLOSE)
        ans2hab.curtain(vrut.CLOSE)
        ans2exp.curtain(vrut.CLOSE)
        ans2eva.curtain(vrut.CLOSE)
        ans2cont.curtain(vrut.CLOSE)
        ans2close.curtain(vrut.CLOSE)
        ans2store.curtain(vrut.CLOSE)
        ans2centr.curtain(vrut.CLOSE)
        ans6noknow.curtain(vrut.CLOSE)
        ans3health.curtain(vrut.CLOSE)
        ans3hab.curtain(vrut.CLOSE)
        ans3exp.curtain(vrut.CLOSE)
        ans3eva.curtain(vrut.CLOSE)
        ans3cont.curtain(vrut.CLOSE)
        ans3close.curtain(vrut.CLOSE)
        ans3store.curtain(vrut.CLOSE)
        ans3centr.curtain(vrut.CLOSE)
        ans7noknow.curtain(vrut.CLOSE)
        ans4health.curtain(vrut.CLOSE)
        ans4hab.curtain(vrut.CLOSE)
        ans4exp.curtain(vrut.CLOSE)
        ans4eva.curtain(vrut.CLOSE)
        ans4cont.curtain(vrut.CLOSE)
        ans4close.curtain(vrut.CLOSE)
        ans4store.curtain(vrut.CLOSE)
        ans4centr.curtain(vrut.CLOSE)
        ans8noknow.curtain(vrut.CLOSE)


def detWhat():
    global routeflag,module,count,moremodflag, nextflag
    print('in detWhat')
    if routeflag == off:
        return module
    else:
        routeflag = off
        if moremodflag == on:
            moremodflag = off
            return other[count][0]
        else:
            moremodflag = on
            nextflag = on
            return other[count][1]


def myKeyboard(key):
    global begintime
    if key == ' ':
        reminder.curtain(vrut.CLOSE)
        vrut.starttimer(layout,0.001)
        begintime = time.time()
        out = str(time.time()-begintime) + '\t' + str(count) +
'\tcount\n'
        keys.write(out)
        keys.flush()
    elif key == 'r':
        tracker.command(6)  # RESET1   ... see multi.py in AndyL
        tracker.reset()
    elif key == '1':
        showTarget(module,.75)

def mytimer(timernum):
    global module, count, routeflag, showing, set1flag,moremodflag,
nextflag
    global maxx,maxy,maxz, x,y,z, stepx,stepy,stepz,
startpoint,begintime
    global yaw,pitch,roll, turny,turnp,turnr
    global q1flag,q1index,q2flag,q2index,q3flag,q3index,q4flag,q4index
    global q5flag,q5index,q6flag,q6index,q7flag,q7index,q8flag,q8index
    if timernum == layout:
        # initialize variables
        initVariables()
        # show target; close
        # show astronaut or pop up SIM
        if routeflag == on:
            showTarget(module,2)
            out = str(initpost[module])+'\t'+str(module)+'\t'+str(-
1)+'\t'+str(-1)+'\t'+str(-1)+'\t'+str(-1)+'\t'+str(-1)+'\t'
            file.write(out)
            file.flush()
            vrut.starttimer(placeAstrnt,2.02)
        else:
            print( count, 'this is count')
            if count == 0 or count == 1 or count == 2:
                vrut.starttimer(next,0.001)
            elif count == 3 or count == 4 or count == 5:
                startpoint = time.time()
                vrut.starttimer(taskpoint,2.02)
            else: print('there is an error in layout')
        # first questions
        # travel to the astronaut
        # second question
        # play sound/back to Control
        # show targe; close
        # pointing task
        # first question
        # second question
        # determine what are the two other targets
        # show target of other target 1; close
        # pointing task for other target 1
        # first question
        # second question
        # show target of other target 2; close
        # point
        # first question
        # second question

    elif timernum == closeTarget:
        closeAll()

    elif timernum == placeAstrnt:
        astronaut.curtain(vrut.OPEN)

astronaut.translate(geoxyz[module][0],geoxyz[module][1],geoxyz[mod
ule][2])
        angles = defAngles(segori[module][0],segori[module][1])
        print('angles',angles)
        quat = Eul2Quats(angles[0],angles[1],angles[2])
        astronaut.rotate(quat[0],quat[1],quat[2],quat[3])
        startpoint = time.time()
        if count == 0 or count == 1 or count == 2:
            vrut.starttimer(showFirstQuest,2)
        elif count == 3 or count == 4 or count == 5:
            vrut.starttimer(waitFLYIN, 0.001)

    elif timernum == waitPOP:
        if sid.buttons() == sid.BUTTON10:
            if set1flag == on:
                vrut.starttimer(showFirstQuest,0.001)
            else:
                vrut.starttimer(showSecondQuest,0.001)
        else:
            vrut.starttimer(waitPOP,rate)

    elif timernum == showFirstQuest:
        if showing == on:
            showing = off
            closeEverything()
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\thide_quest_set1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(waitPOP,rate)
        elif showing == off:
            showing = on
            firstQuestion.curtain(vrut.OPEN)
            line.curtain(vrut.OPEN)
            scrollAns('q1flag',q1index)
            scrollAns('q2flag',q2index)
            scrollAns('q3flag',q3index)
            scrollAns('q4flag',q4index)
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\tshow_quest_set1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)

    elif timernum == showSecondQuest:
        if showing == on:
            showing = off
            closeEverything()
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\thide_quest_set2\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(waitPOP,rate)
        elif showing == off:
            showing = on
            secondQuestion.curtain(vrut.OPEN)
            line.curtain(vrut.OPEN)
            print('2nd question',q5index,q6index,q7index,q8index)
            scrollAns('q5flag',q5index)
            scrollAns('q6flag',q6index)
            scrollAns('q7flag',q7index)
            scrollAns('q8flag',q8index)
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\tshow_quest_set1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)

    elif timernum == scroller:
        if sid.buttons() == sid.BUTTON10 and routeflag == on:
            if set1flag == on:
                vrut.starttimer(showFirstQuest,0.001)
            else:
                vrut.starttimer(showSecondQuest,0.001)
            # watch out for a possible BUG HERE!
        elif sid.buttons() == sid.BUTTON1:
            # check for which question
            # scroll up (add)
            if q2flag == on:
                if q2index == 0:
                    q2index = 1
                else:
                    q2index = 0
                scrollAns('q2flag',q2index)
            elif q1flag == on:
                if q1index == 4:
                    q1index = 0
                else:
                    q1index = q1index + 1
                scrollAns('q1flag',q1index)
            elif q3flag == on:
                if q3index == 6:
                    q3index = 0
                else:
                    q3index = q3index + 1
                scrollAns('q3flag',q3index)
            elif q4flag == on:
                if q4index == 4:
                    q4index = 0
                else:
                    q4index = q4index + 1
                scrollAns('q4flag',q4index)
            elif q5flag == on:
                if q5index == 8:
                    q5index = 0
                else:
                    q5index = q5index + 1
                scrollAns('q5flag',q5index)
            elif q6flag == on:
                if q6index == 8:
                    q6index = 0
                else:
                    q6index = q6index + 1
                scrollAns('q6flag',q6index)
            elif q7flag == on:
                if q7index == 8:
                    q7index = 0
                else:
                    q7index = q7index + 1
                scrollAns('q7flag',q7index)
            elif q8flag == on:
                if q8index == 8:
                    q8index = 0
                else:
                    q8index = q8index + 1
        scrollAns('q8flag',q8index)
      vrut.starttimer(scroller,rate)
      out = str(time.time()-begintime) + '\t' + str(sid.BUTTON1)
+ '\tscroll_up\n'
      keys.write(out)
      keys.flush()

    elif sid.buttons() == sid.BUTTON4:
      # check for which question
      # scroll down (subtract)
      if q2flag == on:
        if q2index == 1:
          q2index = 0
        else:
          q2index = 1
        scrollAns('q2flag',q2index)
      elif q1flag == on:
        if q1index == 0:
          q1index = 4
        else:
          q1index = q1index - 1
        scrollAns('q1flag',q1index)
      elif q3flag == on:
        if q3index == 0:
          q3index = 6
        else:
          q3index = q3index - 1
        scrollAns('q3flag',q3index)
      elif q4flag == on:
        if q4index == 0:
          q4index = 4
        else:
          q4index = q4index - 1
        scrollAns('q4flag',q4index)
      elif q5flag == on:
        if q5index == 0:
          q5index = 8
        else:
          q5index = q5index - 1
        scrollAns('q5flag',q5index)
      elif q6flag == on:
        if q6index == 0:
          q6index = 8
        else:
          q6index = q6index - 1
        scrollAns('q6flag',q6index)
      elif q7flag == on:
        if q7index == 0:
          q7index = 8
        else:
          q7index = q7index - 1
        scrollAns('q7flag',q7index)
      elif q8flag == on:
        if q8index == 0:
          q8index = 8
        else:
          q8index = q8index - 1
        scrollAns('q8flag',q8index)
      vrut.starttimer(scroller,rate)
      out = str(time.time()-begintime) + '\t' + str(sid.BUTTON1)
+ '\tscroll_down\n'
      keys.write(out)
      keys.flush()

    elif sid.buttons() == 768:   # pressing sid.BUTTON9 and
sid.BUTTON10 together
      elapstime = time.time() - startpoint
      startpoint = time.time()
      closeEverything()
```

```
      showing = off
      out = str(time.time()-begintime) + '\t' + str(768) +
'\tenter_hide_quest_set\n'
      keys.write(out)
      keys.flush()
      if set1flag == on:
        q1flag = off
        q2flag = off
        q3flag = off
        q4flag = off
        q5flag = on
        out                                                        =
str(q1index)+'\t'+str(q2index)+'\t'+str(q3index)+'\t'+str(q4index)+'
\t'+ str(elapstime)+'\t'
        file.write(out)
        file.flush()
        if routeflag == on:
          #routeflag = off
          vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\EndQuest.wav')
          vrut.starttimer(waitFLYIN,rate)
        else:
          print('repeating??')
          #print(set1flag, routeflag)
          set1flag = off
          line.translate(0,lay,lz)
          startpoint = time.time()
          vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\EndQuest.wav')
          vrut.starttimer(showSecondQuest,0.1)
      else:
        q5flag = off
        q6flag = off
        q7flag = off
        q8flag = off
        q1flag = on
        line.translate(0,lay,lz)
        out                                                        =
str(q5index)+'\t'+str(q6index)+'\t'+str(q7index)+'\t'+str(q8index)+'
\t'+ str(elapstime)+'\n'
        file.write(out)
        file.flush()
        if routeflag == on:
          routeflag = off
          print('here')
          vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\Reminder.wav')
          vrut.starttimer(layout,2)
        else:
          routeflag = on
          set1flag = on
          if nextflag == on:
            print('increasing count')
            if count == maxtrials:
              print('endprogram!!!')
              vrut.translate(vrut.HEAD_POS, 0,100,0)
              file.close()
              keys.close()
            else:
              count = count + 1
              vrut.starttimer(layout,0.001)
          else:
            startpoint = time.time()
            vrut.starttimer(taskpoint,1)
          #vrut.starttimer(taskpoint,0.1)

    elif sid.buttons() == sid.BUTTON9:
      # enter answer (record)
      # increase which question counter
```

```
        if q1flag == on:
            q1flag = off
            q2flag = on
            line.translate(0,lby,lz)
            scrollAns('q2flag',q2index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q2flag == on:
            q2flag = off
            q3flag = on
            line.translate(0,lcy,lz)
            scrollAns('q3flag',q3index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question2\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q3flag == on:
            q3flag = off
            q4flag = on
            line.translate(0,ldy,lz)
            scrollAns('q4flag',q4index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question3\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q4flag == on:
            q4flag = off
            q1flag = on
            line.translate(0,lay,lz)
            scrollAns('q1flag',q1index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question4\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q5flag == on:
            q5flag = off
            q6flag = on
            line.translate(0,lby,lz)
            scrollAns('q6flag',q6index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question5\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q6flag == on:
            q6flag = off
            q7flag = on
            line.translate(0,lcy,lz)
            scrollAns('q7flag',q7index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question6\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q7flag == on:
            q7flag = off
            q8flag = on
            line.translate(0,ldy,lz)
            scrollAns('q8flag',q8index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question7\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q8flag == on:
            q8flag = off
            q5flag = on
            line.translate(0,lay,lz)
            scrollAns('q5flag',q5index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question8\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)

    else:
        vrut.starttimer(scroller,rate)

  elif timernum == next:
    if count == 0 or count == 1 or count == 2:
        routeflag = on
        set1flag = on
        if count == maxtrials:
            print('endprogram!!!')
            vrut.translate(vrut.HEAD_POS,0,100,0)
            file.close()
            keys.close()
        else:
            count = count + 1
            vrut.starttimer(layout, 0.001)
    elif count == 3 or count == 4 or count == 5:
        routeflag = off
        vrut.starttimer(layout,0.001)
    else:
        print('there is a problem with the next timernum')

  elif timernum == waitFLYIN:
    if sid.buttons() == sid.BUTTON7:
        vrut.starttimer(closeAstr,2)
        vrut.starttimer(move2node,0.001)
        out = str(time.time()-begintime) + '\t' + str(sid.BUTTON7)
+ '\tflyin\n'
        keys.write(out)
        keys.flush()
    else:
        vrut.starttimer(waitFLYIN,rate)

  elif timernum == waitNEXT:
    if sid.buttons() == sid.BUTTON7:
        vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\Reminder.wav')
        vrut.starttimer(next,0.5)
        out = str(time.time()-begintime) + '\t' + str(sid.BUTTON7)
+ '\tend_of_route\n'
        keys.write(out)
        keys.flush()
    else:
        vrut.starttimer(waitNEXT,0.3)

  elif timernum == closeAstr:
    astronaut.curtain(vrut.CLOSE)

  elif timernum == move2node:
    maxx = geoxyz[module][0]
    maxy = geoxyz[module][1]
    maxz = geoxyz[module][2]
    stepx = maxx/flytime
    stepy = maxy/flytime
    stepz = maxz/flytime
    x = 0
    y = 0
    z = 0
```

```
        vrut.starttimer(move,0.001)

    elif timernum == move:
        if abs(x) < abs(maxx) or abs(y) < abs(maxy) or abs(z) <
abs(maxz):
            if abs(x) < abs(maxx):
                x = x + stepx
                vrut.translate(vrut.HEAD_POS, stepx,0,0)
            if abs(y) < abs(maxy):
                y = y + stepy
                vrut.translate(vrut.HEAD_POS, 0, stepy, 0)
            if abs(z) < abs(maxz):
                z = z + stepz
                vrut.translate(vrut.HEAD_POS, 0, 0, stepz)
            vrut.starttimer(move,0.001)
        else:
            vrut.starttimer(rotateSetup,1)

    elif timernum == rotateSetup:
        eulers = defLearnTurns(module)
        yaw = eulers[0]
        pitch = eulers[1]
        roll = eulers[2]
        turny = 0
        turnp = 0
        turnr = 0
        vrut.starttimer(rotating,0.001)

    elif timernum == rotating:
        if yaw <> 0 and turny <> yaw:
            if yaw < 0:
               if turny > yaw:
                   turny = turny - speed
                   vrut.rotate(vrut.BODY_ORI,-speed, 0, 0)
                   vrut.starttimer(rotating,0.001)
            elif yaw > 0:
                if turny < yaw:
                   turny = turny + speed
                   vrut.rotate(vrut.BODY_ORI,speed, 0, 0)
                   vrut.starttimer(rotating,0.001)

        elif pitch <> 0 and turnp <> pitch and turny == yaw:
            if pitch < 0:   # inverted these
                if turnp > pitch:
                   turnp = turnp - speed
                   vrut.rotate(vrut.BODY_ORI, 0, speed, 0)
                   vrut.starttimer(rotating,0.001)
            elif pitch > 0:
                if turnp < pitch:
                   turnp = turnp + speed
                   vrut.rotate(vrut.BODY_ORI, 0, -speed, 0)
                   vrut.starttimer(rotating,0.001)

        elif roll <> 0 and turnr <> roll and turny == yaw and turnp ==
pitch:
            if roll < 0:
                if turnr > roll:
                   turnr = turnr - speed
                   vrut.rotate(vrut.BODY_ORI, 0,0,-speed)
                   vrut.starttimer(rotating, 0.001)
            elif roll > 0:
                if turnr < roll:
                   turnr = turnr + speed
                   vrut.rotate(vrut.BODY_ORI, 0,0,speed)
                   vrut.starttimer(rotating, 0.001)
        elif turny == yaw and turnp == pitch and turnr == roll:
            set1flag = off
            line.translate(0,lay,lz)
            if count == 0 or count == 1 or count == 2:
                vrut.starttimer(showSecondQuest,2)
            elif count == 3 or count == 4 or count == 5:
                vrut.starttimer(waitNEXT,0.001)

    elif timernum == taskpoint:
        what = detWhat()
        print('what',what)
        showTarget(what,2)
        out = str(initpost[count])+'\t'+str(what)+'\t'
        file.write(out)
        file.flush()
        arrow.curtain(vrut.OPEN)
        #startpoint = time.time()
        vrut.starttimer(waitENTER,0.001)
    elif timernum == waitENTER:
        if sid.buttons() == sid.BUTTON9:
            arrow.curtain(vrut.CLOSE)
            data = tracker.get()
            elapstime = time.time() - startpoint
            out                                                    =
str(data[3])+'\t'+str(data[4])+'\t'+str(data[5])+'\t'+str(data[6]*RAD2
DEG)+'\t'+str(elapstime)+'\t'
            file.write(out)
            file.flush()
            startpoint = time.time()
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON9)
+ '\tenter_pointing\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(showFirstQuest,1)
        else:
            vrut.starttimer(waitENTER,rate)



#
******************************************************************
**********************

# Setting up output files
exp_time = time.localtime(time.time())
timestring = time.strftime('%b%d_%H%M',exp_time)
filename              =                'N:\Jessica\Thesis
SIM\Data\Cntrl\Phase2\Phase2a_Cntrl'+str(timestring)+'.dat'
file = open(filename,'w')
out                                                              =
'Posture\tTarget\tQuat0\tQuat1\tQuat2\tQuat3\tTime\tQ1\tQ2\
tQ3\tQ4\tTime1\tQ5\tQ6\tQ7\tQ8\tTime2\n'
file.write(out)
file.flush()
# second file out
filename2             =                'N:\Jessica\Thesis
SIM\Data\Cntrl\Phase2\Phase2a_Cntrl'+str(timestring)+'_allkeys.d
at'
keys = open(filename2,'w')

vrut.callback(vrut.KEYBOARD_EVENT,'myKeyboard')
vrut.callback(vrut.TIMER_EVENT,'mytimer')

reminder.curtain(vrut.OPEN)
```

(Part b)

```
import vrut
import time
import math
import types
import string
import sid

# 6/4/2001 This is adapting trainingCntrl_ver3 to the new protocol.
# 6/6/2001 done
# 6/23/2001 new protocol implementation. deviding phase 2 into
two parts.
# 7/2/2001 Need to edit to have second set of questions only
appear for Route, not for
#        other targets.

#
*****************************************************************
**********************
# KEY:
#
*****************************************************************
**********************

#    1                              direction nomenclature
#    |
# 2 ----- 0 into the page --  4
#    |                  out of the page -- 5
#    3
#
# Target difficulty:
#           0: modules 0 or 1
#           1: modules 4 or 5
#           2: modules 2 or 3
# FarNode:          I  MIGHT  HAVE  GOTTEN  THIS
BACKWARDS... 4/2/2001
#      0: modules 1, 3, or 5
#           1: modules 0, 2, or 4
# Posture: relative to module 6
#           0: roll, with nose at 4 & feet at 1
#           1: pitch, (positive) with nose at 1 & feet at 4
#           2: roll, with nose at 4 & feet at 2
#           3: pitch, (negative) with nose at 3 & feet at 5
#
# Module labelling:
#           0: Columbus, European Module       == Experiment
#           1: Stowage Module                  == EVA
#           2: Zarya, Russian Control Module   == Storage
#           3: Destiny, U.S. Module            == Health Fitness
#           4: Habitation Module               == Habitation
#           5: Kibo, Japanese Module           == Centrifuge
#           6: Zvezda, Russian Service Module  == Control
#           Nodes
#           7: FarNode 1, hatch at end of zvezda (blue)          #
6/4/2001 had these reversed
#           8: FarNode 0, hatch at other end of zvezda (yellow)
#
#                                                       Constants
*****************************************************************
************

on = 1
off = 0
flytime = 100
speed = 1
maxtrials = 5   # double check!
rate = 0.3

qax = .3
qay = .435
qby = .15
qcy = -.15
qdy = -.435
qaz = 2.05
lay = .275
lby = 0
lcy =-.275
ldy = -.57
lz  = 2.0
ansscfac = 4
scfac = 2.25

HMD = on

RAD2DEG = 57.295779513082323
DEG2RAD = 1/57.295779513082323

#
*****************************************************************
**********************

if HMD == on:
   vrut.go(vrut.HMD | vrut.STEREO)
else:
   vrut.go(vrut.CONSOLE)
   vrut.translate(vrut.HEAD_POS, 0,-1.82,0)

#
*****************************************************************
**********************

# Sensor
tracker = vrut.addsensor('isensemulti_beta')   # just rotations??
vrut.tracker()
tracker.command(6)  # RESET1    ... see multi.py in AndyL
tracker.reset()

#
*****************************************************************
**********************
# Loading objects and their initial positions

station = vrut.addchild('n:\Jessica\Thesis SIM\Models\Station2.wrl')
astronaut            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Astronaut.wrl')
astronaut.curtain(vrut.CLOSE)
arrow                =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Cross.wrl',vrut.HEAD)
arrow.translate(0,0,.75)
arrow.curtain(vrut.CLOSE)

firstQuestion        =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Firstquestion.wrl',vrut.HEAD)
firstQuestion.translate(0,0,1.5)
firstQuestion.scale(scfac,scfac,scfac)
firstQuestion.curtain(vrut.CLOSE)
secondQuestion       =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Secondquestion.wrl',vrut.HEAD)
secondQuestion.translate(0,0,1.5)
secondQuestion.scale(scfac,scfac,scfac)
secondQuestion.curtain(vrut.CLOSE)
line                 =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Underline.wrl',vrut.HEAD)
line.translate(0,lay,lz)
line.scale(scfac,scfac,scfac)
line.curtain(vrut.CLOSE)

ansblue              =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansbluehatch.wrl',vrut.HEAD)
ansyellw             =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyellowhatch.wrl',vrut.HEAD)
ansblue.translate(qax,qby,qaz)
ansblue.scale(ansscfac,ansscfac,ansscfac)
ansyellw.translate(qax,qby,qaz)
ansyellw.scale(ansscfac,ansscfac,ansscfac)
ansblue.curtain(vrut.CLOSE)
ansyellw.curtain(vrut.CLOSE)

ansupright          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansupright.wrl',vrut.HEAD)
ansrdown            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansrightdown.wrl',vrut.HEAD)
ansupdown           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansupsidedown.wrl',vrut.HEAD)
ansldown            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansleftdown.wrl',vrut.HEAD)
ansnoknow           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansupright.translate(qax,qay,qaz)
ansupright.scale(ansscfac,ansscfac,ansscfac)
ansrdown.translate(qax,qay,qaz)
ansrdown.scale(ansscfac,ansscfac,ansscfac)
ansupdown.translate(qax,qay,qaz)
ansupdown.scale(ansscfac,ansscfac,ansscfac)
ansldown.translate(qax,qay,qaz)
ansldown.scale(ansscfac,ansscfac,ansscfac)
ansnoknow.translate(qax,qay,qaz)
ansnoknow.scale(ansscfac,ansscfac,ansscfac)
ansupright.curtain(vrut.CLOSE)
ansrdown.curtain(vrut.CLOSE)
ansupdown.curtain(vrut.CLOSE)
ansldown.curtain(vrut.CLOSE)
ansnoknow.curtain(vrut.CLOSE)

ans3pit90f          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Anspitch90f.wrl',vrut.HEAD)
ans3pit90b          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Anspitch90b.wrl',vrut.HEAD)
ans3yaw180          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyaw180.wrl',vrut.HEAD)
ans3yaw90l          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyaw90l.wrl',vrut.HEAD)
ans3yaw90r          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyaw90r.wrl',vrut.HEAD)
ans3norot           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansnorotate.wrl',vrut.HEAD)
ans3noknow          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans3pit90f.translate(qax,qcy,qaz)
ans3pit90f.scale(ansscfac,ansscfac,ansscfac)
ans3pit90b.translate(qax,qcy,qaz)
ans3pit90b.scale(ansscfac,ansscfac,ansscfac)
ans3yaw180.translate(qax,qcy,qaz)
ans3yaw180.scale(ansscfac,ansscfac,ansscfac)
ans3yaw90l.translate(qax,qcy,qaz)
ans3yaw90l.scale(ansscfac,ansscfac,ansscfac)
ans3yaw90r.translate(qax,qcy,qaz)
ans3yaw90r.scale(ansscfac,ansscfac,ansscfac)
ans3norot.translate(qax,qcy,qaz)
ans3norot.scale(ansscfac,ansscfac,ansscfac)
ans3noknow.translate(qax,qcy,qaz)
ans3noknow.scale(ansscfac,ansscfac,ansscfac)
ans3pit90f.curtain(vrut.CLOSE)
ans3pit90b.curtain(vrut.CLOSE)
ans3yaw180.curtain(vrut.CLOSE)
ans3yaw90l.curtain(vrut.CLOSE)
ans3yaw90r.curtain(vrut.CLOSE)
ans3norot.curtain(vrut.CLOSE)
ans3noknow.curtain(vrut.CLOSE)
```

```
ansr180             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll180.wrl',vrut.HEAD)
ansr90l             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll90l.wrl',vrut.HEAD)
ansr90r             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll90r.wrl',vrut.HEAD)
ansr0               =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll0.wrl',vrut.HEAD)
ans4noknow          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansr0.translate(qax,qdy,qaz)
ansr0.scale(ansscfac,ansscfac,ansscfac)
ansr90l.translate(qax,qdy,qaz)
ansr90l.scale(ansscfac,ansscfac,ansscfac)
ansr90r.translate(qax,qdy,qaz)
ansr90r.scale(ansscfac,ansscfac,ansscfac)
ansr180.translate(qax,qdy,qaz)
ansr180.scale(ansscfac,ansscfac,ansscfac)
ans4noknow.translate(qax,qdy,qaz)
ans4noknow.scale(ansscfac,ansscfac,ansscfac)
ansr180.curtain(vrut.CLOSE)
ansr90l.curtain(vrut.CLOSE)
ansr90r.curtain(vrut.CLOSE)
ansr0.curtain(vrut.CLOSE)
ans4noknow.curtain(vrut.CLOSE)

anshealth           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
anshab              =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ansexp              =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
anseva              =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
anscont             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ansclose            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ansstore            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
anscentr            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans5noknow          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansexp.translate(qax,qay,qaz)
ansexp.scale(ansscfac,ansscfac,ansscfac)
anseva.translate(qax,qay,qaz)
anseva.scale(ansscfac,ansscfac,ansscfac)
ansstore.translate(qax,qay,qaz)
ansstore.scale(ansscfac,ansscfac,ansscfac)
anshealth.translate(qax,qay,qaz)
anshealth.scale(ansscfac,ansscfac,ansscfac)
anshab.translate(qax,qay,qaz)
anshab.scale(ansscfac,ansscfac,ansscfac)
anscentr.translate(qax,qay,qaz)
anscentr.scale(ansscfac,ansscfac,ansscfac)
anscont.translate(qax,qay,qaz)
anscont.scale(ansscfac,ansscfac,ansscfac)
ansclose.translate(qax,qay,qaz)
ansclose.scale(ansscfac,ansscfac,ansscfac)
ans5noknow.translate(qax,qay,qaz)
ans5noknow.scale(ansscfac,ansscfac,ansscfac)
anshealth.curtain(vrut.CLOSE)
anshab.curtain(vrut.CLOSE)
ansexp.curtain(vrut.CLOSE)
anseva.curtain(vrut.CLOSE)
anscont.curtain(vrut.CLOSE)
ansclose.curtain(vrut.CLOSE)
ansstore.curtain(vrut.CLOSE)
```

```
anscentr.curtain(vrut.CLOSE)
ans5noknow.curtain(vrut.CLOSE)

ans2health          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
ans2hab             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans2exp             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
ans2eva             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
ans2cont            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans2close           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans2store           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
ans2centr           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans6noknow          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans2exp.translate(qax,qby,qaz)
ans2exp.scale(ansscfac,ansscfac,ansscfac)
ans2eva.translate(qax,qby,qaz)
ans2eva.scale(ansscfac,ansscfac,ansscfac)
ans2store.translate(qax,qby,qaz)
ans2store.scale(ansscfac,ansscfac,ansscfac)
ans2health.translate(qax,qby,qaz)
ans2health.scale(ansscfac,ansscfac,ansscfac)
ans2hab.translate(qax,qby,qaz)
ans2hab.scale(ansscfac,ansscfac,ansscfac)
ans2centr.translate(qax,qby,qaz)
ans2centr.scale(ansscfac,ansscfac,ansscfac)
ans2cont.translate(qax,qby,qaz)
ans2cont.scale(ansscfac,ansscfac,ansscfac)
ans2close.translate(qax,qby,qaz)
ans2close.scale(ansscfac,ansscfac,ansscfac)
ans6noknow.translate(qax,qby,qaz)
ans6noknow.scale(ansscfac,ansscfac,ansscfac)
ans2health.curtain(vrut.CLOSE)
ans2hab.curtain(vrut.CLOSE)
ans2exp.curtain(vrut.CLOSE)
ans2eva.curtain(vrut.CLOSE)
ans2cont.curtain(vrut.CLOSE)
ans2close.curtain(vrut.CLOSE)
ans2store.curtain(vrut.CLOSE)
ans2centr.curtain(vrut.CLOSE)
ans6noknow.curtain(vrut.CLOSE)

ans3health          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
ans3hab             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans3exp             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
ans3eva             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
ans3cont            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans3close           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans3store           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
ans3centr           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans7noknow          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans3exp.translate(qax,qcy,qaz)
ans3exp.scale(ansscfac,ansscfac,ansscfac)
ans3eva.translate(qax,qcy,qaz)
ans3eva.scale(ansscfac,ansscfac,ansscfac)
ans3store.translate(qax,qcy,qaz)
ans3store.scale(ansscfac,ansscfac,ansscfac)
ans3health.translate(qax,qcy,qaz)
ans3health.scale(ansscfac,ansscfac,ansscfac)
ans3hab.translate(qax,qcy,qaz)
ans3hab.scale(ansscfac,ansscfac,ansscfac)
ans3centr.translate(qax,qcy,qaz)
ans3centr.scale(ansscfac,ansscfac,ansscfac)
ans3cont.translate(qax,qcy,qaz)
ans3cont.scale(ansscfac,ansscfac,ansscfac)
ans3close.translate(qax,qcy,qaz)
ans3close.scale(ansscfac,ansscfac,ansscfac)
ans7noknow.translate(qax,qcy,qaz)
ans7noknow.scale(ansscfac,ansscfac,ansscfac)
ans3health.curtain(vrut.CLOSE)
ans3hab.curtain(vrut.CLOSE)
ans3exp.curtain(vrut.CLOSE)
ans3eva.curtain(vrut.CLOSE)
ans3cont.curtain(vrut.CLOSE)
ans3close.curtain(vrut.CLOSE)
ans3store.curtain(vrut.CLOSE)
ans3centr.curtain(vrut.CLOSE)
ans7noknow.curtain(vrut.CLOSE)

ans4health          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
ans4hab             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans4exp             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
ans4eva             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
ans4cont            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans4close           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans4store           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
ans4centr           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans8noknow          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans4exp.translate(qax,qdy,qaz)
ans4exp.scale(ansscfac,ansscfac,ansscfac)
ans4eva.translate(qax,qdy,qaz)
ans4eva.scale(ansscfac,ansscfac,ansscfac)
ans4store.translate(qax,qdy,qaz)
ans4store.scale(ansscfac,ansscfac,ansscfac)
ans4health.translate(qax,qdy,qaz)
ans4health.scale(ansscfac,ansscfac,ansscfac)
ans4hab.translate(qax,qdy,qaz)
ans4hab.scale(ansscfac,ansscfac,ansscfac)
ans4centr.translate(qax,qdy,qaz)
ans4centr.scale(ansscfac,ansscfac,ansscfac)
ans4cont.translate(qax,qdy,qaz)
ans4cont.scale(ansscfac,ansscfac,ansscfac)
ans4close.translate(qax,qdy,qaz)
ans4close.scale(ansscfac,ansscfac,ansscfac)
ans8noknow.translate(qax,qdy,qaz)
ans8noknow.scale(ansscfac,ansscfac,ansscfac)
ans4health.curtain(vrut.CLOSE)
ans4hab.curtain(vrut.CLOSE)
ans4exp.curtain(vrut.CLOSE)
ans4eva.curtain(vrut.CLOSE)
ans4cont.curtain(vrut.CLOSE)
ans4close.curtain(vrut.CLOSE)
ans4store.curtain(vrut.CLOSE)
```

```
ans4centr.curtain(vrut.CLOSE)
ans8noknow.curtain(vrut.CLOSE)

columbus          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Columbus.wrl',vrut.HEAD)
stowage          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Stowage.wrl',vrut.HEAD)
zarya          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Zarya.wrl',vrut.HEAD)
destiny          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Destiny.wrl',vrut.HEAD)
habitation          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Habitation.wrl',vrut.HEAD)
kibo          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Kibo.wrl',vrut.HEAD)
reminder          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Reminder.wrl', vrut.HEAD)
columbus.scale(0.25,.25,.25)
stowage.scale(0.25,.25,.25)
zarya.scale(0.25,.25,.25)
destiny.scale(0.25,.25,.25)
habitation.scale(0.25,.25,.25)
kibo.scale(0.25,.25,.25)
columbus.translate(0,-.1,.5)
stowage.translate(0,-.1,.5)
zarya.translate(0,-.1,.5)
destiny.translate(0,-.1,.5)
habitation.translate(0,-.1,.5)
kibo.translate(0,-.1,.5)
reminder.translate(0,-.4,2)
columbus.curtain(vrut.CLOSE)
stowage.curtain(vrut.CLOSE)
zarya.curtain(vrut.CLOSE)
destiny.curtain(vrut.CLOSE)
habitation.curtain(vrut.CLOSE)
kibo.curtain(vrut.CLOSE)
reminder.curtain(vrut.CLOSE)


#
****************************************************************
**********************

# geographic location of each denoted point relative to center of
station
# geoxyz[endpoint] = x,y,z          endpoint can be module or at
nodes
# interesting note: you need to make this floating numbers!
# 6/4/2001 this is changed to just the nodes!
#geoxyz = [[-5.0,0.0,-5.25],[-5.0,0.0,5.25],[0.0,-5.0,-5.25],
#          [0.0,5.0,5.25],[5.0,0.0,-5.25],[5.0,0.0,5.25],
#          [0.0,0.0,0.0],[0.0,0.0,5.25],[0.0,0.0,-5.25]]

geoxyz = [[0.0,0.0,-5.25],[0.0,0.0,5.25],[0.0,0.0,-5.25],
        [0.0,0.0,5.25],[0.0,0.0,-5.25],[0.0,0.0,5.25],
        [0.0,0.0,0.0],[0.0,0.0,5.25],[0.0,0.0,-5.25]]

trials = [1,0,3,2,5,4]
initpost = [0,0,0,0,1,1]
# initpost[module] = 0 if in canonical, 1 if right shldr down; posture
you learn "route"

# segments[module] = endpoints passing necessary to get there from
center of station
#segments = [[8,0],[7,1],[8,2],[7,3],[8,4],[7,5]]
# segments is unnecessary because all you need is to do
geoxyz[module][2] !!!!

# segori[module] = orientation (relative to reset body_ori)
# orientation given as (nose,feet)
segori = [[2,3],[2,3],[3,4],[1,4],[0,5],[0,4]]

# other[count] = [other module, other module] within that hatch
other = [[3,5],[2,4],[1,5],[0,4],[3,1],[2,0]]

global          nextflag,module,count,routeflag,          showing,
set1flag,moremodflag
global q1flag,q2flag,q3flag,q4flag,q5flag,q6flag,q7flag,q8flag
global
q1index,q2index,q3index,q4index,q5index,q6index,q7index,q8index,b
egintime
nextflag = off
module = 0
count = 2
routeflag = on
set1flag = on
showing = off
moremodflag = on
q1flag = off
q2flag = off
q3flag = off
q4flag = off
q5flag = off
q6flag = off
q7flag = off
q8flag = off
q1index = 0
q2index = 0
q3index = 0
q4index = 0
q5index = 0
q6index = 0
q7index = 0
q8index = 0

# Timernums
layout = 1
closeTarget = 2
closeAstr = 3
placeAstrnt = 4
waitPOP = 5
waitFLYIN = 6
waitENTER = 7
showFirstQuest = 8
showSecondQuest = 9
scroller = 10
move2node = 11
move = 12
rotateSetup = 13
rotating = 14
taskpoint = 15
next = 16
waitNEXT = 17

#
****************************************************************
**********************

def initVariables():
   global module,count, routeflag, showing, q1flag, set1flag, nextflag
   global
q1index,q2index,q3index,q4index,q5index,q6index,q7index,q8index
   q1index = 0
   q2index = 0
   q3index = 0
   q4index = 0
   q5index = 0
   q6index = 0
```

```
    q7index = 0
    q8index = 0
    module = trials[count]
    # increase count??? where is it?
    if nextflag == on:
        nextflag = off
        routeflag = on
    showing = off
    q1flag = on
    line.translate(0,lay,lz)
    set1flag = on
    initpos = initpost[module]
    vrut.reset(vrut.BODY_ORI)
    vrut.reset(vrut.HEAD_POS)
    if initpos == 1:
        vrut.rotate(vrut.BODY_ORI, 0,0,90)

def showTarget(modulenum, tclose):
    if modulenum == 0:
        columbus.curtain(vrut.OPEN)
    elif modulenum == 1:
        stowage.curtain(vrut.OPEN)
    elif modulenum == 2:
        zarya.curtain(vrut.OPEN)
    elif modulenum == 3:
        destiny.curtain(vrut.OPEN)
    elif modulenum == 4:
        habitation.curtain(vrut.OPEN)
    elif modulenum == 5:
        kibo.curtain(vrut.OPEN)
    vrut.starttimer(closeTarget,tclose)

def closeAll():
    columbus.curtain(vrut.CLOSE)
    stowage.curtain(vrut.CLOSE)
    zarya.curtain(vrut.CLOSE)
    destiny.curtain(vrut.CLOSE)
    habitation.curtain(vrut.CLOSE)
    kibo.curtain(vrut.CLOSE)

def defAngles(nose,feet):
    # print(nose,feet)
    # these are defined as if starting from original orientation
    # 4/25/2001 editted nose 3 or 1
    print('nose,feet',nose,feet)
    if nose == 2 or nose == 0:
        pitch = 0
        if nose == 2: yaw = -90
        else: yaw = 90
        if feet == 4:
            if nose == 2:
                roll = -90
            else: roll = 90
        elif feet == 1:
            roll = 180
        elif feet == 5:
            if nose == 2:
                roll = 90
            else: roll = -90
        elif feet == 3:
            roll = 0

    if nose == 3 or nose == 1:
        if feet == 5:
            if nose == 3:
                pitch = -90
                yaw = 0
                roll = 0
            else:
                yaw = 180
                pitch = 90 # ?
                roll = 0
        elif feet == 0:
            if nose == 3:
                yaw = 90
                pitch = -90
                roll = 0
            else:
                yaw = -90
                pitch = 90
                roll = 0
        elif feet == 4:
            if nose == 3:
                yaw = 180
                pitch = -90
                roll = 0
            else:
                yaw = 0
                pitch = 90
                roll = 0
        elif feet == 2:
            if nose == 3:
                yaw = -90
                pitch = -90
                roll = 0
            else:
                yaw = 90
                pitch = 90
                roll = 0

    if nose == 4 or nose == 5:
        pitch = 0
        if nose == 4: yaw = 0
        else: yaw = 180
        if feet == 3:
            roll = 0
        elif feet == 1:
            roll = 180
        elif feet == 0:
            if nose == 4:
                roll = 90
            else: roll = -90
        elif feet == 2:
            if nose == 4:
                roll = -90
            else: roll = 90
    return [yaw,pitch,roll]

def Eul2Quats(psi,theta,phi):
    # psi == yaw; theta == pitch; phi == roll
    # Mode of implementation : model.avatar.rotate(-q2,q3,-
q1,2*math.acos(q0)*RAD2DEG)
    cr = math.cos(phi*DEG2RAD/2)
    cp = math.cos(theta*DEG2RAD/2)
    cy = math.cos(psi*DEG2RAD/2)
    sr = math.sin(phi*DEG2RAD/2)
    sp = math.sin(theta*DEG2RAD/2)
    sy = math.sin(psi*DEG2RAD/2)
    cpcy = cp*cy
    spsy = sp*sy
    q0 = cr*cpcy + sr*spsy
    q1 = sr*cpcy - cr*spsy
    q2 = cr*sp*cy + sr*cp*sy
    q3 = cr*cp*sy - sr*sp*cy
    quat0 = -q2
    quat1 = q3
    quat2 = -q1
    quat3 = 2*math.acos(q0)*RAD2DEG
    return [quat0,quat1,quat2,quat3]


def defLearnTurns(modulenum):
    if modulenum == 0:
        yaw = 90
        pitch = 0
        roll = 0
    elif modulenum == 1:
        yaw = -90
        pitch = 0
        roll = 0
    elif modulenum == 2:
        yaw = 0
        pitch = 90
        roll = 0
    elif modulenum == 3:
        yaw = 0
        pitch = 90
        roll = 0
    elif modulenum == 4:
        yaw = -90
        pitch = 0
        roll = 0
    elif modulenum == 5:
        yaw = 90
        pitch = 0
        roll = 0

    return [yaw,pitch,roll]

def closeOthers(flag):
    #global q1flag,q2flag,q3flag,q4flag,q5flag,q6flag,q7flag,q8flag
    if flag == 'q2flag':
        ansblue.curtain(vrut.CLOSE)
        ansyellw.curtain(vrut.CLOSE)
    elif flag == 'q1flag':
        ansupright.curtain(vrut.CLOSE)
        ansrdown.curtain(vrut.CLOSE)
        ansupdown.curtain(vrut.CLOSE)
        ansldown.curtain(vrut.CLOSE)
        ansnoknow.curtain(vrut.CLOSE)
    elif flag == 'q3flag':
        ans3pit90f.curtain(vrut.CLOSE)
        ans3pit90b.curtain(vrut.CLOSE)
        ans3yaw180.curtain(vrut.CLOSE)
        ans3yaw90l.curtain(vrut.CLOSE)
        ans3yaw90r.curtain(vrut.CLOSE)
        ans3norot.curtain(vrut.CLOSE)
        ans3noknow.curtain(vrut.CLOSE)
    elif flag == 'q4flag':
        ansr180.curtain(vrut.CLOSE)
        ansr90l.curtain(vrut.CLOSE)
        ansr90r.curtain(vrut.CLOSE)
        ansr0.curtain(vrut.CLOSE)
        ans4noknow.curtain(vrut.CLOSE)
    elif flag == 'q5flag':
        anshealth.curtain(vrut.CLOSE)
        anshab.curtain(vrut.CLOSE)
        ansexp.curtain(vrut.CLOSE)
        anseva.curtain(vrut.CLOSE)
        anscont.curtain(vrut.CLOSE)
        ansclose.curtain(vrut.CLOSE)
        ansstore.curtain(vrut.CLOSE)
        anscentr.curtain(vrut.CLOSE)
        ans5noknow.curtain(vrut.CLOSE)
    elif flag == 'q6flag':
        ans2health.curtain(vrut.CLOSE)
        ans2hab.curtain(vrut.CLOSE)
        ans2exp.curtain(vrut.CLOSE)
        ans2eva.curtain(vrut.CLOSE)
        ans2cont.curtain(vrut.CLOSE)
        ans2close.curtain(vrut.CLOSE)
        ans2store.curtain(vrut.CLOSE)
        ans2centr.curtain(vrut.CLOSE)
        ans6noknow.curtain(vrut.CLOSE)
    elif flag == 'q7flag':
        ans3health.curtain(vrut.CLOSE)
        ans3hab.curtain(vrut.CLOSE)
        ans3exp.curtain(vrut.CLOSE)
        ans3eva.curtain(vrut.CLOSE)
        ans3cont.curtain(vrut.CLOSE)
        ans3close.curtain(vrut.CLOSE)
        ans3store.curtain(vrut.CLOSE)
        ans3centr.curtain(vrut.CLOSE)
        ans7noknow.curtain(vrut.CLOSE)
    elif flag == 'q8flag':
        ans4health.curtain(vrut.CLOSE)
        ans4hab.curtain(vrut.CLOSE)
        ans4exp.curtain(vrut.CLOSE)
        ans4eva.curtain(vrut.CLOSE)
        ans4cont.curtain(vrut.CLOSE)
        ans4close.curtain(vrut.CLOSE)
        ans4store.curtain(vrut.CLOSE)
        ans4centr.curtain(vrut.CLOSE)
        ans8noknow.curtain(vrut.CLOSE)

def scrollAns(flag,which):
    if flag == 'q2flag':
        print(which)
        closeOthers('q2flag')
        if which == 0:
            ansblue.curtain(vrut.OPEN)
        elif which == 1:
            ansyellw.curtain(vrut.OPEN)
        else:
            print('error in q2flag')
    elif flag == 'q1flag':
        closeOthers('q1flag')
        if which == 0:
            ansupright.curtain(vrut.OPEN)
        elif which == 1:
            ansrdown.curtain(vrut.OPEN)
        elif which == 2:
            ansupdown.curtain(vrut.OPEN)
        elif which == 3:
            ansldown.curtain(vrut.OPEN)
        elif which == 4:
            ansnoknow.curtain(vrut.OPEN)
        else:
            print('error in q1flag')
    elif flag == 'q3flag':
        closeOthers('q3flag')
        if which == 0:
            ans3pit90f.curtain(vrut.OPEN)
        elif which == 1:
            ans3pit90b.curtain(vrut.OPEN)
        elif which == 2:
            ans3yaw180.curtain(vrut.OPEN)
        elif which == 3:
            ans3yaw90l.curtain(vrut.OPEN)
        elif which == 4:
            ans3yaw90r.curtain(vrut.OPEN)
        elif which == 5:
            ans3norot.curtain(vrut.OPEN)
        elif which == 6:
            ans3noknow.curtain(vrut.OPEN)
        else:
```

```
        print('error in q3flag')
    elif flag == 'q4flag':
        closeOthers('q4flag')
        if which == 0:
            ansr0.curtain(vrut.OPEN)
        elif which == 1:
            ansr90l.curtain(vrut.OPEN)
        elif which == 2:
            ansr90r.curtain(vrut.OPEN)
        elif which == 3:
            ansr180.curtain(vrut.OPEN)
        elif which == 4:
            ans4noknow.curtain(vrut.OPEN)
        else:
            print('error in q4flag')
    elif flag == 'q5flag':
        closeOthers('q5flag')
        if which == 0:
            ansexp.curtain(vrut.OPEN)
        elif which == 1:
            anseva.curtain(vrut.OPEN)
        elif which == 2:
            ansstore.curtain(vrut.OPEN)
        elif which == 3:
            anshealth.curtain(vrut.OPEN)
        elif which == 4:
            anshab.curtain(vrut.OPEN)
        elif which == 5:
            anscentr.curtain(vrut.OPEN)
        elif which == 6:
            anscont.curtain(vrut.OPEN)
        elif which == 7:
            ansclose.curtain(vrut.OPEN)
        elif which == 8:
            ans5noknow.curtain(vrut.OPEN)
        else:
            print('error in q5flag')
    elif flag == 'q6flag':
        closeOthers('q6flag')
        print('which', which)
        if which == 0:
            ans2exp.curtain(vrut.OPEN)
        elif which == 1:
            ans2eva.curtain(vrut.OPEN)
        elif which == 2:
            ans2store.curtain(vrut.OPEN)
        elif which == 3:
            ans2health.curtain(vrut.OPEN)
        elif which == 4:
            ans2hab.curtain(vrut.OPEN)
        elif which == 5:
            ans2centr.curtain(vrut.OPEN)
        elif which == 6:
            ans2cont.curtain(vrut.OPEN)
        elif which == 7:
            ans2close.curtain(vrut.OPEN)
        elif which == 8:
            ans6noknow.curtain(vrut.OPEN)
    elif flag == 'q7flag':
        closeOthers('q7flag')
        if which == 0:
            ans3exp.curtain(vrut.OPEN)
        elif which == 1:
            ans3eva.curtain(vrut.OPEN)
        elif which == 2:
            ans3store.curtain(vrut.OPEN)
        elif which == 3:
            ans3health.curtain(vrut.OPEN)
        elif which == 4:
            ans3hab.curtain(vrut.OPEN)
        elif which == 5:
            ans3centr.curtain(vrut.OPEN)
        elif which == 6:
            ans3cont.curtain(vrut.OPEN)
        elif which == 7:
            ans3close.curtain(vrut.OPEN)
        elif which == 8:
            ans7noknow.curtain(vrut.OPEN)
        else:
            print('error in q7flag')
    elif flag == 'q8flag':
        closeOthers('q8flag')
        if which == 0:
            ans4exp.curtain(vrut.OPEN)
        elif which == 1:
            ans4eva.curtain(vrut.OPEN)
        elif which == 2:
            ans4store.curtain(vrut.OPEN)
        elif which == 3:
            ans4health.curtain(vrut.OPEN)
        elif which == 4:
            ans4hab.curtain(vrut.OPEN)
        elif which == 5:
            ans4centr.curtain(vrut.OPEN)
        elif which == 6:
            ans4cont.curtain(vrut.OPEN)
        elif which == 7:
            ans4close.curtain(vrut.OPEN)
        elif which == 8:
            ans8noknow.curtain(vrut.OPEN)
        else:
            print('error in q8flag')


def closeEverything():
    global set1flag
    firstQuestion.curtain(vrut.CLOSE)
    secondQuestion.curtain(vrut.CLOSE)
    line.curtain(vrut.CLOSE)
    if set1flag == on:
        ansblue.curtain(vrut.CLOSE)
        ansyellw.curtain(vrut.CLOSE)
        ansupright.curtain(vrut.CLOSE)
        ansrdown.curtain(vrut.CLOSE)
        ansupdown.curtain(vrut.CLOSE)
        ansldown.curtain(vrut.CLOSE)
        ansnoknow.curtain(vrut.CLOSE)
        ans3pit90f.curtain(vrut.CLOSE)
        ans3pit90b.curtain(vrut.CLOSE)
        ans3yaw180.curtain(vrut.CLOSE)
        ans3yaw90l.curtain(vrut.CLOSE)
        ans3yaw90r.curtain(vrut.CLOSE)
        ans3norot.curtain(vrut.CLOSE)
        ans3noknow.curtain(vrut.CLOSE)
        ansr180.curtain(vrut.CLOSE)
        ansr90l.curtain(vrut.CLOSE)
        ansr90r.curtain(vrut.CLOSE)
        ansr0.curtain(vrut.CLOSE)
        ans4noknow.curtain(vrut.CLOSE)
    elif set1flag == off:
        anshealth.curtain(vrut.CLOSE)
        anshab.curtain(vrut.CLOSE)
        ansexp.curtain(vrut.CLOSE)
        anseva.curtain(vrut.CLOSE)
        anscont.curtain(vrut.CLOSE)
        ansclose.curtain(vrut.CLOSE)
        ansstore.curtain(vrut.CLOSE)
        anscentr.curtain(vrut.CLOSE)
        ans5noknow.curtain(vrut.CLOSE)
        ans2health.curtain(vrut.CLOSE)
        ans2hab.curtain(vrut.CLOSE)
        ans2exp.curtain(vrut.CLOSE)
        ans2eva.curtain(vrut.CLOSE)
        ans2cont.curtain(vrut.CLOSE)
        ans2close.curtain(vrut.CLOSE)
        ans2store.curtain(vrut.CLOSE)
        ans2centr.curtain(vrut.CLOSE)
        ans6noknow.curtain(vrut.CLOSE)
        ans3health.curtain(vrut.CLOSE)
        ans3hab.curtain(vrut.CLOSE)
        ans3exp.curtain(vrut.CLOSE)
        ans3eva.curtain(vrut.CLOSE)
        ans3cont.curtain(vrut.CLOSE)
        ans3close.curtain(vrut.CLOSE)
        ans3store.curtain(vrut.CLOSE)
        ans3centr.curtain(vrut.CLOSE)
        ans7noknow.curtain(vrut.CLOSE)
        ans4health.curtain(vrut.CLOSE)
        ans4hab.curtain(vrut.CLOSE)
        ans4exp.curtain(vrut.CLOSE)
        ans4eva.curtain(vrut.CLOSE)
        ans4cont.curtain(vrut.CLOSE)
        ans4close.curtain(vrut.CLOSE)
        ans4store.curtain(vrut.CLOSE)
        ans4centr.curtain(vrut.CLOSE)
        ans8noknow.curtain(vrut.CLOSE)


def detWhat():
    global routeflag,module,count,moremodflag, nextflag
    print('in detWhat')
    if routeflag == off:
        return module
    else:
        routeflag = off
        if moremodflag == on:
            moremodflag = off
            return other[count][0]
        else:
            moremodflag = on
            nextflag = on
            return other[count][1]


def myKeyboard(key):
    global begintime
    if key == ' ':
        reminder.curtain(vrut.CLOSE)
        vrut.starttimer(layout,0.001)
        begintime = time.time()
        out = str(time.time()-begintime) + '\t' + str(count) +
'\tcount\n'
        keys.write(out)
        keys.flush()
    elif key == 'r':
        tracker.command(6)  # RESET1    ... see multi.py in AndyL
        tracker.reset()
    elif key == '1':
        showTarget(module,.75)

def mytimer(timernum):
    global module, count, routeflag, showing, set1flag,moremodflag,
nextflag
    global maxx,maxy,maxz, x,y,z, stepx,stepy,stepz, startpoint,
begintime
    global yaw,pitch,roll, turny,turnp,turnr
    global q1flag,q1index,q2flag,q2index,q3flag,q3index,q4flag,q4index
    global q5flag,q5index,q6flag,q6index,q7flag,q7index,q8flag,q8index
    if timernum == layout:
        # initialize variables
        initVariables()
        # show target; close
        # show astronaut or pop up SIM
        if routeflag == on:
            showTarget(module,2)
            #out  =  str(initpost[module])+'\t'+str(module)+'\t'+str(-
1)+'\t'+str(-1)+'\t'+str(-1)+'\t'+str(-1)+'\t'+str(-1)+'\t'
            #out = str(initpost[module])+'\t'+str(module)+'\t'
            #file.write(out)
            #file.flush()
            vrut.starttimer(placeAstrnt,2.02)
        else:
            print( count, 'this is count')
            if count == 0 or count == 1 :
                vrut.starttimer(next,0.001)
            elif count == 2 or count == 3 or count == 4 or count == 5:
                startpoint = time.time()
                vrut.starttimer(taskpoint,2.02)
            else: print('there is an error in layout')
        # first questions
        # travel to the astronaut
        # second question
        # play sound/back to Control
        # show targe; close
        # pointing task
        # first question
        # second question
        # determine what are the two other targets
        # show target of other target 1; close
        # pointing task for other target 1
        # first question
        # second question
        # show target of other target 2; close
        # point
        # first question
        # second question

    elif timernum == closeTarget:
        closeAll()

    elif timernum == placeAstrnt:
        astronaut.curtain(vrut.OPEN)

astronaut.translate(geoxyz[module][0],geoxyz[module][1],geoxyz[mod
ule][2])
        angles = defAngles(segori[module][0],segori[module][1])
        print('angles',angles)
        quat = Eul2Quats(angles[0],angles[1],angles[2])
        astronaut.rotate(quat[0],quat[1],quat[2],quat[3])
        startpoint = time.time()
        if count == 0 or count == 1 :
            vrut.starttimer(showFirstQuest,2)
        elif count == 2 or count == 3 or count == 4 or count == 5:
            vrut.starttimer(waitFLYIN, 0.001)

    elif timernum == waitPOP:
        if sid.buttons() == sid.BUTTON10:
            if set1flag == on:
                vrut.starttimer(showFirstQuest,0.001)
            else:
                vrut.starttimer(showSecondQuest,0.001)
        else:
            vrut.starttimer(waitPOP,rate)

    elif timernum == showFirstQuest:
        if showing == on:
            showing = off
            closeEverything()
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\thide_quest_set1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(waitPOP,rate)
        elif showing == off:
            showing = on
            firstQuestion.curtain(vrut.OPEN)
            line.curtain(vrut.OPEN)
            scrollAns('q1flag',q1index)
            scrollAns('q2flag',q2index)
            scrollAns('q3flag',q3index)
            scrollAns('q4flag',q4index)
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\tshow_quest_set1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)

    elif timernum == showSecondQuest:
        if showing == on:
            showing = off
            closeEverything()
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\thide_quest_set2\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(waitPOP,rate)
        elif showing == off:
            showing = on
            secondQuestion.curtain(vrut.OPEN)
            line.curtain(vrut.OPEN)
            print('2nd question',q5index,q6index,q7index,q8index)
            scrollAns('q5flag',q5index)
            scrollAns('q6flag',q6index)
            scrollAns('q7flag',q7index)
            scrollAns('q8flag',q8index)
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\tshow_quest_set1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)

    elif timernum == scroller:
        if sid.buttons() == sid.BUTTON10 and routeflag == on:
            if set1flag == on:
                vrut.starttimer(showFirstQuest,0.001)
            else:
                vrut.starttimer(showSecondQuest,0.001)
            # watch out for a possible BUG HERE!
        elif sid.buttons() == sid.BUTTON1:
            # check for which question
            # scroll up (add)
            if q2flag == on:
                if q2index == 0:
                    q2index = 1
                else:
                    q2index = 0
                scrollAns('q2flag',q2index)
            elif q1flag == on:
                if q1index == 4:
                    q1index = 0
                else:
                    q1index = q1index + 1
                scrollAns('q1flag',q1index)
            elif q3flag == on:
                if q3index == 6:
                    q3index = 0
                else:
                    q3index = q3index + 1
                scrollAns('q3flag',q3index)
            elif q4flag == on:
                if q4index == 4:
                    q4index = 0
                else:
                    q4index = q4index + 1
                scrollAns('q4flag',q4index)
            elif q5flag == on:
                if q5index == 8:
                    q5index = 0
                else:
                    q5index = q5index + 1
                scrollAns('q5flag',q5index)
            elif q6flag == on:
                if q6index == 8:
                    q6index = 0
                else:
                    q6index = q6index + 1
                scrollAns('q6flag',q6index)
            elif q7flag == on:
                if q7index == 8:
                    q7index = 0
                else:
                    q7index = q7index + 1
                scrollAns('q7flag',q7index)
            elif q8flag == on:
                if q8index == 8:
                    q8index = 0
                else:
                    q8index = q8index + 1
                scrollAns('q8flag',q8index)
            vrut.starttimer(scroller,rate)
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON1)
+ '\tscroll_up\n'
            keys.write(out)
            keys.flush()

        elif sid.buttons() == sid.BUTTON4:
            # check for which question
            # scroll down (subtract)
            if q2flag == on:
                if q2index == 1:
                    q2index = 0
                else:
                    q2index = 1
                scrollAns('q2flag',q2index)
            elif q1flag == on:
                if q1index == 0:
                    q1index = 4
                else:
                    q1index = q1index - 1
                scrollAns('q1flag',q1index)
            elif q3flag == on:
                if q3index == 0:
                    q3index = 6
                else:
                    q3index = q3index - 1
                scrollAns('q3flag',q3index)
            elif q4flag == on:
                if q4index == 0:
                    q4index = 4
                else:
                    q4index = q4index - 1
                scrollAns('q4flag',q4index)
            elif q5flag == on:
                if q5index == 0:
```

```
            q5index = 8
         else:
            q5index = q5index - 1
         scrollAns('q5flag',q5index)
      elif q6flag == on:
         if q6index == 0:
            q6index = 8
         else:
            q6index = q6index - 1
         scrollAns('q6flag',q6index)
      elif q7flag == on:
         if q7index == 0:
            q7index = 8
         else:
            q7index = q7index - 1
         scrollAns('q7flag',q7index)
      elif q8flag == on:
         if q8index == 0:
            q8index = 8
         else:
            q8index = q8index - 1
         scrollAns('q8flag',q8index)
      vrut.starttimer(scroller,rate)
      out = str(time.time()-begintime) + '\t' + str(sid.BUTTON1)
+ '\tscroll_down\n'
      keys.write(out)
      keys.flush()

   elif sid.buttons() == 768:   # pressing sid.BUTTON9 and
sid.BUTTON10 together
      elapstime = time.time() - startpoint
      startpoint = time.time()
      closeEverything()
      showing = off
      out = str(time.time()-begintime) + '\t' + str(768) +
'\tenter_hide_quest_set\n'
      keys.write(out)
      keys.flush()
      if set1flag == on:
         q1flag = off
         q2flag = off
         q3flag = off
         q4flag = off
         q5flag = on
         out                                                    =
str(q1index)+'\t'+str(q2index)+'\t'+str(q3index)+'\t'+str(q4index)+'
\t'+ str(elapstime)+'\t'
         file.write(out)
         file.flush()
         if routeflag == on:
            #routeflag = off
            vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\EndQuest.wav')
            vrut.starttimer(waitFLYIN,rate)
         else:
            print('repeating??')
            #print(set1flag, routeflag)
            set1flag = off
            line.translate(0,lay,lz)
            startpoint = time.time()
            vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\EndQuest.wav')
            if nextflag == off and moremodflag == on:
               vrut.starttimer(showSecondQuest,0.1)
            else:
               q5flag = off
               q1flag = on
               line.translate(0,lay,lz)
               routeflag = on
               set1flag = on
               out    =    str(-1)+'\t'+str(-1)+'\t'+str(-1)+'\t'+str(-
1)+'\t'+str(-1)+'\t'+'\n'
               file.write(out)
               file.flush()
               if nextflag == on:
                  print('increasing count')
                  if count == maxtrials:
                     print('endprogram!!!')
                     vrut.translate(vrut.HEAD_POS, 0,100,0)
                     file.close()
                     keys.close()
                  else:
                     count = count + 1
                     vrut.starttimer(layout,0.001)
               else:
                  startpoint = time.time()
                  vrut.starttimer(taskpoint,1)
      else:
         q5flag = off
         q6flag = off
         q7flag = off
         q8flag = off
         q1flag = on
         line.translate(0,lay,lz)
         out                                                    =
str(q5index)+'\t'+str(q6index)+'\t'+str(q7index)+'\t'+str(q8index)+'
\t'+ str(elapstime)+'\n'
         file.write(out)
         file.flush()
         if routeflag == on:
            routeflag = off
            print('here')
            vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\Reminder.wav')
            vrut.starttimer(layout,2)
         else:  # i might not need this part anymore...
            routeflag = on
            set1flag = on
            if nextflag == on:
               print('increasing count')
               if count == maxtrials:
                  print('endprogram!!!')
                  vrut.translate(vrut.HEAD_POS, 0,100,0)
                  file.close()
                  keys.close()
               else:
                  count = count + 1
                  vrut.starttimer(layout,0.001)
            else:
               startpoint = time.time()
               vrut.starttimer(taskpoint,1)
            #vrut.starttimer(taskpoint,0.1)

   elif sid.buttons() == sid.BUTTON9:
      # enter answer  (record)
      # increase which question counter
      if q1flag == on:
         q1flag = off
         q2flag = on
         line.translate(0,lby,lz)
         scrollAns('q2flag',q2index)
         out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question1\n'
         keys.write(out)
         keys.flush()
         vrut.starttimer(scroller,rate)
      elif q2flag == on:
         q2flag = off
```

```
        q3flag = on
        line.translate(0,lcy,lz)
        scrollAns('q3flag',q3index)
        out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question2\n'
        keys.write(out)
        keys.flush()
        vrut.starttimer(scroller,rate)
      elif q3flag == on:
        q3flag = off
        q4flag = on
        line.translate(0,ldy,lz)
        scrollAns('q4flag',q4index)
        out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question3\n'
        keys.write(out)
        keys.flush()
        vrut.starttimer(scroller,rate)
      elif q4flag == on:
        q4flag = off
        q1flag = on
        line.translate(0,lay,lz)
        scrollAns('q1flag',q1index)
        out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question4\n'
        keys.write(out)
        keys.flush()
        vrut.starttimer(scroller,rate)
      elif q5flag == on:
        q5flag = off
        q6flag = on
        line.translate(0,lby,lz)
        scrollAns('q6flag',q6index)
        out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question5\n'
        keys.write(out)
        keys.flush()
        vrut.starttimer(scroller,rate)
      elif q6flag == on:
        q6flag = off
        q7flag = on
        line.translate(0,lcy,lz)
        scrollAns('q7flag',q7index)
        out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question6\n'
        keys.write(out)
        keys.flush()
        vrut.starttimer(scroller,rate)
      elif q7flag == on:
        q7flag = off
        q8flag = on
        line.translate(0,ldy,lz)
        scrollAns('q8flag',q8index)
        out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question7\n'
        keys.write(out)
        keys.flush()
        vrut.starttimer(scroller,rate)
      elif q8flag == on:
        q8flag = off
        q5flag = on
        line.translate(0,lay,lz)
        scrollAns('q5flag',q5index)
        out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question8\n'
        keys.write(out)
        keys.flush()
        vrut.starttimer(scroller,rate)
```

```
    else:
      vrut.starttimer(scroller,rate)

  elif timernum == next:
    if count == 0 or count == 1 :
      routeflag = on
      set1flag = on
      if count == maxtrials:
        print('endprogram!!!')
        vrut.translate(vrut.HEAD_POS,0,100,0)
        file.close()
        keys.close()
      else:
        count = count + 1
        vrut.starttimer(layout, 0.001)
    elif count == 2 or count == 3 or count == 4 or count == 5:
      routeflag = off
      vrut.starttimer(layout,0.001)
    else:
      print('there is a problem with the next timernum')

  elif timernum == waitFLYIN:
    if sid.buttons() == sid.BUTTON7:
      vrut.starttimer(closeAstr,2)
      vrut.starttimer(move2node,0.001)
      out = str(time.time()-begintime) + '\t' + str(sid.BUTTON7)
+ '\tflyin\n'
      keys.write(out)
      keys.flush()
    else:
      vrut.starttimer(waitFLYIN,rate)

  elif timernum == waitNEXT:
    if sid.buttons() == sid.BUTTON7:
      vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\Reminder.wav')
      vrut.starttimer(next,0.5)
      out = str(time.time()-begintime) + '\t' + str(sid.BUTTON7)
+ '\tend_of_route\n'
      keys.write(out)
      keys.flush()
    else:
      vrut.starttimer(waitNEXT,0.3)

  elif timernum == closeAstr:
    astronaut.curtain(vrut.CLOSE)

  elif timernum == move2node:
    maxx = geoxyz[module][0]
    maxy = geoxyz[module][1]
    maxz = geoxyz[module][2]
    stepx = maxx/flytime
    stepy = maxy/flytime
    stepz = maxz/flytime
    x = 0
    y = 0
    z = 0
    vrut.starttimer(move,0.001)

  elif timernum == move:
    if abs(x) < abs(maxx) or abs(y) < abs(maxy) or abs(z) <
abs(maxz):
      if abs(x) < abs(maxx):
        x = x + stepx
        vrut.translate(vrut.HEAD_POS, stepx,0,0)
      if abs(y) < abs(maxy):
        y = y + stepy
        vrut.translate(vrut.HEAD_POS, 0, stepy, 0)
      if abs(z) < abs(maxz):
            z = z + stepz
            vrut.translate(vrut.HEAD_POS, 0, 0, stepz)
        vrut.starttimer(move,0.001)
    else:
        vrut.starttimer(rotateSetup,1)

  elif timernum == rotateSetup:
    eulers = defLearnTurns(module)
    yaw = eulers[0]
    pitch = eulers[1]
    roll = eulers[2]
    turny = 0
    turnp = 0
    turnr = 0
    vrut.starttimer(rotating,0.001)

  elif timernum == rotating:
    if yaw <> 0 and turny <> yaw:
      if yaw < 0:
        if turny > yaw:
            turny = turny - speed
            vrut.rotate(vrut.BODY_ORI,-speed, 0, 0)
            vrut.starttimer(rotating,0.001)
      elif yaw > 0:
        if turny < yaw:
            turny = turny + speed
            vrut.rotate(vrut.BODY_ORI,speed, 0, 0)
            vrut.starttimer(rotating,0.001)

    elif pitch <> 0 and turnp <> pitch and turny == yaw:
      if pitch < 0:   # inverted these
        if turnp > pitch:
            turnp = turnp - speed
            vrut.rotate(vrut.BODY_ORI, 0, speed, 0)
            vrut.starttimer(rotating,0.001)
      elif pitch > 0:
        if turnp < pitch:
            turnp = turnp + speed
            vrut.rotate(vrut.BODY_ORI, 0, -speed, 0)
            vrut.starttimer(rotating,0.001)

    elif roll <> 0 and turnr <> roll and turny == yaw and turnp ==
pitch:
      if roll < 0:
        if turnr > roll:
            turnr = turnr - speed
            vrut.rotate(vrut.BODY_ORI, 0,0,-speed)
            vrut.starttimer(rotating, 0.001)
      elif roll > 0:
        if turnr < roll:
            turnr = turnr + speed
            vrut.rotate(vrut.BODY_ORI, 0,0,speed)
            vrut.starttimer(rotating, 0.001)
    elif turny == yaw and turnp == pitch and turnr == roll:
      set1flag = off
      line.translate(0,lay,lz)
      if count == 0 or count == 1 :
        vrut.starttimer(showSecondQuest,2)
      elif count == 2 or count == 3 or count == 4 or count == 5:
        vrut.starttimer(waitNEXT,0.001)

  elif timernum == taskpoint:
    what = detWhat()
    print('WHAT',what)

print('routeflag,nextflag,set1flag,moremodflag,showing',routeflag,next
flag,set1flag,moremodflag,showing)
    showTarget(what,2)
    out = str(initpost[count])+'\t'+str(what)+'\t'
    file.write(out)
    file.flush()
    arrow.curtain(vrut.OPEN)
    #startpoint = time.time()
    vrut.starttimer(waitENTER,0.001)
  elif timernum == waitENTER:
    if sid.buttons() == sid.BUTTON9:
      arrow.curtain(vrut.CLOSE)
      data = tracker.get()
      elapstime = time.time() - startpoint
      out                                                          =
str(data[3])+'\t'+str(data[4])+'\t'+str(data[5])+'\t'+str(data[6]*RAD2
DEG)+'\t'+str(elapstime)+'\t'
      file.write(out)
      file.flush()
      startpoint = time.time()
      out = str(time.time()-begintime) + '\t' + str(sid.BUTTON9)
+ '\tenter_pointing\n'
      keys.write(out)
      keys.flush()
      vrut.starttimer(showFirstQuest,1)
    else:
      vrut.starttimer(waitENTER,rate)



#
****************************************************************
**********************

# Setting up output files
exp_time = time.localtime(time.time())
timestring = time.strftime('%b%d_%H%M',exp_time)
filename              =                 'N:\Jessica\Thesis
SIM\Data\Cntrl\Phase2\Phase2b_Cntrl'+str(timestring)+'.dat'
file = open(filename,'w')
out                                                            =
'Posture\tTarget\tQuat0\tQuat1\tQuat2\tQuat3\tTime\tQ1\tQ2\
tQ3\tQ4\tTime1\tQ5\tQ6\tQ7\tQ8\tTime2\n'
file.write(out)
file.flush()
# second file out
filename2             =                 'N:\Jessica\Thesis
SIM\Data\Cntrl\Phase2\Phase2b_Cntrl'+str(timestring)+'_allkeys.d
at'
keys = open(filename2,'w')

vrut.callback(vrut.KEYBOARD_EVENT,'myKeyboard')
vrut.callback(vrut.TIMER_EVENT,'mytimer')

reminder.curtain(vrut.OPEN)
```

SIM Group (Parts A & B)

```
import vrut
import time
import math
import types
import string
import sid

# 6/4/2001  This is adapting trainingCntrl_ver3 to the new protocol.
# 6/6/2001  After finishing phase 2 training ver 4 for control,
adapting this for SIM
```

```
# 6/7/2001 done.
# 6/23/2001 protocol is too long. dividing phase 2 into two parts.

#
*****************************************************************
*********************
# KEY:
#
*****************************************************************
*********************


#    1                                 direction nomenclature
#    |
# 2 ----- 0 into the page --   4
#    |                   out of the page -- 5
#    3
#
# Target difficulty:
#          0: modules 0 or 1
#          1: modules 4 or 5
#          2: modules 2 or 3
# FarNode:        I   MIGHT   HAVE   GOTTEN   THIS
BACKWARDS... 4/2/2001
#      0: modules 1, 3, or 5
#          1: modules 0, 2, or 4
# Posture:  relative to module 6
#          0: roll, with nose at 4 & feet at 1
#          1: pitch, (positive) with nose at 1 & feet at 4
#          2: roll, with nose at 4 & feet at 2
#          3: pitch, (negative) with nose at 3 & feet at 5
#
# Module labelling:
#          0: Columbus, European Module      == Experiment
#          1: Stowage Module                 == EVA
#          2: Zarya, Russian Control Module  == Storage
#          3: Destiny, U.S. Module           == Health Fitness
#          4: Habitation Module              == Habitation
#          5: Kibo, Japanese Module          == Centrifuge
#          6: Zvezda, Russian Service Module  == Control
#          Nodes
#          7: FarNode 1, hatch at end of zvezda (blue)          #
6/4/2001  had these reversed
#          8: FarNode 0, hatch at other end of zvezda (yellow)
#
#                                                      Constants
*****************************************************************
************


on = 1
off = 0
flytime = 100
speed = .25
flytime = 85
maxtrials = 1   # double check!
rate = 0.3
modscfac = 0.075

qax = .3
qay = .435
qby = .15
qcy = -.15
qdy = -.435
qaz = 2.05
lay = .275
lby = 0
lcy =-.275
ldy = -.57
lz  = 2.0
ansscfac = 4
scfac = 2.25

HMD = off

RAD2DEG = 57.295779513082323
DEG2RAD = 1/57.295779513082323

#
*****************************************************************
*********************

if HMD == on:
  vrut.go(vrut.HMD | vrut.STEREO)
else:
  vrut.go(vrut.CONSOLE)
  vrut.translate(vrut.HEAD_POS, 0,-1.82,0)

#
*****************************************************************
*********************
#          Sensor          &          Commands
*****************************************************************
****

MODE_EULER = 1
MODE_QUAT = 2
MODE_QUAT_NORM = 3
USE_REFFRAME = 4
STORE_REFFRAME = 5
RESET1 = 6
RESET2 = 7
RESET3 = 8
RESET4 = 9
RATCHET1 = 10
RATCHET2 = 11
RATCHET3 = 12
RATCHET4 = 13

# head tracker : must do this one first
tracker = vrut.addsensor('isensemulti_beta2')
vrut.tracker()
tracker.command(RESET1)  # RESET1
tracker.reset()

# wand sensor
wand = vrut.addsensor('isensemulti_beta2')
#tracker.command(MODE_QUAT_NORM)
#tracker.command(RATCHET2)
tracker.command(RESET2)
tracker.reset()

#
*****************************************************************
*********************
# Loading objects and their initial positions

global dispx, dispy, dispz          # initial displacement of model
dispx = 0        # offset in the x
dispy = -.15     # offset in the y
dispz = 1.3      # offset in the z
offsetx = dispx
offsety = dispy
offsetz = dispz

class mod_ava(vrut.EventClass):
      def __init__(modava):  #  for  now.  could  be  more,  see
SIM_ratchet...
           vrut.EventClass.__init__(modava)
```

```
        modava.model        =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\station_astr_obj.wrl',vrut.HEAD)
        modava.avatar = modava.model.getchild('astronaut')
        modava.avatar2 = modava.model.getchild('astrnt')

model = mod_ava()
model.model.scale(modscfac,modscfac,modscfac)
model.model.translate(dispx,dispy,dispz)
model.model.curtain(vrut.CLOSE) # initially, model hidden

station = vrut.addchild('n:\Jessica\Thesis SIM\Models\Station2.wrl')
arrow            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Cross.wrl',vrut.HEAD)
arrow.translate(0,0,.75)
arrow.curtain(vrut.CLOSE)

firstQuestion        =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Firstquestion.wrl',vrut.HEAD)
firstQuestion.translate(0,0,1.5)
firstQuestion.scale(scfac,scfac,scfac)
firstQuestion.curtain(vrut.CLOSE)
secondQuestion        =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Secondquestion.wrl',vrut.HEAD)
secondQuestion.translate(0,0,1.5)
secondQuestion.scale(scfac,scfac,scfac)
secondQuestion.curtain(vrut.CLOSE)
line            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Underline.wrl',vrut.HEAD)
line.translate(0,lay,lz)
line.scale(scfac,scfac,scfac)
line.curtain(vrut.CLOSE)

ansblue            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansbluehatch.wrl',vrut.HEAD)
ansyellw            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyellowhatch.wrl',vrut.HEAD)
ansblue.translate(qax,qby,qaz)
ansblue.scale(ansscfac,ansscfac,ansscfac)
ansyellw.translate(qax,qby,qaz)
ansyellw.scale(ansscfac,ansscfac,ansscfac)
ansblue.curtain(vrut.CLOSE)
ansyellw.curtain(vrut.CLOSE)

ansupright            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansupright.wrl',vrut.HEAD)
ansrdown            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansrightdown.wrl',vrut.HEAD)
ansupdown            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansupsidedown.wrl',vrut.HEAD)
ansldown            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansleftdown.wrl',vrut.HEAD)
ansnoknow            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansupright.translate(qax,qay,qaz)
ansupright.scale(ansscfac,ansscfac,ansscfac)
ansrdown.translate(qax,qay,qaz)
ansrdown.scale(ansscfac,ansscfac,ansscfac)
ansupdown.translate(qax,qay,qaz)
ansupdown.scale(ansscfac,ansscfac,ansscfac)
ansldown.translate(qax,qay,qaz)
ansldown.scale(ansscfac,ansscfac,ansscfac)
ansnoknow.translate(qax,qay,qaz)
ansnoknow.scale(ansscfac,ansscfac,ansscfac)
ansupright.curtain(vrut.CLOSE)
ansrdown.curtain(vrut.CLOSE)
ansupdown.curtain(vrut.CLOSE)
ansldown.curtain(vrut.CLOSE)
ansnoknow.curtain(vrut.CLOSE)
```

```
ans3pit90f            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Anspitch90f.wrl',vrut.HEAD)
ans3pit90b            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Anspitch90b.wrl',vrut.HEAD)
ans3yaw180            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyaw180.wrl',vrut.HEAD)
ans3yaw90l            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyaw90l.wrl',vrut.HEAD)
ans3yaw90r            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyaw90r.wrl',vrut.HEAD)
ans3norot            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansnorotate.wrl',vrut.HEAD)
ans3noknow            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans3pit90f.translate(qax,qcy,qaz)
ans3pit90f.scale(ansscfac,ansscfac,ansscfac)
ans3pit90b.translate(qax,qcy,qaz)
ans3pit90b.scale(ansscfac,ansscfac,ansscfac)
ans3yaw180.translate(qax,qcy,qaz)
ans3yaw180.scale(ansscfac,ansscfac,ansscfac)
ans3yaw90l.translate(qax,qcy,qaz)
ans3yaw90l.scale(ansscfac,ansscfac,ansscfac)
ans3yaw90r.translate(qax,qcy,qaz)
ans3yaw90r.scale(ansscfac,ansscfac,ansscfac)
ans3norot.translate(qax,qcy,qaz)
ans3norot.scale(ansscfac,ansscfac,ansscfac)
ans3noknow.translate(qax,qcy,qaz)
ans3noknow.scale(ansscfac,ansscfac,ansscfac)
ans3pit90f.curtain(vrut.CLOSE)
ans3pit90b.curtain(vrut.CLOSE)
ans3yaw180.curtain(vrut.CLOSE)
ans3yaw90l.curtain(vrut.CLOSE)
ans3yaw90r.curtain(vrut.CLOSE)
ans3norot.curtain(vrut.CLOSE)
ans3noknow.curtain(vrut.CLOSE)

ansr180            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll180.wrl',vrut.HEAD)
ansr90l            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll90l.wrl',vrut.HEAD)
ansr90r            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll90r.wrl',vrut.HEAD)
ansr0            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll0.wrl',vrut.HEAD)
ans4noknow            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansr0.translate(qax,qdy,qaz)
ansr0.scale(ansscfac,ansscfac,ansscfac)
ansr90l.translate(qax,qdy,qaz)
ansr90l.scale(ansscfac,ansscfac,ansscfac)
ansr90r.translate(qax,qdy,qaz)
ansr90r.scale(ansscfac,ansscfac,ansscfac)
ansr180.translate(qax,qdy,qaz)
ansr180.scale(ansscfac,ansscfac,ansscfac)
ans4noknow.translate(qax,qdy,qaz)
ans4noknow.scale(ansscfac,ansscfac,ansscfac)
ansr180.curtain(vrut.CLOSE)
ansr90l.curtain(vrut.CLOSE)
ansr90r.curtain(vrut.CLOSE)
ansr0.curtain(vrut.CLOSE)
ans4noknow.curtain(vrut.CLOSE)

anshealth            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
anshab            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ansexp            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
```

```
anseva = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modeva.wrl',vrut.HEAD)
anscont = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modcontrol.wrl',vrut.HEAD)
ansclose = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ansstore = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modstorage.wrl',vrut.HEAD)
anscentr = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans5noknow = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansexp.translate(qax,qay,qaz)
ansexp.scale(ansscfac,ansscfac,ansscfac)
anseva.translate(qax,qay,qaz)
anseva.scale(ansscfac,ansscfac,ansscfac)
ansstore.translate(qax,qay,qaz)
ansstore.scale(ansscfac,ansscfac,ansscfac)
anshealth.translate(qax,qay,qaz)
anshealth.scale(ansscfac,ansscfac,ansscfac)
anshab.translate(qax,qay,qaz)
anshab.scale(ansscfac,ansscfac,ansscfac)
anscentr.translate(qax,qay,qaz)
anscentr.scale(ansscfac,ansscfac,ansscfac)
anscont.translate(qax,qay,qaz)
anscont.scale(ansscfac,ansscfac,ansscfac)
ansclose.translate(qax,qay,qaz)
ansclose.scale(ansscfac,ansscfac,ansscfac)
ans5noknow.translate(qax,qay,qaz)
ans5noknow.scale(ansscfac,ansscfac,ansscfac)
anshealth.curtain(vrut.CLOSE)
anshab.curtain(vrut.CLOSE)
ansexp.curtain(vrut.CLOSE)
anseva.curtain(vrut.CLOSE)
anscont.curtain(vrut.CLOSE)
ansclose.curtain(vrut.CLOSE)
ansstore.curtain(vrut.CLOSE)
anscentr.curtain(vrut.CLOSE)
ans5noknow.curtain(vrut.CLOSE)

ans2health = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modhealth.wrl',vrut.HEAD)
ans2hab = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans2exp = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modexperiment.wrl',vrut.HEAD)
ans2eva = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modeva.wrl',vrut.HEAD)
ans2cont = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans2close = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans2store = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modstorage.wrl',vrut.HEAD)
ans2centr = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans6noknow = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans2exp.translate(qax,qby,qaz)
ans2exp.scale(ansscfac,ansscfac,ansscfac)
ans2eva.translate(qax,qby,qaz)
ans2eva.scale(ansscfac,ansscfac,ansscfac)
ans2store.translate(qax,qby,qaz)
ans2store.scale(ansscfac,ansscfac,ansscfac)
ans2health.translate(qax,qby,qaz)
ans2health.scale(ansscfac,ansscfac,ansscfac)
ans2hab.translate(qax,qby,qaz)
ans2hab.scale(ansscfac,ansscfac,ansscfac)
ans2centr.translate(qax,qby,qaz)
ans2centr.scale(ansscfac,ansscfac,ansscfac)
ans2cont.translate(qax,qby,qaz)
ans2cont.scale(ansscfac,ansscfac,ansscfac)
ans2close.translate(qax,qby,qaz)
ans2close.scale(ansscfac,ansscfac,ansscfac)
ans6noknow.translate(qax,qby,qaz)
ans6noknow.scale(ansscfac,ansscfac,ansscfac)
ans2health.curtain(vrut.CLOSE)
ans2hab.curtain(vrut.CLOSE)
ans2exp.curtain(vrut.CLOSE)
ans2eva.curtain(vrut.CLOSE)
ans2cont.curtain(vrut.CLOSE)
ans2close.curtain(vrut.CLOSE)
ans2store.curtain(vrut.CLOSE)
ans2centr.curtain(vrut.CLOSE)
ans6noknow.curtain(vrut.CLOSE)

ans3health = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modhealth.wrl',vrut.HEAD)
ans3hab = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans3exp = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modexperiment.wrl',vrut.HEAD)
ans3eva = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modeva.wrl',vrut.HEAD)
ans3cont = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans3close = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans3store = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modstorage.wrl',vrut.HEAD)
ans3centr = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans7noknow = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans3exp.translate(qax,qcy,qaz)
ans3exp.scale(ansscfac,ansscfac,ansscfac)
ans3eva.translate(qax,qcy,qaz)
ans3eva.scale(ansscfac,ansscfac,ansscfac)
ans3store.translate(qax,qcy,qaz)
ans3store.scale(ansscfac,ansscfac,ansscfac)
ans3health.translate(qax,qcy,qaz)
ans3health.scale(ansscfac,ansscfac,ansscfac)
ans3hab.translate(qax,qcy,qaz)
ans3hab.scale(ansscfac,ansscfac,ansscfac)
ans3centr.translate(qax,qcy,qaz)
ans3centr.scale(ansscfac,ansscfac,ansscfac)
ans3cont.translate(qax,qcy,qaz)
ans3cont.scale(ansscfac,ansscfac,ansscfac)
ans3close.translate(qax,qcy,qaz)
ans3close.scale(ansscfac,ansscfac,ansscfac)
ans7noknow.translate(qax,qcy,qaz)
ans7noknow.scale(ansscfac,ansscfac,ansscfac)
ans3health.curtain(vrut.CLOSE)
ans3hab.curtain(vrut.CLOSE)
ans3exp.curtain(vrut.CLOSE)
ans3eva.curtain(vrut.CLOSE)
ans3cont.curtain(vrut.CLOSE)
ans3close.curtain(vrut.CLOSE)
ans3store.curtain(vrut.CLOSE)
ans3centr.curtain(vrut.CLOSE)
ans7noknow.curtain(vrut.CLOSE)

ans4health = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modhealth.wrl',vrut.HEAD)
ans4hab = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans4exp = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modexperiment.wrl',vrut.HEAD)
```

```
ans4eva = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modeva.wrl',vrut.HEAD)
ans4cont = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans4close = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans4store = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modstorage.wrl',vrut.HEAD)
ans4centr = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans8noknow = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans4exp.translate(qax,qdy,qaz)
ans4exp.scale(ansscfac,ansscfac,ansscfac)
ans4eva.translate(qax,qdy,qaz)
ans4eva.scale(ansscfac,ansscfac,ansscfac)
ans4store.translate(qax,qdy,qaz)
ans4store.scale(ansscfac,ansscfac,ansscfac)
ans4health.translate(qax,qdy,qaz)
ans4health.scale(ansscfac,ansscfac,ansscfac)
ans4hab.translate(qax,qdy,qaz)
ans4hab.scale(ansscfac,ansscfac,ansscfac)
ans4centr.translate(qax,qdy,qaz)
ans4centr.scale(ansscfac,ansscfac,ansscfac)
ans4cont.translate(qax,qdy,qaz)
ans4cont.scale(ansscfac,ansscfac,ansscfac)
ans4close.translate(qax,qdy,qaz)
ans4close.scale(ansscfac,ansscfac,ansscfac)
ans8noknow.translate(qax,qdy,qaz)
ans8noknow.scale(ansscfac,ansscfac,ansscfac)
ans4health.curtain(vrut.CLOSE)
ans4hab.curtain(vrut.CLOSE)
ans4exp.curtain(vrut.CLOSE)
ans4eva.curtain(vrut.CLOSE)
ans4cont.curtain(vrut.CLOSE)
ans4close.curtain(vrut.CLOSE)
ans4store.curtain(vrut.CLOSE)
ans4centr.curtain(vrut.CLOSE)
ans8noknow.curtain(vrut.CLOSE)

columbus = vrut.addchild('n:\Jessica\Thesis SIM\Models\Columbus.wrl',vrut.HEAD)
stowage = vrut.addchild('n:\Jessica\Thesis SIM\Models\Stowage.wrl',vrut.HEAD)
zarya = vrut.addchild('n:\Jessica\Thesis SIM\Models\Zarya.wrl',vrut.HEAD)
destiny = vrut.addchild('n:\Jessica\Thesis SIM\Models\Destiny.wrl',vrut.HEAD)
habitation = vrut.addchild('n:\Jessica\Thesis SIM\Models\Habitation.wrl',vrut.HEAD)
kibo = vrut.addchild('n:\Jessica\Thesis SIM\Models\Kibo.wrl',vrut.HEAD)
reminder = vrut.addchild('n:\Jessica\Thesis SIM\Models\Reminder.wrl', vrut.HEAD)
columbus.scale(0.25,.25,.25)
stowage.scale(0.25,.25,.25)
zarya.scale(0.25,.25,.25)
destiny.scale(0.25,.25,.25)
habitation.scale(0.25,.25,.25)
kibo.scale(0.25,.25,.25)
columbus.translate(0,-.1,.5)
stowage.translate(0,-.1,.5)
zarya.translate(0,-.1,.5)
destiny.translate(0,-.1,.5)
habitation.translate(0,-.1,.5)
kibo.translate(0,-.1,.5)
reminder.translate(0,-.4,2)
columbus.curtain(vrut.CLOSE)
stowage.curtain(vrut.CLOSE)
zarya.curtain(vrut.CLOSE)
destiny.curtain(vrut.CLOSE)
habitation.curtain(vrut.CLOSE)
kibo.curtain(vrut.CLOSE)
reminder.curtain(vrut.CLOSE)


# ********************************************************************************

# geographic location of each denoted point relative to center of station
# geoxyz[endpoint] = x,y,z        endpoint can be module or at nodes
# interesting note: you need to make this floating numbers!
# 6/4/2001 this is changed to just the nodes!
#geoxyz = [[-5.0,0.0,-5.25],[-5.0,0.0,5.25],[0.0,-5.0,-5.25],
#        [0.0,5.0,5.25],[5.0,0.0,-5.25],[5.0,0.0,5.25],
#        [0.0,0.0,0.0],[0.0,0.0,5.25],[0.0,0.0,-5.25]]

geoxyz = [[0.0,0.0,-5.25],[0.0,0.0,5.25],[0.0,0.0,-5.25],
        [0.0,0.0,5.25],[0.0,0.0,-5.25],[0.0,0.0,5.25],
        [0.0,0.0,0.0],[0.0,0.0,5.25],[0.0,0.0,-5.25]]

trials = [1,0,3,2,5,4]
initpost = [0,0,0,0,1,1]
# initpost[module] = 0 if in canonical, 1 if right shldr down;  posture you learn "route"

# segments[module] = endpoints passing necessary to get there from center of station
#segments = [[8,0],[7,1],[8,2],[7,3],[8,4],[7,5]]
# segments is unnecessary because all you need is to do geoxyz[module][2] !!!!

# segori[module] = orientation (relative to reset body_ori)
# orientation given as (nose,feet)
segori = [[2,3],[2,3],[3,4],[1,4],[0,5],[0,4]]

# other[count] = [other module, other module]  within that hatch
other = [[3,1],[0,2],[5,1],[0,4],[5,3],[2,4]]

global nextflag,module,count,routeflag, showing, set1flag,moremodflag
global q1flag,q2flag,q3flag,q4flag,q5flag,q6flag,q7flag,q8flag
global q1index,q2index,q3index,q4index,q5index,q6index,q7index,q8index
global data, prevdata, model_flag, flyflag, first, begintime
nextflag = off
module = 0
count = 0
routeflag = on
set1flag = on
showing = off
moremodflag = on
q1flag = off
q2flag = off
q3flag = off
q4flag = off
q5flag = off
q6flag = off
q7flag = off
q8flag = off
q1index = 0
q2index = 0
q3index = 0
q4index = 0
q5index = 0
```

```
q6index = 0
q7index = 0
q8index = 0
data = wand.get()
prevdata = [0,0,0]
model_flag = off
flyflag = off
first = on

# Timernums
layout = 1
closeTarget = 2
move_model = 3
placeAstrnt = 4
waitPOP = 5 #?
waitFLYIN = 6
waitENTER = 7
showFirstQuest = 8
showSecondQuest = 9
scroller = 10
flyin = 11
flyin2 = 12
waitRATCHET= 13
taskpoint = 14
next = 15



#
*****************************************************************
**********************

def initVariables():
   global module,count, routeflag, showing, q1flag, set1flag,
nextflag,flyflag
   global
q1index,q2index,q3index,q4index,q5index,q6index,q7index,q8index
   q1index = 0
   q2index = 0
   q3index = 0
   q4index = 0
   q5index = 0
   q6index = 0
   q7index = 0
   q8index = 0
   module = trials[count]
   # increase count??? where is it?
   if nextflag == on:
      nextflag = off
      routeflag = on
   showing = off
   flyflag = off
   q1flag = on
   line.translate(0,lay,lz)
   set1flag = on
   initpos = initpost[module]
   vrut.reset(vrut.BODY_ORI)
   vrut.reset(vrut.HEAD_POS)
   if initpos == 1:
      vrut.rotate(vrut.BODY_ORI, 0,0,90)

def showTarget(modulenum, tclose):
   if modulenum == 0:
      columbus.curtain(vrut.OPEN)
   elif modulenum == 1:
      stowage.curtain(vrut.OPEN)
   elif modulenum == 2:
      zarya.curtain(vrut.OPEN)
   elif modulenum == 3:
      destiny.curtain(vrut.OPEN)
   elif modulenum == 4:
      habitation.curtain(vrut.OPEN)
   elif modulenum == 5:
      kibo.curtain(vrut.OPEN)
   vrut.starttimer(closeTarget,tclose)

def closeAll():
   columbus.curtain(vrut.CLOSE)
   stowage.curtain(vrut.CLOSE)
   zarya.curtain(vrut.CLOSE)
   destiny.curtain(vrut.CLOSE)
   habitation.curtain(vrut.CLOSE)
   kibo.curtain(vrut.CLOSE)

def defAngles(nose,feet):
   # print(nose,feet)
   # these are defined as if starting from original orientation
   # 4/25/2001 editted nose 3 or 1
   print('nose,feet',nose,feet)
   if nose == 2 or nose == 0:
      pitch = 0
      if nose == 2: yaw = -90
      else: yaw = 90
      if feet == 4:
         if nose == 2:
            roll = -90
         else: roll = 90
      elif feet == 1:
         roll = 180
      elif feet == 5:
         if nose == 2:
            roll = 90
         else: roll = -90
      elif feet == 3:
         roll = 0

   if nose == 3 or nose == 1:
      if feet == 5:
         if nose == 3:
            pitch = -90
            yaw = 0
            roll = 0
         else:
            yaw = 180
            pitch = 90 # ?
            roll = 0
      elif feet == 0:
         if nose == 3:
            yaw = 90
            pitch = -90
            roll = 0
         else:
            yaw = -90
            pitch = 90
            roll = 0
      elif feet == 4:
         if nose == 3:
            yaw = 180
            pitch = -90
            roll = 0
         else:
            yaw = 0
            pitch = 90
            roll = 0
      elif feet == 2:
         if nose == 3:
            yaw = -90
            pitch = -90
            roll = 0
        else:
            yaw = 90
            pitch = 90
            roll = 0

    if nose == 4 or nose == 5:
        pitch = 0
        if nose == 4: yaw = 0
        else: yaw = 180
        if feet == 3:
            roll = 0
        elif feet == 1:
            roll = 180
        elif feet == 0:
            if nose == 4:
                roll = 90
            else: roll = -90
        elif feet == 2:
            if nose == 4:
                roll = -90
            else: roll = 90
    return [yaw,pitch,roll]

def Eul2Quats(psi,theta,phi):
    # psi == yaw; theta == pitch; phi == roll
    # Mode of implementation : model.avatar.rotate(-q2,q3,-
q1,2*math.acos(q0)*RAD2DEG)
    cr = math.cos(phi*DEG2RAD/2)
    cp = math.cos(theta*DEG2RAD/2)
    cy = math.cos(psi*DEG2RAD/2)
    sr = math.sin(phi*DEG2RAD/2)
    sp = math.sin(theta*DEG2RAD/2)
    sy = math.sin(psi*DEG2RAD/2)
    cpcy = cp*cy
    spsy = sp*sy
    q0 = cr*cpcy + sr*spsy
    q1 = sr*cpcy - cr*spsy
    q2 = cr*sp*cy + sr*cp*sy
    q3 = cr*cp*sy - sr*sp*cy
    quat0 = -q2
    quat1 = q3
    quat2 = -q1
    quat3 = 2*math.acos(q0)*RAD2DEG
    return [quat0,quat1,quat2,quat3]


def defLearnTurns(modulenum):
    if modulenum == 0:
        yaw = 90
        pitch = 0
        roll = 0
    elif modulenum == 1:
        yaw = -90
        pitch = 0
        roll = 0
    elif modulenum == 2:
        yaw = 0
        pitch = 90
        roll = 0
    elif modulenum == 3:
        yaw = 0
        pitch = 90
        roll = 0
    elif modulenum == 4:
        yaw = -90
        pitch = 0
        roll = 0
    elif modulenum == 5:
        yaw = 90
        pitch = 0
        roll = 0

    return [yaw,pitch,roll]

def closeOthers(flag):
    #global q1flag,q2flag,q3flag,q4flag,q5flag,q6flag,q7flag,q8flag
    if flag == 'q2flag':
        ansblue.curtain(vrut.CLOSE)
        ansyellw.curtain(vrut.CLOSE)
    elif flag == 'q1flag':
        ansupright.curtain(vrut.CLOSE)
        ansrdown.curtain(vrut.CLOSE)
        ansupdown.curtain(vrut.CLOSE)
        ansldown.curtain(vrut.CLOSE)
        ansnoknow.curtain(vrut.CLOSE)
    elif flag == 'q3flag':
        ans3pit90f.curtain(vrut.CLOSE)
        ans3pit90b.curtain(vrut.CLOSE)
        ans3yaw180.curtain(vrut.CLOSE)
        ans3yaw90l.curtain(vrut.CLOSE)
        ans3yaw90r.curtain(vrut.CLOSE)
        ans3norot.curtain(vrut.CLOSE)
        ans3noknow.curtain(vrut.CLOSE)
    elif flag == 'q4flag':
        ansr180.curtain(vrut.CLOSE)
        ansr90l.curtain(vrut.CLOSE)
        ansr90r.curtain(vrut.CLOSE)
        ansr0.curtain(vrut.CLOSE)
        ans4noknow.curtain(vrut.CLOSE)
    elif flag == 'q5flag':
        anshealth.curtain(vrut.CLOSE)
        anshab.curtain(vrut.CLOSE)
        ansexp.curtain(vrut.CLOSE)
        anseva.curtain(vrut.CLOSE)
        anscont.curtain(vrut.CLOSE)
        ansclose.curtain(vrut.CLOSE)
        ansstore.curtain(vrut.CLOSE)
        anscentr.curtain(vrut.CLOSE)
        ans5noknow.curtain(vrut.CLOSE)
    elif flag == 'q6flag':
        ans2health.curtain(vrut.CLOSE)
        ans2hab.curtain(vrut.CLOSE)
        ans2exp.curtain(vrut.CLOSE)
        ans2eva.curtain(vrut.CLOSE)
        ans2cont.curtain(vrut.CLOSE)
        ans2close.curtain(vrut.CLOSE)
        ans2store.curtain(vrut.CLOSE)
        ans2centr.curtain(vrut.CLOSE)
        ans6noknow.curtain(vrut.CLOSE)
    elif flag == 'q7flag':
        ans3health.curtain(vrut.CLOSE)
        ans3hab.curtain(vrut.CLOSE)
        ans3exp.curtain(vrut.CLOSE)
        ans3eva.curtain(vrut.CLOSE)
        ans3cont.curtain(vrut.CLOSE)
        ans3close.curtain(vrut.CLOSE)
        ans3store.curtain(vrut.CLOSE)
        ans3centr.curtain(vrut.CLOSE)
        ans7noknow.curtain(vrut.CLOSE)
    elif flag == 'q8flag':
        ans4health.curtain(vrut.CLOSE)
        ans4hab.curtain(vrut.CLOSE)
        ans4exp.curtain(vrut.CLOSE)
        ans4eva.curtain(vrut.CLOSE)
        ans4cont.curtain(vrut.CLOSE)
        ans4close.curtain(vrut.CLOSE)
        ans4store.curtain(vrut.CLOSE)
        ans4centr.curtain(vrut.CLOSE)
        ans8noknow.curtain(vrut.CLOSE)

def scrollAns(flag,which):
    if flag == 'q2flag':
        print(which)
        closeOthers('q2flag')
        if which == 0:
            ansblue.curtain(vrut.OPEN)
        elif which == 1:
            ansyellw.curtain(vrut.OPEN)
        else:
            print('error in q2flag')
    elif flag == 'q1flag':
        closeOthers('q1flag')
        if which == 0:
            ansupright.curtain(vrut.OPEN)
        elif which == 1:
            ansrdown.curtain(vrut.OPEN)
        elif which == 2:
            ansupdown.curtain(vrut.OPEN)
        elif which == 3:
            ansldown.curtain(vrut.OPEN)
        elif which == 4:
            ansnoknow.curtain(vrut.OPEN)
        else:
            print('error in q1flag')
    elif flag == 'q3flag':
        closeOthers('q3flag')
        if which == 0:
            ans3pit90f.curtain(vrut.OPEN)
        elif which == 1:
            ans3pit90b.curtain(vrut.OPEN)
        elif which == 2:
            ans3yaw180.curtain(vrut.OPEN)
        elif which == 3:
            ans3yaw90l.curtain(vrut.OPEN)
        elif which == 4:
            ans3yaw90r.curtain(vrut.OPEN)
        elif which == 5:
            ans3norot.curtain(vrut.OPEN)
        elif which == 6:
            ans3noknow.curtain(vrut.OPEN)
        else:
            print('error in q3flag')
    elif flag == 'q4flag':
        closeOthers('q4flag')
        if which == 0:
            ansr0.curtain(vrut.OPEN)
        elif which == 1:
            ansr90l.curtain(vrut.OPEN)
        elif which == 2:
            ansr90r.curtain(vrut.OPEN)
        elif which == 3:
            ansr180.curtain(vrut.OPEN)
        elif which == 4:
            ans4noknow.curtain(vrut.OPEN)
        else:
            print('error in q4flag')
    elif flag == 'q5flag':
        closeOthers('q5flag')
        if which == 0:
            ansexp.curtain(vrut.OPEN)
        elif which == 1:
            anseva.curtain(vrut.OPEN)
        elif which == 2:
            ansstore.curtain(vrut.OPEN)
        elif which == 3:
            anshealth.curtain(vrut.OPEN)
        elif which == 4:
            anshab.curtain(vrut.OPEN)
        elif which == 5:
            anscentr.curtain(vrut.OPEN)
        elif which == 6:
            anscont.curtain(vrut.OPEN)
        elif which == 7:
            ansclose.curtain(vrut.OPEN)
        elif which == 8:
            ans5noknow.curtain(vrut.OPEN)
        else:
            print('error in q5flag')
    elif flag == 'q6flag':
        closeOthers('q6flag')
        print('which', which)
        if which == 0:
            ans2exp.curtain(vrut.OPEN)
        elif which == 1:
            ans2eva.curtain(vrut.OPEN)
        elif which == 2:
            ans2store.curtain(vrut.OPEN)
        elif which == 3:
            ans2health.curtain(vrut.OPEN)
        elif which == 4:
            ans2hab.curtain(vrut.OPEN)
        elif which == 5:
            ans2centr.curtain(vrut.OPEN)
        elif which == 6:
            ans2cont.curtain(vrut.OPEN)
        elif which == 7:
            ans2close.curtain(vrut.OPEN)
        elif which == 8:
            ans6noknow.curtain(vrut.OPEN)
    elif flag == 'q7flag':
        closeOthers('q7flag')
        if which == 0:
            ans3exp.curtain(vrut.OPEN)
        elif which == 1:
            ans3eva.curtain(vrut.OPEN)
        elif which == 2:
            ans3store.curtain(vrut.OPEN)
        elif which == 3:
            ans3health.curtain(vrut.OPEN)
        elif which == 4:
            ans3hab.curtain(vrut.OPEN)
        elif which == 5:
            ans3centr.curtain(vrut.OPEN)
        elif which == 6:
            ans3cont.curtain(vrut.OPEN)
        elif which == 7:
            ans3close.curtain(vrut.OPEN)
        elif which == 8:
            ans7noknow.curtain(vrut.OPEN)
        else:
            print('error in q7flag')
    elif flag == 'q8flag':
        closeOthers('q8flag')
        if which == 0:
            ans4exp.curtain(vrut.OPEN)
        elif which == 1:
            ans4eva.curtain(vrut.OPEN)
        elif which == 2:
            ans4store.curtain(vrut.OPEN)
        elif which == 3:
            ans4health.curtain(vrut.OPEN)
        elif which == 4:
            ans4hab.curtain(vrut.OPEN)
        elif which == 5:
            ans4centr.curtain(vrut.OPEN)
    elif which == 6:
        ans4cont.curtain(vrut.OPEN)
    elif which == 7:
        ans4close.curtain(vrut.OPEN)
    elif which == 8:
        ans8noknow.curtain(vrut.OPEN)
    else:
        print('error in q8flag')


def closeEverything():
    global set1flag
    firstQuestion.curtain(vrut.CLOSE)
    secondQuestion.curtain(vrut.CLOSE)
    line.curtain(vrut.CLOSE)
    if set1flag == on:
        ansblue.curtain(vrut.CLOSE)
        ansyellw.curtain(vrut.CLOSE)
        ansupright.curtain(vrut.CLOSE)
        ansrdown.curtain(vrut.CLOSE)
        ansupdown.curtain(vrut.CLOSE)
        ansldown.curtain(vrut.CLOSE)
        ansnoknow.curtain(vrut.CLOSE)
        ans3pit90f.curtain(vrut.CLOSE)
        ans3pit90b.curtain(vrut.CLOSE)
        ans3yaw180.curtain(vrut.CLOSE)
        ans3yaw90l.curtain(vrut.CLOSE)
        ans3yaw90r.curtain(vrut.CLOSE)
        ans3norot.curtain(vrut.CLOSE)
        ans3noknow.curtain(vrut.CLOSE)
        ansr180.curtain(vrut.CLOSE)
        ansr90l.curtain(vrut.CLOSE)
        ansr90r.curtain(vrut.CLOSE)
        ansr0.curtain(vrut.CLOSE)
        ans4noknow.curtain(vrut.CLOSE)
    elif set1flag == off:
        anshealth.curtain(vrut.CLOSE)
        anshab.curtain(vrut.CLOSE)
        ansexp.curtain(vrut.CLOSE)
        anseva.curtain(vrut.CLOSE)
        anscont.curtain(vrut.CLOSE)
        ansclose.curtain(vrut.CLOSE)
        ansstore.curtain(vrut.CLOSE)
        anscentr.curtain(vrut.CLOSE)
        ans5noknow.curtain(vrut.CLOSE)
        ans2health.curtain(vrut.CLOSE)
        ans2hab.curtain(vrut.CLOSE)
        ans2exp.curtain(vrut.CLOSE)
        ans2eva.curtain(vrut.CLOSE)
        ans2cont.curtain(vrut.CLOSE)
        ans2close.curtain(vrut.CLOSE)
        ans2store.curtain(vrut.CLOSE)
        ans2centr.curtain(vrut.CLOSE)
        ans6noknow.curtain(vrut.CLOSE)
        ans3health.curtain(vrut.CLOSE)
        ans3hab.curtain(vrut.CLOSE)
        ans3exp.curtain(vrut.CLOSE)
        ans3eva.curtain(vrut.CLOSE)
        ans3cont.curtain(vrut.CLOSE)
        ans3close.curtain(vrut.CLOSE)
        ans3store.curtain(vrut.CLOSE)
        ans3centr.curtain(vrut.CLOSE)
        ans7noknow.curtain(vrut.CLOSE)
        ans4health.curtain(vrut.CLOSE)
        ans4hab.curtain(vrut.CLOSE)
        ans4exp.curtain(vrut.CLOSE)
        ans4eva.curtain(vrut.CLOSE)
        ans4cont.curtain(vrut.CLOSE)
        ans4close.curtain(vrut.CLOSE)
        ans4store.curtain(vrut.CLOSE)
        ans4centr.curtain(vrut.CLOSE)
        ans8noknow.curtain(vrut.CLOSE)


def detWhat():
    global routeflag,module,count,moremodflag, nextflag
    print('in detWhat')
    if routeflag == off:
        return module
    else:
        routeflag = off
        if moremodflag == on:
            moremodflag = off
            return other[count][0]
        else:
            moremodflag = on
            nextflag = on
            return other[count][1]


def myKeyboard(key):
    global begintime
    if key == ' ':
        reminder.curtain(vrut.CLOSE)
        vrut.starttimer(layout,0.001)
        begintime = time.time()
        out = str(time.time()-begintime) + '\t' + str(count) +
'\tcount\n'
        keys.write(out)
        keys.flush()
    elif key == 'r':
        tracker.command(6)  # RESET1    ... see multi.py in AndyL
        tracker.reset()
    elif key == '1':
        showTarget(module,.75)

def mytimer(timernum):
    global module, count, routeflag, showing, set1flag,moremodflag,
nextflag,flyflag,model_flag
    global maxx,maxy,maxz, x,y,z, stepx,stepy,stepz, yaw,pitch,roll,
turny,turnp,turnr, startpoint
    global q1flag,q1index,q2flag,q2index,q3flag,q3index,q4flag,q4index
    global q5flag,q5index,q6flag,q6index,q7flag,q7index,q8flag,q8index
    global                                          dispx,dispy,dispz,
movez,movey,movex,boundx,boundy,boundz,
R,data,prevdata,first,begintime
    if timernum == layout:
        # initialize variables
        initVariables()
        # show target; close
        # show astronaut or pop up SIM
        if routeflag == on:
            showTarget(module,2)
            out    =    str(initpost[module])+'\t'+str(module)+'\t'+str(-
1)+'\t'+str(-1)+'\t'+str(-1)+'\t'+str(-1)+'\t'+str(-1)+'\t'
            file.write(out)
            file.flush()
            vrut.starttimer(placeAstrnt,2.02)
        else:
            if count == 0 or count == 1 or count == 2:
                vrut.starttimer(next,0.001)
            elif count == 3 or count == 4 or count == 5:
                startpoint = time.time()
                vrut.starttimer(taskpoint,2.02)
            else: print('there is an error in layout')
        # first questions
        # travel to the astronaut
        # second question
        # play sound/back to Control
        # show targe; close
        # pointing task
        # first question
        # second question
        # determine what are the two other targets
        # show target of other target 1; close
        # pointing task for other target 1
        # first question
        # second question
        # show target of other target 2; close
        # point
        # first question
        # second question

    elif timernum == closeTarget:
        closeAll()

    elif timernum == placeAstrnt:
        # should only happen once every trial set
        print('placing astronaut: count',count)
        angles = defAngles(segori[module][0],segori[module][1])
        quat = Eul2Quats(angles[0],angles[1],angles[2])
        if initpost[module] == 1:
            learn = Eul2Quats(0,0,90)
            print('is astronaut right shoulder down?!')
        else:
            learn = Eul2Quats(0,0,0)
        model.avatar.translate(0,0,0)
        model.avatar.rotate(learn[0],learn[1],learn[2],learn[3])

model.avatar2.translate(geoxyz[module][0],geoxyz[module][1],geoxyz[
module][2])
        model.avatar2.rotate(quat[0],quat[1],quat[2],quat[3])
        startpoint = time.time()
        vrut.starttimer(waitRATCHET,0.1)
        if count == 0 or count == 1 or count == 2:
            vrut.starttimer(showFirstQuest,2)

    elif timernum == waitRATCHET:
        if sid.buttons() == sid.BUTTON8 and model_flag == off:
            model.model.curtain(vrut.OPEN)
            tracker.command(RESET2)
            tracker.reset()
            model_flag = on
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON8)
+ '\tratchet_on\n'
            keys.write(out)
            keys.flush()
            print('within ratchet, model_flag = on')
            vrut.starttimer(move_model,0.001)
        else:
            vrut.starttimer(waitRATCHET,0.3)

    elif timernum == move_model:
        if model_flag == on and sid.buttons() == 0:
            #print('moving')
            data = wand.get()
            model.model.rotate(-data[3],-
data[4],data[5],data[6]*RAD2DEG)
            model.model.translate(offsetx+data[0],         offsety+data[1],
offsetz+data[2])
            prevdata = [data[0],data[1],data[2]]
            vrut.starttimer(move_model,0.001)
        elif model_flag == on:
            if  sid.buttons()  ==  sid.BUTTON7  or  sid.buttons()  ==
sid.BUTTON8:
                print('pressed flyin or ratchet, model_flag off')
                model_flag = off
                if sid.buttons() == sid.BUTTON7 and flyflag == off:
                    model.model.curtain(vrut.CLOSE)
                    tracker.command(STORE_REFFRAME)
                    if first == on:
                        first = off
                        tracker.command(USE_REFFRAME)
                    vrut.starttimer(waitRATCHET,rate)
                    out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON7) + '\thide_model_ratchet_off\n'
                    keys.write(out)
                    keys.flush()
#                   model_flag = on
#                   print('have not answered questions, model_flag on')
#                   vrut.starttimer(move_model,0.001)
                elif sid.buttons() == sid.BUTTON7 and flyflag == on:
                    #  Rotate  position  vector  of  avatar  by  rotation  of
model/sensor
                    # In data quaternions
                    a1 = -data[3]      # 4/27/2001 negative included due to
new multi sensor
                    a2 = -data[4]      # 4/27/2001 negative included due to
new multi sensor
                    a3 = data[5]
                    ang = -data[6]          # which is in radians
                    A11  =  math.cos(ang);  A22  =  math.cos(ang);  A33  =
math.cos(ang)
                    A12 = 0; A13 = 0; A21 = 0; A23 = 0; A31 = 0; A32 = 0
                    K = 1 - math.cos(ang)
                    B11 = K*a1*a1
                    B12 = K*a1*a2
                    B13 = K*a1*a3
                    B21 = K*a2*a1
                    B22 = K*a2*a2
                    B23 = K*a2*a3
                    B31 = K*a3*a1
                    B32 = K*a3*a2
                    B33 = K*a3*a3
                    S = -math.sin(ang)
                    C11 = 0
                    C12 = -S*a3
                    C13 = S*a2
                    C21 = S*a3
                    C22 = 0
                    C23 = -S*a1
                    C31 = -S*a2
                    C32 = S*a1
                    C33 = 0
                    R1 = [A11+B11+C11,A12+B12+C12,A13+B13+C13]
                    R2 = [A21+B21+C21,A22+B22+C22,A23+B23+C23]
                    R3 = [A31+B31+C31,A32+B32+C32,A33+B33+C33]
                    R = [R1, R2, R3]   # TADA! :)
                    dispx = prevdata[0]
                    dispy = prevdata[1]
                    dispz = prevdata[2]
                    prevdata = [0,0,0]
                    vrut.starttimer(flyin,0.001)
                    out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON7) + '\tflyin_ratchet_off\n'
                    keys.write(out)
                    keys.flush()
                elif sid.buttons() == sid.BUTTON8:
                    print('store refframe button8')
                    tracker.command(STORE_REFFRAME)
                    if first == on:
                        first = off
                        tracker.command(USE_REFFRAME)
                    vrut.starttimer(waitRATCHET,rate)
                    out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON8) + '\tratchet_off\n'
```

```
                keys.write(out)
                keys.flush()
            else:
                vrut.starttimer(move_model,0.001)
                out = str(time.time()-begintime) + '\t' + str(sid.buttons())
+ '\tincorrect_button\n'
                keys.write(out)
                keys.flush()

    elif timernum == flyin:
        movez = 0
        movey = 0
        movex = 0
        boundx = off
        boundy = off
        boundz = off
        oldx = geoxyz[module][0]*modscfac
        oldy = geoxyz[module][1]*modscfac
        oldz = geoxyz[module][2]*modscfac
        avaposx = R[0][0]*oldx + R[0][1]*oldy + R[0][2]*oldz
        avaposy = R[1][0]*oldx + R[1][1]*oldy + R[1][2]*oldz
        avaposz = R[2][0]*oldx + R[2][1]*oldy + R[2][2]*oldz
        dispx = offsetx + data[0]
        dispy = offsety + data[1]
        dispz = offsetz + data[2]
        maxx = -(dispx + avaposx)
        maxy = -(dispy + avaposy)
        maxz = -(dispz + avaposz)
        stepx = maxx/flytime
        stepy = maxy/flytime
        stepz = maxz/flytime
        tracker.command(STORE_REFFRAME)
        if first == on:
            first = off
            tracker.command(USE_REFFRAME)
        station.curtain(vrut.CLOSE)
        vrut.starttimer(flyin2,0.001)

    elif timernum == flyin2:
        if abs(movex) > abs(maxx): boundx = on
        if abs(movey) > abs(maxy): boundy = on
        if abs(movez) > abs(maxz): boundz = on
        if boundx == off or boundy == off or boundz == off:
            if boundx == off:
                movex = movex + stepx
            if boundy == off:
                movey = movey + stepy
            if boundz == off:
                movez = movez + stepz

model.model.translate(dispx+movex,dispy+movey,dispz+movez)
            vrut.starttimer(flyin2,0.001)
        else:
            vrut.reset(vrut.HEAD_POS)

vrut.translate(vrut.HEAD_POS,geoxyz[module][0],geoxyz[module][1
],geoxyz[module][2])
            vrut.reset(vrut.BODY_ORI)
            angles = defAngles(segori[module][0],segori[module][1])
            vrut.rotate(vrut.BODY_ORI,angles[0],-angles[1],angles[2])
            print('are the body rotations correct???')
            model.model.curtain(vrut.CLOSE)
            station.curtain(vrut.OPEN)
            set1flag = off
            line.translate(0,lay,lz)
            if count == 0 or count == 1 or count == 2:
                vrut.starttimer(showSecondQuest,2)
            elif count == 3 or count == 4 or count == 5:
                vrut.starttimer(next, 7)
            else:
                print('there is something wrong at the end of flyin')

    elif timernum == next:
        routeflag = on
        set1flag = on
        print('increasing count')
        if count == maxtrials:
            print('endprogram!!!')
            vrut.translate(vrut.HEAD_POS, 0,100,0)
            file.close()
            keys.close()
        else:
            count = count + 1
            vrut.starttimer(layout,0.001)

    elif timernum == waitPOP:
        if sid.buttons() == sid.BUTTON10:
            if set1flag == on:
                vrut.starttimer(showFirstQuest,0.001)
            else:
                vrut.starttimer(showSecondQuest,0.001)
        else:
            vrut.starttimer(waitPOP,rate)

    elif timernum == showFirstQuest:
        if showing == on:
            model.model.curtain(vrut.OPEN)
            showing = off
            closeEverything()
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\thide_quest_set1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(waitPOP,rate)
        elif showing == off:
            showing = on
            model.model.curtain(vrut.CLOSE)
            firstQuestion.curtain(vrut.OPEN)
            line.curtain(vrut.OPEN)
            scrollAns('q1flag',q1index)
            scrollAns('q2flag',q2index)
            scrollAns('q3flag',q3index)
            scrollAns('q4flag',q4index)
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\tshow_quest_set1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)

    elif timernum == showSecondQuest:
        if showing == on:
            showing = off
            closeEverything()
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\thide_quest_set2\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(waitPOP,rate)
        elif showing == off:
            showing = on
            secondQuestion.curtain(vrut.OPEN)
            line.curtain(vrut.OPEN)
            print('2nd question',q5index,q6index,q7index,q8index)
            scrollAns('q5flag',q5index)
            scrollAns('q6flag',q6index)
            scrollAns('q7flag',q7index)
            scrollAns('q8flag',q8index)
        out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\tshow_quest_set2\n'
        keys.write(out)
        keys.flush()
        vrut.starttimer(scroller,rate)

    elif timernum == scroller:
      if sid.buttons() == sid.BUTTON10 and routeflag == on:
        if set1flag == on:
          vrut.starttimer(showFirstQuest,0.001)
        else:
          vrut.starttimer(showSecondQuest,0.001)
        # watch out for a possible BUG HERE!
      elif sid.buttons() == sid.BUTTON1:
        # check for which question
        # scroll up (add)
        if q2flag == on:
          if q2index == 0:
            q2index = 1
          else:
            q2index = 0
          scrollAns('q2flag',q2index)
        elif q1flag == on:
          if q1index == 4:
            q1index = 0
          else:
            q1index = q1index + 1
          scrollAns('q1flag',q1index)
        elif q3flag == on:
          if q3index == 6:
            q3index = 0
          else:
            q3index = q3index + 1
          scrollAns('q3flag',q3index)
        elif q4flag == on:
          if q4index == 4:
            q4index = 0
          else:
            q4index = q4index + 1
          scrollAns('q4flag',q4index)
        elif q5flag == on:
          if q5index == 8:
            q5index = 0
          else:
            q5index = q5index + 1
          scrollAns('q5flag',q5index)
        elif q6flag == on:
          if q6index == 8:
            q6index = 0
          else:
            q6index = q6index + 1
          scrollAns('q6flag',q6index)
        elif q7flag == on:
          if q7index == 8:
            q7index = 0
          else:
            q7index = q7index + 1
          scrollAns('q7flag',q7index)
        elif q8flag == on:
          if q8index == 8:
            q8index = 0
          else:
            q8index = q8index + 1
          scrollAns('q8flag',q8index)
        vrut.starttimer(scroller,rate)
        out = str(time.time()-begintime) + '\t' + str(sid.BUTTON1)
+ '\tscroll_up\n'
        keys.write(out)
        keys.flush()

      elif sid.buttons() == sid.BUTTON4:
        # check for which question
        # scroll down (subtract)
        if q2flag == on:
          if q2index == 1:
            q2index = 0
          else:
            q2index = 1
          scrollAns('q2flag',q2index)
        elif q1flag == on:
          if q1index == 0:
            q1index = 4
          else:
            q1index = q1index - 1
          scrollAns('q1flag',q1index)
        elif q3flag == on:
          if q3index == 0:
            q3index = 6
          else:
            q3index = q3index - 1
          scrollAns('q3flag',q3index)
        elif q4flag == on:
          if q4index == 0:
            q4index = 4
          else:
            q4index = q4index - 1
          scrollAns('q4flag',q4index)
        elif q5flag == on:
          if q5index == 0:
            q5index = 8
          else:
            q5index = q5index - 1
          scrollAns('q5flag',q5index)
        elif q6flag == on:
          if q6index == 0:
            q6index = 8
          else:
            q6index = q6index - 1
          scrollAns('q6flag',q6index)
        elif q7flag == on:
          if q7index == 0:
            q7index = 8
          else:
            q7index = q7index - 1
          scrollAns('q7flag',q7index)
        elif q8flag == on:
          if q8index == 0:
            q8index = 8
          else:
            q8index = q8index - 1
          scrollAns('q8flag',q8index)
        vrut.starttimer(scroller,rate)
        out = str(time.time()-begintime) + '\t' + str(sid.BUTTON4)
+ '\tscroll_down\n'
        keys.write(out)
        keys.flush()

      elif sid.buttons() == 768:   # pressing sid.BUTTON9 and
sid.BUTTON10 together
        elapstime = time.time() - startpoint
        startpoint = time.time()
        closeEverything()
        showing = off
        out = str(time.time()-begintime) + '\t' + str(768) +
'\tenter_hide_quest_set\n'
        keys.write(out)
        keys.flush()
        if set1flag == on:
            q1flag = off
            q2flag = off
            q3flag = off
            q4flag = off
            q5flag = on
            out                                                                      =
str(q1index)+'\t'+str(q2index)+'\t'+str(q3index)+'\t'+str(q4index)+'
\t'+ str(elapstime)+'\t'
            file.write(out)
            file.flush()
            if routeflag == on:
               flyflag = on
               model.model.curtain(vrut.OPEN)
               #routeflag = off
               # NO waiting for flyin here! completely different
               # from control.  Just follow model/ratchet program
               #vrut.starttimer(waitFLYIN,rate)
               vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\EndQuest.wav')
               print('not going to FLYIN')
            else:
               print('repeating??')
               #print(set1flag, routeflag)
               set1flag = off
               line.translate(0,lay,lz)
               startpoint = time.time()
               vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\EndQuest.wav')
               vrut.starttimer(showSecondQuest,0.1)
         else:
            q5flag = off
            q6flag = off
            q7flag = off
            q8flag = off
            q1flag = on
            line.translate(0,lay,lz)
            out                                                                      =
str(q5index)+'\t'+str(q6index)+'\t'+str(q7index)+'\t'+str(q8index)+'
\t'+ str(elapstime)+'\n'
            file.write(out)
            file.flush()
            if routeflag == on:
               routeflag = off
               print('here')
               vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\Reminder.wav')
               vrut.starttimer(layout,2)
            else:
               routeflag = on
               set1flag = on
               if nextflag == on:
                  print('increasing count')
                  if count == maxtrials:
                     print('endprogram!!!')
                     vrut.translate(vrut.HEAD_POS, 0,100,0)
                     file.close()
                     keys.close()
                  else:
                     count = count + 1
                     vrut.starttimer(layout,0.001)
               else:
                  startpoint = time.time()
                  vrut.starttimer(taskpoint,1)
               #vrut.starttimer(taskpoint,0.1)

      elif sid.buttons() == sid.BUTTON9:
         # enter answer  (record)
         # increase which question counter
         if q1flag == on:
            q1flag = off
            q2flag = on
            line.translate(0,lby,lz)
            scrollAns('q2flag',q2index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
         elif q2flag == on:
            q2flag = off
            q3flag = on
            line.translate(0,lcy,lz)
            scrollAns('q3flag',q3index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question2\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
         elif q3flag == on:
            q3flag = off
            q4flag = on
            line.translate(0,ldy,lz)
            scrollAns('q4flag',q4index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question3\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
         elif q4flag == on:
            q4flag = off
            q1flag = on
            line.translate(0,lay,lz)
            scrollAns('q1flag',q1index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question4\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
         elif q5flag == on:
            q5flag = off
            q6flag = on
            line.translate(0,lby,lz)
            scrollAns('q6flag',q6index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question5\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
         elif q6flag == on:
            q6flag = off
            q7flag = on
            line.translate(0,lcy,lz)
            scrollAns('q7flag',q7index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question6\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
         elif q7flag == on:
            q7flag = off
            q8flag = on
            line.translate(0,ldy,lz)
            scrollAns('q8flag',q8index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question7\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
```

```
        elif q8flag == on:
            q8flag = off
            q5flag = on
            line.translate(0,lay,lz)
            scrollAns('q5flag',q5index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question8\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)

    else:
        vrut.starttimer(scroller,rate)

  elif timernum == taskpoint:
    what = detWhat()
    print('what',what)
    showTarget(what,2)
    out = str(initpost[count])+'\t'+str(what)+'\t'
    file.write(out)
    file.flush()
    arrow.curtain(vrut.OPEN)
    #startpoint = time.time()
    vrut.starttimer(waitENTER,0.001)

  elif timernum == waitENTER:
    if sid.buttons() == sid.BUTTON9:
        arrow.curtain(vrut.CLOSE)
        data = tracker.get()
        elapstime = time.time() - startpoint
        out                                                              =
str(data[3])+'\t'+str(data[4])+'\t'+str(data[5])+'\t'+str(data[6]*RAD2
DEG)+'\t'+str(elapstime)+'\t'
        file.write(out)
        file.flush()
        startpoint = time.time()
        out = str(time.time()-begintime) + '\t' + str(sid.BUTTON9)
+ '\tenter_pointing\n'
        keys.write(out)
        keys.flush()
        vrut.starttimer(showFirstQuest,1)
    else:
        vrut.starttimer(waitENTER,rate)



#
*****************************************************************
**********************

# Setting up output files
exp_time = time.localtime(time.time())
timestring = time.strftime('%b%d_%H%M',exp_time)
filename          =                    'N:\Jessica\Thesis
SIM\Data\Sim\Phase2\Phase2a_SIM'+str(timestring)+'.dat'
file = open(filename,'w')
out                                                              =
'Posture\tTarget\tQuat0\tQuat1\tQuat2\tQuat3\tTime\tQ1\tQ2\
tQ3\tQ4\tTime1\tQ5\tQ6\tQ7\tQ8\tTime2\n'
file.write(out)
file.flush()
# second file out
filename2          =                    'N:\Jessica\Thesis
SIM\Data\Sim\Phase2\Phase2a_SIM'+str(timestring)+'_allkeys.dat'
keys = open(filename2,'w')


vrut.callback(vrut.KEYBOARD_EVENT,'myKeyboard')

vrut.callback(vrut.TIMER_EVENT,'mytimer')

reminder.curtain(vrut.OPEN)
```

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

(Part b)

```
import vrut
import time
import math
import types
import string
import sid

# 6/4/2001  This is adapting trainingCntrl_ver3 to the new protocol.
# 6/6/2001  After finishing phase 2 training ver 4 for control,
adapting this for SIM
# 6/7/2001  done.
# 6/23/2001 protocol is too long. dividing phase 2 into two parts.
this isn't exactly
#         necessarily like partA...
# 7/2/2001  Need to edit to have second set of questions only
appear for Route, not for
#         other targets.

#
*****************************************************************
**********************
# KEY:
#
*****************************************************************
**********************

#    1                          direction nomenclature
#    |
# 2 ----- 0 into the page --   4
#    |                 out of the page -- 5
#    3
#
# Target difficulty:
#          0: modules 0 or 1
#          1: modules 4 or 5
#          2: modules 2 or 3
# FarNode:          I    MIGHT    HAVE    GOTTEN    THIS
BACKWARDS... 4/2/2001
#     0: modules 1, 3, or 5
#          1: modules 0, 2, or 4
# Posture:  relative to module 6
#          0: roll, with nose at 4 & feet at 1
#          1: pitch, (positive) with nose at 1 & feet at 4
#          2: roll, with nose at 4 & feet at 2
#          3: pitch, (negative) with nose at 3 & feet at 5
#
# Module labelling:
#          0: Columbus, European Module       == Experiment
#          1: Stowage Module                  == EVA
#          2: Zarya, Russian Control Module   == Storage
#          3: Destiny, U.S. Module            == Health Fitness
#          4: Habitation Module               == Habitation
#          5: Kibo, Japanese Module           == Centrifuge
#          6: Zvezda, Russian Service Module  == Control
#          Nodes
#          7: FarNode 1, hatch at end of zvezda (blue)         #
6/4/2001  had these reversed
#          8: FarNode 0, hatch at other end of zvezda (yellow)
#
```

```
#                                                              Constants
*****************************************************************
************

on = 1
off = 0
flytime = 100
speed = .25
flytime = 85
maxtrials = 5   # double check!
rate = 0.3
modscfac = 0.075

qax = .3
qay = .435
qby = .15
qcy = -.15
qdy = -.435
qaz = 2.05
lay = .275
lby = 0
lcy =-.275
ldy = -.57
lz  = 2.0
ansscfac = 4
scfac = 2.25

HMD = on

RAD2DEG = 57.295779513082323
DEG2RAD = 1/57.295779513082323

#
*****************************************************************
**********************

if HMD == on:
   vrut.go(vrut.HMD | vrut.STEREO)
else:
   vrut.go(vrut.CONSOLE)
   vrut.translate(vrut.HEAD_POS, 0,-1.82,0)

#
*****************************************************************
**********************
#              Sensor              &              Commands
*****************************************************************
****

MODE_EULER = 1
MODE_QUAT = 2
MODE_QUAT_NORM = 3
USE_REFFRAME = 4
STORE_REFFRAME = 5
RESET1 = 6
RESET2 = 7
RESET3 = 8
RESET4 = 9
RATCHET1 = 10
RATCHET2 = 11
RATCHET3 = 12
RATCHET4 = 13

# head tracker : must do this one first
tracker = vrut.addsensor('isensemulti_beta2')
vrut.tracker()
tracker.command(RESET1)  # RESET1
tracker.reset()

# wand sensor
wand = vrut.addsensor('isensemulti_beta2')
#tracker.command(MODE_QUAT_NORM)
#tracker.command(RATCHET2)
tracker.command(RESET2)
tracker.reset()

#
*****************************************************************
**********************
# Loading objects and their initial positions

global dispx, dispy, dispz            # initial displacement of model
dispx = 0          # offset in the x
dispy = -.15       # offset in the y
dispz = 1.3        # offset in the z
offsetx = dispx
offsety = dispy
offsetz = dispz

class mod_ava(vrut.EventClass):
     def __init__(modava):  #  for now.  could  be  more,  see
SIM_ratchet...
          vrut.EventClass.__init__(modava)
          modava.model      =         vrut.addchild('n:\Jessica\Thesis
SIM\Models\station_astr_obj.wrl',vrut.HEAD)
          modava.avatar = modava.model.getchild('astronaut')
          modava.avatar2 = modava.model.getchild('astrnt')

model = mod_ava()
model.model.scale(modscfac,modscfac,modscfac)
model.model.translate(dispx,dispy,dispz)
model.model.curtain(vrut.CLOSE) # initially, model hidden

station = vrut.addchild('n:\Jessica\Thesis SIM\Models\Station2.wrl')
arrow            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Cross.wrl',vrut.HEAD)
arrow.translate(0,0,.75)
arrow.curtain(vrut.CLOSE)

firstQuestion          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Firstquestion.wrl',vrut.HEAD)
firstQuestion.translate(0,0,1.5)
firstQuestion.scale(scfac,scfac,scfac)
firstQuestion.curtain(vrut.CLOSE)
secondQuestion          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Secondquestion.wrl',vrut.HEAD)
secondQuestion.translate(0,0,1.5)
secondQuestion.scale(scfac,scfac,scfac)
secondQuestion.curtain(vrut.CLOSE)
line             =             vrut.addchild('n:\Jessica\Thesis
SIM\Models\Underline.wrl',vrut.HEAD)
line.translate(0,lay,lz)
line.scale(scfac,scfac,scfac)
line.curtain(vrut.CLOSE)

ansblue            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansbluehatch.wrl',vrut.HEAD)
ansyellw            =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyellowhatch.wrl',vrut.HEAD)
ansblue.translate(qax,qby,qaz)
ansblue.scale(ansscfac,ansscfac,ansscfac)
ansyellw.translate(qax,qby,qaz)
ansyellw.scale(ansscfac,ansscfac,ansscfac)
ansblue.curtain(vrut.CLOSE)
ansyellw.curtain(vrut.CLOSE)

ansupright          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansupright.wrl',vrut.HEAD)
```

```
ansrdown = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansrightdown.wrl',vrut.HEAD)
ansupdown = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansupsidedown.wrl',vrut.HEAD)
ansldown = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansleftdown.wrl',vrut.HEAD)
ansnoknow = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansupright.translate(qax,qay,qaz)
ansupright.scale(ansscfac,ansscfac,ansscfac)
ansrdown.translate(qax,qay,qaz)
ansrdown.scale(ansscfac,ansscfac,ansscfac)
ansupdown.translate(qax,qay,qaz)
ansupdown.scale(ansscfac,ansscfac,ansscfac)
ansldown.translate(qax,qay,qaz)
ansldown.scale(ansscfac,ansscfac,ansscfac)
ansnoknow.translate(qax,qay,qaz)
ansnoknow.scale(ansscfac,ansscfac,ansscfac)
ansupright.curtain(vrut.CLOSE)
ansrdown.curtain(vrut.CLOSE)
ansupdown.curtain(vrut.CLOSE)
ansldown.curtain(vrut.CLOSE)
ansnoknow.curtain(vrut.CLOSE)

ans3pit90f = vrut.addchild('n:\Jessica\Thesis SIM\Models\Anspitch90f.wrl',vrut.HEAD)
ans3pit90b = vrut.addchild('n:\Jessica\Thesis SIM\Models\Anspitch90b.wrl',vrut.HEAD)
ans3yaw180 = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansyaw180.wrl',vrut.HEAD)
ans3yaw90l = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansyaw90l.wrl',vrut.HEAD)
ans3yaw90r = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansyaw90r.wrl',vrut.HEAD)
ans3norot = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansnorotate.wrl',vrut.HEAD)
ans3noknow = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans3pit90f.translate(qax,qcy,qaz)
ans3pit90f.scale(ansscfac,ansscfac,ansscfac)
ans3pit90b.translate(qax,qcy,qaz)
ans3pit90b.scale(ansscfac,ansscfac,ansscfac)
ans3yaw180.translate(qax,qcy,qaz)
ans3yaw180.scale(ansscfac,ansscfac,ansscfac)
ans3yaw90l.translate(qax,qcy,qaz)
ans3yaw90l.scale(ansscfac,ansscfac,ansscfac)
ans3yaw90r.translate(qax,qcy,qaz)
ans3yaw90r.scale(ansscfac,ansscfac,ansscfac)
ans3norot.translate(qax,qcy,qaz)
ans3norot.scale(ansscfac,ansscfac,ansscfac)
ans3noknow.translate(qax,qcy,qaz)
ans3noknow.scale(ansscfac,ansscfac,ansscfac)
ans3pit90f.curtain(vrut.CLOSE)
ans3pit90b.curtain(vrut.CLOSE)
ans3yaw180.curtain(vrut.CLOSE)
ans3yaw90l.curtain(vrut.CLOSE)
ans3yaw90r.curtain(vrut.CLOSE)
ans3norot.curtain(vrut.CLOSE)
ans3noknow.curtain(vrut.CLOSE)

ansr180 = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansroll180.wrl',vrut.HEAD)
ansr90l = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansroll90l.wrl',vrut.HEAD)
ansr90r = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansroll90r.wrl',vrut.HEAD)
ansr0 = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansroll0.wrl',vrut.HEAD)
ans4noknow = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansr0.translate(qax,qdy,qaz)
ansr0.scale(ansscfac,ansscfac,ansscfac)
ansr90l.translate(qax,qdy,qaz)
ansr90l.scale(ansscfac,ansscfac,ansscfac)
ansr90r.translate(qax,qdy,qaz)
ansr90r.scale(ansscfac,ansscfac,ansscfac)
ansr180.translate(qax,qdy,qaz)
ansr180.scale(ansscfac,ansscfac,ansscfac)
ans4noknow.translate(qax,qdy,qaz)
ans4noknow.scale(ansscfac,ansscfac,ansscfac)
ansr180.curtain(vrut.CLOSE)
ansr90l.curtain(vrut.CLOSE)
ansr90r.curtain(vrut.CLOSE)
ansr0.curtain(vrut.CLOSE)
ans4noknow.curtain(vrut.CLOSE)

anshealth = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modhealth.wrl',vrut.HEAD)
anshab = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modhabitation.wrl',vrut.HEAD)
ansexp = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modexperiment.wrl',vrut.HEAD)
anseva = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modeva.wrl',vrut.HEAD)
anscont = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modcontrol.wrl',vrut.HEAD)
ansclose = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ansstore = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modstorage.wrl',vrut.HEAD)
anscentr = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans5noknow = vrut.addchild('n:\Jessica\Thesis SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansexp.translate(qax,qay,qaz)
ansexp.scale(ansscfac,ansscfac,ansscfac)
anseva.translate(qax,qay,qaz)
anseva.scale(ansscfac,ansscfac,ansscfac)
ansstore.translate(qax,qay,qaz)
ansstore.scale(ansscfac,ansscfac,ansscfac)
anshealth.translate(qax,qay,qaz)
anshealth.scale(ansscfac,ansscfac,ansscfac)
anshab.translate(qax,qay,qaz)
anshab.scale(ansscfac,ansscfac,ansscfac)
anscentr.translate(qax,qay,qaz)
anscentr.scale(ansscfac,ansscfac,ansscfac)
anscont.translate(qax,qay,qaz)
anscont.scale(ansscfac,ansscfac,ansscfac)
ansclose.translate(qax,qay,qaz)
ansclose.scale(ansscfac,ansscfac,ansscfac)
ans5noknow.translate(qax,qay,qaz)
ans5noknow.scale(ansscfac,ansscfac,ansscfac)
anshealth.curtain(vrut.CLOSE)
anshab.curtain(vrut.CLOSE)
ansexp.curtain(vrut.CLOSE)
anseva.curtain(vrut.CLOSE)
anscont.curtain(vrut.CLOSE)
ansclose.curtain(vrut.CLOSE)
ansstore.curtain(vrut.CLOSE)
anscentr.curtain(vrut.CLOSE)
ans5noknow.curtain(vrut.CLOSE)

ans2health = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modhealth.wrl',vrut.HEAD)
ans2hab = vrut.addchild('n:\Jessica\Thesis SIM\Models\Modhabitation.wrl',vrut.HEAD)
```

```
ans2exp        =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
ans2eva        =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
ans2cont       =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans2close      =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans2store      =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
ans2centr      =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans6noknow     =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans2exp.translate(qax,qby,qaz)
ans2exp.scale(ansscfac,ansscfac,ansscfac)
ans2eva.translate(qax,qby,qaz)
ans2eva.scale(ansscfac,ansscfac,ansscfac)
ans2store.translate(qax,qby,qaz)
ans2store.scale(ansscfac,ansscfac,ansscfac)
ans2health.translate(qax,qby,qaz)
ans2health.scale(ansscfac,ansscfac,ansscfac)
ans2hab.translate(qax,qby,qaz)
ans2hab.scale(ansscfac,ansscfac,ansscfac)
ans2centr.translate(qax,qby,qaz)
ans2centr.scale(ansscfac,ansscfac,ansscfac)
ans2cont.translate(qax,qby,qaz)
ans2cont.scale(ansscfac,ansscfac,ansscfac)
ans2close.translate(qax,qby,qaz)
ans2close.scale(ansscfac,ansscfac,ansscfac)
ans6noknow.translate(qax,qby,qaz)
ans6noknow.scale(ansscfac,ansscfac,ansscfac)
ans2health.curtain(vrut.CLOSE)
ans2hab.curtain(vrut.CLOSE)
ans2exp.curtain(vrut.CLOSE)
ans2eva.curtain(vrut.CLOSE)
ans2cont.curtain(vrut.CLOSE)
ans2close.curtain(vrut.CLOSE)
ans2store.curtain(vrut.CLOSE)
ans2centr.curtain(vrut.CLOSE)
ans6noknow.curtain(vrut.CLOSE)

ans3health     =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
ans3hab        =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans3exp        =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
ans3eva        =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
ans3cont       =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans3close      =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans3store      =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
ans3centr      =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans7noknow     =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans3exp.translate(qax,qcy,qaz)
ans3exp.scale(ansscfac,ansscfac,ansscfac)
ans3eva.translate(qax,qcy,qaz)
ans3eva.scale(ansscfac,ansscfac,ansscfac)
ans3store.translate(qax,qcy,qaz)
ans3store.scale(ansscfac,ansscfac,ansscfac)
ans3health.translate(qax,qcy,qaz)
ans3health.scale(ansscfac,ansscfac,ansscfac)
ans3hab.translate(qax,qcy,qaz)
ans3hab.scale(ansscfac,ansscfac,ansscfac)
ans3centr.translate(qax,qcy,qaz)
ans3centr.scale(ansscfac,ansscfac,ansscfac)
ans3cont.translate(qax,qcy,qaz)
ans3cont.scale(ansscfac,ansscfac,ansscfac)
ans3close.translate(qax,qcy,qaz)
ans3close.scale(ansscfac,ansscfac,ansscfac)
ans7noknow.translate(qax,qcy,qaz)
ans7noknow.scale(ansscfac,ansscfac,ansscfac)
ans3health.curtain(vrut.CLOSE)
ans3hab.curtain(vrut.CLOSE)
ans3exp.curtain(vrut.CLOSE)
ans3eva.curtain(vrut.CLOSE)
ans3cont.curtain(vrut.CLOSE)
ans3close.curtain(vrut.CLOSE)
ans3store.curtain(vrut.CLOSE)
ans3centr.curtain(vrut.CLOSE)
ans7noknow.curtain(vrut.CLOSE)

ans4health     =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
ans4hab        =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans4exp        =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
ans4eva        =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
ans4cont       =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans4close      =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans4store      =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
ans4centr      =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans8noknow     =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans4exp.translate(qax,qdy,qaz)
ans4exp.scale(ansscfac,ansscfac,ansscfac)
ans4eva.translate(qax,qdy,qaz)
ans4eva.scale(ansscfac,ansscfac,ansscfac)
ans4store.translate(qax,qdy,qaz)
ans4store.scale(ansscfac,ansscfac,ansscfac)
ans4health.translate(qax,qdy,qaz)
ans4health.scale(ansscfac,ansscfac,ansscfac)
ans4hab.translate(qax,qdy,qaz)
ans4hab.scale(ansscfac,ansscfac,ansscfac)
ans4centr.translate(qax,qdy,qaz)
ans4centr.scale(ansscfac,ansscfac,ansscfac)
ans4cont.translate(qax,qdy,qaz)
ans4cont.scale(ansscfac,ansscfac,ansscfac)
ans4close.translate(qax,qdy,qaz)
ans4close.scale(ansscfac,ansscfac,ansscfac)
ans8noknow.translate(qax,qdy,qaz)
ans8noknow.scale(ansscfac,ansscfac,ansscfac)
ans4health.curtain(vrut.CLOSE)
ans4hab.curtain(vrut.CLOSE)
ans4exp.curtain(vrut.CLOSE)
ans4eva.curtain(vrut.CLOSE)
ans4cont.curtain(vrut.CLOSE)
ans4close.curtain(vrut.CLOSE)
ans4store.curtain(vrut.CLOSE)
ans4centr.curtain(vrut.CLOSE)
ans8noknow.curtain(vrut.CLOSE)

columbus       =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Columbus.wrl',vrut.HEAD)
stowage        =        vrut.addchild('n:\Jessica\Thesis
SIM\Models\Stowage.wrl',vrut.HEAD)
```

```
zarya          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Zarya.wrl',vrut.HEAD)
destiny        =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Destiny.wrl',vrut.HEAD)
habitation     =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Habitation.wrl',vrut.HEAD)
kibo           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Kibo.wrl',vrut.HEAD)
reminder       =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Reminder.wrl', vrut.HEAD)
columbus.scale(0.25,.25,.25)
stowage.scale(0.25,.25,.25)
zarya.scale(0.25,.25,.25)
destiny.scale(0.25,.25,.25)
habitation.scale(0.25,.25,.25)
kibo.scale(0.25,.25,.25)
columbus.translate(0,-.1,.5)
stowage.translate(0,-.1,.5)
zarya.translate(0,-.1,.5)
destiny.translate(0,-.1,.5)
habitation.translate(0,-.1,.5)
kibo.translate(0,-.1,.5)
reminder.translate(0,-.4,2)
columbus.curtain(vrut.CLOSE)
stowage.curtain(vrut.CLOSE)
zarya.curtain(vrut.CLOSE)
destiny.curtain(vrut.CLOSE)
habitation.curtain(vrut.CLOSE)
kibo.curtain(vrut.CLOSE)
reminder.curtain(vrut.CLOSE)


#
*****************************************************************
*********************


# geographic location of each denoted point relative to center of
station
# geoxyz[endpoint] = x,y,z       endpoint can be module or at
nodes
# interesting note: you need to make this floating numbers!
# 6/4/2001 this is changed to just the nodes!
#geoxyz = [[-5.0,0.0,-5.25],[-5.0,0.0,5.25],[0.0,-5.0,-5.25],
#        [0.0,5.0,5.25],[5.0,0.0,-5.25],[5.0,0.0,5.25],
#        [0.0,0.0,0.0],[0.0,0.0,5.25],[0.0,0.0,-5.25]]

geoxyz = [[0.0,0.0,-5.25],[0.0,0.0,5.25],[0.0,0.0,-5.25],
        [0.0,0.0,5.25],[0.0,0.0,-5.25],[0.0,0.0,5.25],
        [0.0,0.0,0.0],[0.0,0.0,5.25],[0.0,0.0,-5.25]]

trials = [1,0,3,2,5,4]
initpost = [0,0,0,0,1,1]
# initpost[module] = 0 if in canonical, 1 if right shldr down; posture
you learn "route"

# segments[module] = endpoints passing necessary to get there from
center of station
#segments = [[8,0],[7,1],[8,2],[7,3],[8,4],[7,5]]
# segments is unnecessary because all you need is to do
geoxyz[module][2] !!!

# segori[module] = orientation (relative to reset body_ori)
# orientation given as (nose,feet)
segori = [[2,3],[2,3],[3,4],[1,4],[0,5],[0,4]]

# other[count] = [other module, other module] within that hatch
other = [[3,5],[2,4],[1,5],[0,4],[3,1],[2,0]]
```

```
global          nextflag,module,count,routeflag,          showing,
set1flag,moremodflag
global q1flag,q2flag,q3flag,q4flag,q5flag,q6flag,q7flag,q8flag
global
q1index,q2index,q3index,q4index,q5index,q6index,q7index,q8index
global data, prevdata, model_flag, flyflag, first, begintime
nextflag = off
module = 0
count = 2
routeflag = on
set1flag = on
showing = off
moremodflag = on
q1flag = off
q2flag = off
q3flag = off
q4flag = off
q5flag = off
q6flag = off
q7flag = off
q8flag = off
q1index = 0
q2index = 0
q3index = 0
q4index = 0
q5index = 0
q6index = 0
q7index = 0
q8index = 0
data = wand.get()
prevdata = [0,0,0]
model_flag = off
if count == 0 or count == 1 :
   flyflag = off
else:
   flyflag = on
first = on

# Timernums
layout = 1
closeTarget = 2
move_model = 3
placeAstrnt = 4
waitPOP = 5 #?
waitFLYIN = 6
waitENTER = 7
showFirstQuest = 8
showSecondQuest = 9
scroller = 10
flyin = 11
flyin2 = 12
waitRATCHET= 13
taskpoint = 14
next = 15



#
*****************************************************************
*********************


def initVariables():
   global  module,count,  routeflag,  showing,  q1flag,  set1flag,
nextflag,flyflag
   global
q1index,q2index,q3index,q4index,q5index,q6index,q7index,q8index
   q1index = 0
   q2index = 0
   q3index = 0
```

```
    q4index = 0
    q5index = 0
    q6index = 0
    q7index = 0
    q8index = 0
    module = trials[count]
    # increase count??? where is it?
    if nextflag == on:
        nextflag = off
        routeflag = on
    showing = off
    if count == 0 or count == 1 :
        flyflag = off
    else:
        flyflag = on
    q1flag = on
    line.translate(0,lay,lz)
    set1flag = on
    initpos = initpost[module]
    vrut.reset(vrut.BODY_ORI)
    vrut.reset(vrut.HEAD_POS)
    if initpos == 1:
        vrut.rotate(vrut.BODY_ORI, 0,0,90)

def showTarget(modulenum, tclose):
    if modulenum == 0:
        columbus.curtain(vrut.OPEN)
    elif modulenum == 1:
        stowage.curtain(vrut.OPEN)
    elif modulenum == 2:
        zarya.curtain(vrut.OPEN)
    elif modulenum == 3:
        destiny.curtain(vrut.OPEN)
    elif modulenum == 4:
        habitation.curtain(vrut.OPEN)
    elif modulenum == 5:
        kibo.curtain(vrut.OPEN)
    vrut.starttimer(closeTarget,tclose)

def closeAll():
    columbus.curtain(vrut.CLOSE)
    stowage.curtain(vrut.CLOSE)
    zarya.curtain(vrut.CLOSE)
    destiny.curtain(vrut.CLOSE)
    habitation.curtain(vrut.CLOSE)
    kibo.curtain(vrut.CLOSE)

def defAngles(nose,feet):
    # print(nose,feet)
    # these are defined as if starting from original orientation
    # 4/25/2001 editted nose 3 or 1
    print('nose,feet',nose,feet)
    if nose == 2 or nose == 0:
        pitch = 0
        if nose == 2: yaw = -90
        else: yaw = 90
        if feet == 4:
            if nose == 2:
                roll = -90
            else: roll = 90
        elif feet == 1:
            roll = 180
        elif feet == 5:
            if nose == 2:
                roll = 90
            else: roll = -90
        elif feet == 3:
            roll = 0
    if nose == 3 or nose == 1:
        if feet == 5:
            if nose == 3:
                pitch = -90
                yaw = 0
                roll = 0
            else:
                yaw = 180
                pitch = 90 # ?
                roll = 0
        elif feet == 0:
            if nose == 3:
                yaw = 90
                pitch = -90
                roll = 0
            else:
                yaw = -90
                pitch = 90
                roll = 0
        elif feet == 4:
            if nose == 3:
                yaw = 180
                pitch = -90
                roll = 0
            else:
                yaw = 0
                pitch = 90
                roll = 0
        elif feet == 2:
            if nose == 3:
                yaw = -90
                pitch = -90
                roll = 0
            else:
                yaw = 90
                pitch = 90
                roll = 0

    if nose == 4 or nose == 5:
        pitch = 0
        if nose == 4: yaw = 0
        else: yaw = 180
        if feet == 3:
            roll = 0
        elif feet == 1:
            roll = 180
        elif feet == 0:
            if nose == 4:
                roll = 90
            else: roll = -90
        elif feet == 2:
            if nose == 4:
                roll = -90
            else: roll = 90
    return [yaw,pitch,roll]

def Eul2Quats(psi,theta,phi):
    # psi == yaw; theta == pitch; phi == roll
    # Mode of implementation : model.avatar.rotate(-q2,q3,-
q1,2*math.acos(q0)*RAD2DEG)
    cr = math.cos(phi*DEG2RAD/2)
    cp = math.cos(theta*DEG2RAD/2)
    cy = math.cos(psi*DEG2RAD/2)
    sr = math.sin(phi*DEG2RAD/2)
    sp = math.sin(theta*DEG2RAD/2)
    sy = math.sin(psi*DEG2RAD/2)
    cpcy = cp*cy
    spsy = sp*sy
    q0 = cr*cpcy + sr*spsy
    q1 = sr*cpcy - cr*spsy
    q2 = cr*sp*cy + sr*cp*sy
    q3 = cr*cp*sy - sr*sp*cy
    quat0 = -q2
    quat1 = q3
    quat2 = -q1
    quat3 = 2*math.acos(q0)*RAD2DEG
    return [quat0,quat1,quat2,quat3]


def defLearnTurns(modulenum):
    if modulenum == 0:
        yaw = 90
        pitch = 0
        roll = 0
    elif modulenum == 1:
        yaw = -90
        pitch = 0
        roll = 0
    elif modulenum == 2:
        yaw = 0
        pitch = 90
        roll = 0
    elif modulenum == 3:
        yaw = 0
        pitch = 90
        roll = 0
    elif modulenum == 4:
        yaw = -90
        pitch = 0
        roll = 0
    elif modulenum == 5:
        yaw = 90
        pitch = 0
        roll = 0

    return [yaw,pitch,roll]

def closeOthers(flag):
    #global q1flag,q2flag,q3flag,q4flag,q5flag,q6flag,q7flag,q8flag
    if flag == 'q2flag':
        ansblue.curtain(vrut.CLOSE)
        ansyellw.curtain(vrut.CLOSE)
    elif flag == 'q1flag':
        ansupright.curtain(vrut.CLOSE)
        ansrdown.curtain(vrut.CLOSE)
        ansupdown.curtain(vrut.CLOSE)
        ansldown.curtain(vrut.CLOSE)
        ansnoknow.curtain(vrut.CLOSE)
    elif flag == 'q3flag':
        ans3pit90f.curtain(vrut.CLOSE)
        ans3pit90b.curtain(vrut.CLOSE)
        ans3yaw180.curtain(vrut.CLOSE)
        ans3yaw90l.curtain(vrut.CLOSE)
        ans3yaw90r.curtain(vrut.CLOSE)
        ans3norot.curtain(vrut.CLOSE)
        ans3noknow.curtain(vrut.CLOSE)
    elif flag == 'q4flag':
        ansr180.curtain(vrut.CLOSE)
        ansr90l.curtain(vrut.CLOSE)
        ansr90r.curtain(vrut.CLOSE)
        ansr0.curtain(vrut.CLOSE)
        ans4noknow.curtain(vrut.CLOSE)
    elif flag == 'q5flag':
        anshealth.curtain(vrut.CLOSE)
        anshab.curtain(vrut.CLOSE)
        ansexp.curtain(vrut.CLOSE)
        anseva.curtain(vrut.CLOSE)
        anscont.curtain(vrut.CLOSE)
        ansclose.curtain(vrut.CLOSE)
        ansstore.curtain(vrut.CLOSE)
        anscentr.curtain(vrut.CLOSE)
        ans5noknow.curtain(vrut.CLOSE)
    elif flag == 'q6flag':
        ans2health.curtain(vrut.CLOSE)
        ans2hab.curtain(vrut.CLOSE)
        ans2exp.curtain(vrut.CLOSE)
        ans2eva.curtain(vrut.CLOSE)
        ans2cont.curtain(vrut.CLOSE)
        ans2close.curtain(vrut.CLOSE)
        ans2store.curtain(vrut.CLOSE)
        ans2centr.curtain(vrut.CLOSE)
        ans6noknow.curtain(vrut.CLOSE)
    elif flag == 'q7flag':
        ans3health.curtain(vrut.CLOSE)
        ans3hab.curtain(vrut.CLOSE)
        ans3exp.curtain(vrut.CLOSE)
        ans3eva.curtain(vrut.CLOSE)
        ans3cont.curtain(vrut.CLOSE)
        ans3close.curtain(vrut.CLOSE)
        ans3store.curtain(vrut.CLOSE)
        ans3centr.curtain(vrut.CLOSE)
        ans7noknow.curtain(vrut.CLOSE)
    elif flag == 'q8flag':
        ans4health.curtain(vrut.CLOSE)
        ans4hab.curtain(vrut.CLOSE)
        ans4exp.curtain(vrut.CLOSE)
        ans4eva.curtain(vrut.CLOSE)
        ans4cont.curtain(vrut.CLOSE)
        ans4close.curtain(vrut.CLOSE)
        ans4store.curtain(vrut.CLOSE)
        ans4centr.curtain(vrut.CLOSE)
        ans8noknow.curtain(vrut.CLOSE)

def scrollAns(flag,which):
    if flag == 'q2flag':
        print(which)
        closeOthers('q2flag')
        if which == 0:
            ansblue.curtain(vrut.OPEN)
        elif which == 1:
            ansyellw.curtain(vrut.OPEN)
        else:
            print('error in q2flag')
    elif flag == 'q1flag':
        closeOthers('q1flag')
        if which == 0:
            ansupright.curtain(vrut.OPEN)
        elif which == 1:
            ansrdown.curtain(vrut.OPEN)
        elif which == 2:
            ansupdown.curtain(vrut.OPEN)
        elif which == 3:
            ansldown.curtain(vrut.OPEN)
        elif which == 4:
            ansnoknow.curtain(vrut.OPEN)
        else:
            print('error in q1flag')
    elif flag == 'q3flag':
        closeOthers('q3flag')
        if which == 0:
            ans3pit90f.curtain(vrut.OPEN)
        elif which == 1:
            ans3pit90b.curtain(vrut.OPEN)
        elif which == 2:
            ans3yaw180.curtain(vrut.OPEN)
        elif which == 3:
            ans3yaw90l.curtain(vrut.OPEN)
    elif which == 4:
        ans3yaw90r.curtain(vrut.OPEN)
    elif which == 5:
        ans3norot.curtain(vrut.OPEN)
    elif which == 6:
        ans3noknow.curtain(vrut.OPEN)
    else:
        print('error in q3flag')
elif flag == 'q4flag':
    closeOthers('q4flag')
    if which == 0:
        ansr0.curtain(vrut.OPEN)
    elif which == 1:
        ansr90l.curtain(vrut.OPEN)
    elif which == 2:
        ansr90r.curtain(vrut.OPEN)
    elif which == 3:
        ansr180.curtain(vrut.OPEN)
    elif which == 4:
        ans4noknow.curtain(vrut.OPEN)
    else:
        print('error in q4flag')
elif flag == 'q5flag':
    closeOthers('q5flag')
    if which == 0:
        ansexp.curtain(vrut.OPEN)
    elif which == 1:
        anseva.curtain(vrut.OPEN)
    elif which == 2:
        ansstore.curtain(vrut.OPEN)
    elif which == 3:
        anshealth.curtain(vrut.OPEN)
    elif which == 4:
        anshab.curtain(vrut.OPEN)
    elif which == 5:
        anscentr.curtain(vrut.OPEN)
    elif which == 6:
        anscont.curtain(vrut.OPEN)
    elif which == 7:
        ansclose.curtain(vrut.OPEN)
    elif which == 8:
        ans5noknow.curtain(vrut.OPEN)
    else:
        print('error in q5flag')
elif flag == 'q6flag':
    closeOthers('q6flag')
    print('which', which)
    if which == 0:
        ans2exp.curtain(vrut.OPEN)
    elif which == 1:
        ans2eva.curtain(vrut.OPEN)
    elif which == 2:
        ans2store.curtain(vrut.OPEN)
    elif which == 3:
        ans2health.curtain(vrut.OPEN)
    elif which == 4:
        ans2hab.curtain(vrut.OPEN)
    elif which == 5:
        ans2centr.curtain(vrut.OPEN)
    elif which == 6:
        ans2cont.curtain(vrut.OPEN)
    elif which == 7:
        ans2close.curtain(vrut.OPEN)
    elif which == 8:
        ans6noknow.curtain(vrut.OPEN)
elif flag == 'q7flag':
    closeOthers('q7flag')
    if which == 0:
        ans3exp.curtain(vrut.OPEN)
    elif which == 1:
        ans3eva.curtain(vrut.OPEN)
    elif which == 2:
        ans3store.curtain(vrut.OPEN)
    elif which == 3:
        ans3health.curtain(vrut.OPEN)
    elif which == 4:
        ans3hab.curtain(vrut.OPEN)
    elif which == 5:
        ans3centr.curtain(vrut.OPEN)
    elif which == 6:
        ans3cont.curtain(vrut.OPEN)
    elif which == 7:
        ans3close.curtain(vrut.OPEN)
    elif which == 8:
        ans7noknow.curtain(vrut.OPEN)
    else:
        print('error in q7flag')
elif flag == 'q8flag':
    closeOthers('q8flag')
    if which == 0:
        ans4exp.curtain(vrut.OPEN)
    elif which == 1:
        ans4eva.curtain(vrut.OPEN)
    elif which == 2:
        ans4store.curtain(vrut.OPEN)
    elif which == 3:
        ans4health.curtain(vrut.OPEN)
    elif which == 4:
        ans4hab.curtain(vrut.OPEN)
    elif which == 5:
        ans4centr.curtain(vrut.OPEN)
    elif which == 6:
        ans4cont.curtain(vrut.OPEN)
    elif which == 7:
        ans4close.curtain(vrut.OPEN)
    elif which == 8:
        ans8noknow.curtain(vrut.OPEN)
    else:
        print('error in q8flag')


def closeEverything():
    global set1flag
    firstQuestion.curtain(vrut.CLOSE)
    secondQuestion.curtain(vrut.CLOSE)
    line.curtain(vrut.CLOSE)
    if set1flag == on:
        ansblue.curtain(vrut.CLOSE)
        ansyellw.curtain(vrut.CLOSE)
        ansupright.curtain(vrut.CLOSE)
        ansrdown.curtain(vrut.CLOSE)
        ansupdown.curtain(vrut.CLOSE)
        ansldown.curtain(vrut.CLOSE)
        ansnoknow.curtain(vrut.CLOSE)
        ans3pit90f.curtain(vrut.CLOSE)
        ans3pit90b.curtain(vrut.CLOSE)
        ans3yaw180.curtain(vrut.CLOSE)
        ans3yaw90l.curtain(vrut.CLOSE)
        ans3yaw90r.curtain(vrut.CLOSE)
        ans3norot.curtain(vrut.CLOSE)
        ans3noknow.curtain(vrut.CLOSE)
        ansr180.curtain(vrut.CLOSE)
        ansr90l.curtain(vrut.CLOSE)
        ansr90r.curtain(vrut.CLOSE)
        ansr0.curtain(vrut.CLOSE)
        ans4noknow.curtain(vrut.CLOSE)
    elif set1flag == off:
        anshealth.curtain(vrut.CLOSE)
    anshab.curtain(vrut.CLOSE)
    ansexp.curtain(vrut.CLOSE)
    anseva.curtain(vrut.CLOSE)
    anscont.curtain(vrut.CLOSE)
    ansclose.curtain(vrut.CLOSE)
    ansstore.curtain(vrut.CLOSE)
    anscentr.curtain(vrut.CLOSE)
    ans5noknow.curtain(vrut.CLOSE)
    ans2health.curtain(vrut.CLOSE)
    ans2hab.curtain(vrut.CLOSE)
    ans2exp.curtain(vrut.CLOSE)
    ans2eva.curtain(vrut.CLOSE)
    ans2cont.curtain(vrut.CLOSE)
    ans2close.curtain(vrut.CLOSE)
    ans2store.curtain(vrut.CLOSE)
    ans2centr.curtain(vrut.CLOSE)
    ans6noknow.curtain(vrut.CLOSE)
    ans3health.curtain(vrut.CLOSE)
    ans3hab.curtain(vrut.CLOSE)
    ans3exp.curtain(vrut.CLOSE)
    ans3eva.curtain(vrut.CLOSE)
    ans3cont.curtain(vrut.CLOSE)
    ans3close.curtain(vrut.CLOSE)
    ans3store.curtain(vrut.CLOSE)
    ans3centr.curtain(vrut.CLOSE)
    ans7noknow.curtain(vrut.CLOSE)
    ans4health.curtain(vrut.CLOSE)
    ans4hab.curtain(vrut.CLOSE)
    ans4exp.curtain(vrut.CLOSE)
    ans4eva.curtain(vrut.CLOSE)
    ans4cont.curtain(vrut.CLOSE)
    ans4close.curtain(vrut.CLOSE)
    ans4store.curtain(vrut.CLOSE)
    ans4centr.curtain(vrut.CLOSE)
    ans8noknow.curtain(vrut.CLOSE)


def detWhat():
  global routeflag,module,count,moremodflag, nextflag
  print('in detWhat')
  if routeflag == off:
    return module
  else:
    routeflag = off
    if moremodflag == on:
      moremodflag = off
      return other[count][0]
    else:
      moremodflag = on
      nextflag = on
      return other[count][1]


def myKeyboard(key):
  global begintime
  if key == ' ':
    reminder.curtain(vrut.CLOSE)
    vrut.starttimer(layout,0.001)
    begintime = time.time()
    out = str(time.time()-begintime) + '\t' + str(count) +
'\tcount\n'
    keys.write(out)
    keys.flush()
  elif key == 'r':
    tracker.command(6)  # RESET1   ... see multi.py in AndyL
    tracker.reset()
  elif key == '1':
    showTarget(module,.75)
```

```
def mytimer(timernum):
  global module, count, routeflag, showing, set1flag,moremodflag,
nextflag,flyflag,model_flag
  global maxx,maxy,maxz, x,y,z, stepx,stepy,stepz, yaw,pitch,roll,
turny,turnp,turnr, startpoint
  global q1flag,q1index,q2flag,q2index,q3flag,q3index,q4flag,q4index
  global q5flag,q5index,q6flag,q6index,q7flag,q7index,q8flag,q8index
  global                                           dispx,dispy,dispz,
movez,movey,movex,boundx,boundy,boundz,
R,data,prevdata,first,begintime
  if timernum == layout:
    # initialize variables
    initVariables()
    # show target; close
    # show astronaut or pop up SIM
    if routeflag == on:
      showTarget(module,2)
      #out   =   str(initpost[module])+'\t'+str(module)+'\t'+str(-
1)+'\t'+str(-1)+'\t'+str(-1)+'\t'+str(-1)+'\t'+str(-1)+'\t'
      #out = str(initpost[module])+'\t'+str(module)+'\t')
      #file.write(out)
      #file.flush()
      vrut.starttimer(placeAstrnt,2.02)
    else:
      print(count,'this is count')
      if count == 0 or count == 1 :
        vrut.starttimer(next,0.001)
      elif count == 2 or count == 3 or count == 4 or count == 5:
        startpoint = time.time()
        vrut.starttimer(taskpoint,2.02)
      else: print('there is an error in layout')
    # first questions
    # travel to the astronaut
    # second question
    # play sound/back to Control
    # show targe; close
    # pointing task
    # first question
    # second question
    # determine what are the two other targets
    # show target of other target 1; close
    # pointing task for other target 1
    # first question
    # second question
    # show target of other target 2; close
    # point
    # first question
    # second question

  elif timernum == closeTarget:
    closeAll()

  elif timernum == placeAstrnt:
    # should only happen once every trial set
    print('placing astronaut: count',count)
    angles = defAngles(segori[module][0],segori[module][1])
    quat = Eul2Quats(angles[0],angles[1],angles[2])
    if initpost[module] == 1:
      learn = Eul2Quats(0,0,90)
      print('is astronaut right shoulder down?!')
    else:
      learn = Eul2Quats(0,0,0)
    model.avatar.translate(0,0,0)
    model.avatar.rotate(learn[0],learn[1],learn[2],learn[3])

model.avatar2.translate(geoxyz[module][0],geoxyz[module][1],geoxyz[
module][2])
    model.avatar2.rotate(quat[0],quat[1],quat[2],quat[3])
    startpoint = time.time()
```

```
        vrut.starttimer(waitRATCHET,0.1)
        if count == 0 or count == 1 :
            vrut.starttimer(showFirstQuest,2)

    elif timernum == waitRATCHET:
        if sid.buttons() == sid.BUTTON8 and model_flag == off:
            model.model.curtain(vrut.OPEN)
            tracker.command(RESET2)
            tracker.reset()
            model_flag = on
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON8)
+ '\tratchet_on\n'
            keys.write(out)
            keys.flush()
            print('within ratchet, model_flag = on')
            vrut.starttimer(move_model,0.001)
        else:
            vrut.starttimer(waitRATCHET,0.3)

    elif timernum == move_model:
        if model_flag == on and sid.buttons() == 0:
            #print('moving')
            data = wand.get()
            model.model.rotate(-data[3],-
data[4],data[5],data[6]*RAD2DEG)
            model.model.translate(offsetx+data[0],          offsety+data[1],
offsetz+data[2])
            prevdata = [data[0],data[1],data[2]]
            vrut.starttimer(move_model,0.001)
        elif model_flag == on:
            if sid.buttons() == sid.BUTTON7 or sid.buttons() ==
sid.BUTTON8:
                print('pressed flyin or ratchet, model_flag off')
                model_flag = off
                if sid.buttons() == sid.BUTTON7 and flyflag == off:
                    model.model.curtain(vrut.CLOSE)
                    tracker.command(STORE_REFFRAME)
                    if first == on:
                        first = off
                        tracker.command(USE_REFFRAME)
                    vrut.starttimer(waitRATCHET,rate)
                    out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON7) + '\thide_model_ratchet_off\n'
                    keys.write(out)
                    keys.flush()
#                   model_flag = on
#                   print('have not answered questions, model_flag on')
#                   vrut.starttimer(move_model,0.001)
                elif sid.buttons() == sid.BUTTON7 and flyflag == on:
                    # Rotate position vector of avatar by rotation of
model/sensor
                    # In data quaternions
                    a1 = -data[3]       # 4/27/2001 negative included due to
new multi sensor
                    a2 = -data[4]       # 4/27/2001 negative included due to
new multi sensor
                    a3 = data[5]
                    ang = -data[6]          # which is in radians
                    A11 = math.cos(ang); A22 = math.cos(ang); A33 =
math.cos(ang)
                    A12 = 0; A13 = 0; A21 = 0; A23 = 0; A31 = 0; A32 = 0
                    K = 1 - math.cos(ang)
                    B11 = K*a1*a1
                    B12 = K*a1*a2
                    B13 = K*a1*a3
                    B21 = K*a2*a1
                    B22 = K*a2*a2
                    B23 = K*a2*a3
                    B31 = K*a3*a1
                    B32 = K*a3*a2
                    B33 = K*a3*a3
                    S = -math.sin(ang)
                    C11 = 0
                    C12 = -S*a3
                    C13 = S*a2
                    C21 = S*a3
                    C22 = 0
                    C23 = -S*a1
                    C31 = -S*a2
                    C32 = S*a1
                    C33 = 0
                    R1 = [A11+B11+C11,A12+B12+C12,A13+B13+C13]
                    R2 = [A21+B21+C21,A22+B22+C22,A23+B23+C23]
                    R3 = [A31+B31+C31,A32+B32+C32,A33+B33+C33]
                    R = [R1, R2, R3]    # TADA! :)
                    dispx = prevdata[0]
                    dispy = prevdata[1]
                    dispz = prevdata[2]
                    prevdata = [0,0,0]
                    vrut.starttimer(flyin,0.001)
                    out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON7) + '\tflyin_ratchet_off\n'
                    keys.write(out)
                    keys.flush()
                elif sid.buttons() == sid.BUTTON8:
                    print('store refframe button8')
                    tracker.command(STORE_REFFRAME)
                    if first == on:
                        first = off
                        tracker.command(USE_REFFRAME)
                    vrut.starttimer(waitRATCHET,rate)
                    out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON8) + '\tratchet_off\n'
                    keys.write(out)
                    keys.flush()
            else:
                vrut.starttimer(move_model,0.001)
                out = str(time.time()-begintime) + '\t' + str(sid.buttons())
+ '\tincorrect_button\n'
                keys.write(out)
                keys.flush()

    elif timernum == flyin:
        movez = 0
        movey = 0
        movex = 0
        boundx = off
        boundy = off
        boundz = off
        oldx = geoxyz[module][0]*modscfac
        oldy = geoxyz[module][1]*modscfac
        oldz = geoxyz[module][2]*modscfac
        avaposx = R[0][0]*oldx + R[0][1]*oldy + R[0][2]*oldz
        avaposy = R[1][0]*oldx + R[1][1]*oldy + R[1][2]*oldz
        avaposz = R[2][0]*oldx + R[2][1]*oldy + R[2][2]*oldz
        dispx = offsetx + data[0]
        dispy = offsety + data[1]
        dispz = offsetz + data[2]
        maxx = -(dispx + avaposx)
        maxy = -(dispy + avaposy)
        maxz = -(dispz + avaposz)
        stepx = maxx/flytime
        stepy = maxy/flytime
        stepz = maxz/flytime
        tracker.command(STORE_REFFRAME)
        if first == on:
            first = off
            tracker.command(USE_REFFRAME)
        station.curtain(vrut.CLOSE)
        vrut.starttimer(flyin2,0.001)

    elif timernum == flyin2:
        if abs(movex) > abs(maxx): boundx = on
        if abs(movey) > abs(maxy): boundy = on
        if abs(movez) > abs(maxz): boundz = on
        if boundx == off or boundy == off or boundz == off:
            if boundx == off:
                movex = movex + stepx
            if boundy == off:
                movey = movey + stepy
            if boundz == off:
                movez = movez + stepz

model.model.translate(dispx+movex,dispy+movey,dispz+movez)
            vrut.starttimer(flyin2,0.001)
        else:
            vrut.reset(vrut.HEAD_POS)

vrut.translate(vrut.HEAD_POS,geoxyz[module][0],geoxyz[module][1
],geoxyz[module][2])
            vrut.reset(vrut.BODY_ORI)
            angles = defAngles(segori[module][0],segori[module][1])
            vrut.rotate(vrut.BODY_ORI,angles[0],-angles[1],angles[2])
            print('are the body rotations correct???')
            model.model.curtain(vrut.CLOSE)
            station.curtain(vrut.OPEN)
            set1flag = off
            line.translate(0,lay,lz)
            if count == 0 or count == 1 :
                vrut.starttimer(showSecondQuest,2)
            elif count == 2 or count == 3 or count == 4 or count == 5:
                vrut.starttimer(waitFLYIN,0.1)
            else:
                print('there is something wrong at the end of flyin')

    elif timernum == next:
        if count == 0 or count == 1 :
            routeflag = on
            set1flag = on
            print('increasing count')
            if count == maxtrials:
                print('endprogram!!!')
                vrut.translate(vrut.HEAD_POS, 0,100,0)
                file.close()
                keys.close()
            else:
                count = count + 1
                vrut.starttimer(layout,0.001)
        elif count == 2 or count == 3 or count == 4 or count == 5:
            set1flag = off
            routeflag = off
            vrut.starttimer(layout,0.001)

    elif timernum == waitFLYIN:
        if sid.buttons() == sid.BUTTON7:
            vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\Reminder.wav')
            vrut.starttimer(next,0.5)
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON7)
+ '\tend_of_route\n'
            keys.write(out)
            keys.flush()
        else:
            vrut.starttimer(waitFLYIN,0.3)

    elif timernum == waitPOP:
        if sid.buttons() == sid.BUTTON10:
            if set1flag == on:
                vrut.starttimer(showFirstQuest,0.001)
            else:
                vrut.starttimer(showSecondQuest,0.001)
        else:
            vrut.starttimer(waitPOP,rate)

    elif timernum == showFirstQuest:
        if showing == on:
            model.model.curtain(vrut.OPEN)
            showing = off
            closeEverything()
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\thide_quest_set1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(waitPOP,rate)
        elif showing == off:
            showing = on
            model.model.curtain(vrut.CLOSE)
            firstQuestion.curtain(vrut.OPEN)
            line.curtain(vrut.OPEN)
            scrollAns('q1flag',q1index)
            scrollAns('q2flag',q2index)
            scrollAns('q3flag',q3index)
            scrollAns('q4flag',q4index)
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\tshow_quest_set1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)

    elif timernum == showSecondQuest:
        if showing == on:
            showing = off
            closeEverything()
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\thide_quest_set2\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(waitPOP,rate)
        elif showing == off:
            showing = on
            secondQuestion.curtain(vrut.OPEN)
            line.curtain(vrut.OPEN)
            print('2nd question',q5index,q6index,q7index,q8index)
            scrollAns('q5flag',q5index)
            scrollAns('q6flag',q6index)
            scrollAns('q7flag',q7index)
            scrollAns('q8flag',q8index)
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON10)
+ '\tshow_quest_set2\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)

    elif timernum == scroller:
        if sid.buttons() == sid.BUTTON10 and routeflag == on:
            if set1flag == on:
                vrut.starttimer(showFirstQuest,0.001)
            else:
                vrut.starttimer(showSecondQuest,0.001)
            # watch out for a possible BUG HERE!
        elif sid.buttons() == sid.BUTTON1:
            # check for which question
            # scroll up (add)
            if q2flag == on:
                if q2index == 0:
                    q2index = 1
            else:
               q2index = 0
            scrollAns('q2flag',q2index)
         elif q1flag == on:
            if q1index == 4:
               q1index = 0
            else:
               q1index = q1index + 1
            scrollAns('q1flag',q1index)
         elif q3flag == on:
            if q3index == 6:
               q3index = 0
            else:
               q3index = q3index + 1
            scrollAns('q3flag',q3index)
         elif q4flag == on:
            if q4index == 4:
               q4index = 0
            else:
               q4index = q4index + 1
            scrollAns('q4flag',q4index)
         elif q5flag == on:
            if q5index == 8:
               q5index = 0
            else:
               q5index = q5index + 1
            scrollAns('q5flag',q5index)
         elif q6flag == on:
            if q6index == 8:
               q6index = 0
            else:
               q6index = q6index + 1
            scrollAns('q6flag',q6index)
         elif q7flag == on:
            if q7index == 8:
               q7index = 0
            else:
               q7index = q7index + 1
            scrollAns('q7flag',q7index)
         elif q8flag == on:
            if q8index == 8:
               q8index = 0
            else:
               q8index = q8index + 1
            scrollAns('q8flag',q8index)
         vrut.starttimer(scroller,rate)
         out = str(time.time()-begintime) + '\t' + str(sid.BUTTON1)
+ '\tscroll_up\n'
         keys.write(out)
         keys.flush()

      elif sid.buttons() == sid.BUTTON4:
         # check for which question
         # scroll down (subtract)
         if q2flag == on:
            if q2index == 1:
               q2index = 0
            else:
               q2index = 1
            scrollAns('q2flag',q2index)
         elif q1flag == on:
            if q1index == 0:
               q1index = 4
            else:
               q1index = q1index - 1
            scrollAns('q1flag',q1index)
         elif q3flag == on:
            if q3index == 0:
               q3index = 6
            else:
               q3index = q3index - 1
            scrollAns('q3flag',q3index)
         elif q4flag == on:
            if q4index == 0:
               q4index = 4
            else:
               q4index = q4index - 1
            scrollAns('q4flag',q4index)
         elif q5flag == on:
            if q5index == 0:
               q5index = 8
            else:
               q5index = q5index - 1
            scrollAns('q5flag',q5index)
         elif q6flag == on:
            if q6index == 0:
               q6index = 8
            else:
               q6index = q6index - 1
            scrollAns('q6flag',q6index)
         elif q7flag == on:
            if q7index == 0:
               q7index = 8
            else:
               q7index = q7index - 1
            scrollAns('q7flag',q7index)
         elif q8flag == on:
            if q8index == 0:
               q8index = 8
            else:
               q8index = q8index - 1
            scrollAns('q8flag',q8index)
         vrut.starttimer(scroller,rate)
         out = str(time.time()-begintime) + '\t' + str(sid.BUTTON4)
+ '\tscroll_down\n'
         keys.write(out)
         keys.flush()

      elif sid.buttons() == 768:   # pressing sid.BUTTON9 and
sid.BUTTON10 together
         elapstime = time.time() - startpoint
         startpoint = time.time()
         closeEverything()
         showing = off
         out = str(time.time()-begintime) + '\t' + str(768) +
'\tenter_hide_quest_set\n'
         keys.write(out)
         keys.flush()
         if set1flag == on:
            q1flag = off
            q2flag = off
            q3flag = off
            q4flag = off
            q5flag = on
            out                                                    =
str(q1index)+'\t'+str(q2index)+'\t'+str(q3index)+'\t'+str(q4index)+'
\t'+ str(elapstime)+'\t'
            file.write(out)
            file.flush()
            if routeflag == on:
               flyflag = on
               model.model.curtain(vrut.OPEN)
               #routeflag = off
               # NO waiting for flyin here! completely different
               # from control. Just follow model/ratchet program
               #vrut.starttimer(waitFLYIN,rate)
               vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\EndQuest.wav')
```

```
                print('not going to FLYIN')
            else:
                print('repeating??')
                #print(set1flag, routeflag)
                set1flag = off
                line.translate(0,lay,lz)
                startpoint = time.time()
                vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\EndQuest.wav')
                if nextflag == off and moremodflag == on:
                    vrut.starttimer(showSecondQuest,0.1)
                else:
                    q5flag = off
                    q1flag = on
                    line.translate(0,lay,lz)
                    routeflag = on
                    set1flag = on
                    out    =    str(-1)+'\t'+str(-1)+'\t'+str(-1)+'\t'+str(-
1)+'\t'+str(-1)+'\t'+'\n'
                    file.write(out)
                    file.flush()
                    if nextflag == on:
                        print('increasing count')
                        if count == maxtrials:
                            print('endprogram!!!')
                            vrut.translate(vrut.HEAD_POS, 0,100,0)
                            file.close()
                            keys.close()
                        else:
                            count = count + 1
                            vrut.starttimer(layout,0.001)
                    else:
                        startpoint = time.time()
                        vrut.starttimer(taskpoint,1)
        else:
            q5flag = off
            q6flag = off
            q7flag = off
            q8flag = off
            q1flag = on
            line.translate(0,lay,lz)
            out                                                          =
str(q5index)+'\t'+str(q6index)+'\t'+str(q7index)+'\t'+str(q8index)+'
\t'+ str(elapstime)+'\n'
            file.write(out)
            file.flush()
            if routeflag == on:
                routeflag = off
                print('here')
                vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\Reminder.wav')
                vrut.starttimer(layout,2)
            else:
                routeflag = on
                set1flag = on
                if nextflag == on:
                    print('increasing count')
                    if count == maxtrials:
                        print('endprogram!!!')
                        vrut.translate(vrut.HEAD_POS, 0,100,0)
                        file.close()
                        keys.close()
                    else:
                        count = count + 1
                        vrut.starttimer(layout,0.001)
                else:
                    startpoint = time.time()
                    vrut.starttimer(taskpoint,1)
                #vrut.starttimer(taskpoint,0.1)
```

```
    elif sid.buttons() == sid.BUTTON9:
        # enter answer  (record)
        # increase which question counter
        if q1flag == on:
            q1flag = off
            q2flag = on
            line.translate(0,lby,lz)
            scrollAns('q2flag',q2index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q2flag == on:
            q2flag = off
            q3flag = on
            line.translate(0,lcy,lz)
            scrollAns('q3flag',q3index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question2\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q3flag == on:
            q3flag = off
            q4flag = on
            line.translate(0,ldy,lz)
            scrollAns('q4flag',q4index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question3\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q4flag == on:
            q4flag = off
            q1flag = on
            line.translate(0,lay,lz)
            scrollAns('q1flag',q1index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question4\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q5flag == on:
            q5flag = off
            q6flag = on
            line.translate(0,lby,lz)
            scrollAns('q6flag',q6index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question5\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q6flag == on:
            q6flag = off
            q7flag = on
            line.translate(0,lcy,lz)
            scrollAns('q7flag',q7index)
            out     =     str(time.time()-begintime)     +     '\t'     +
str(sid.BUTTON9) + '\tenter_question6\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q7flag == on:
            q7flag = off
            q8flag = on
            line.translate(0,ldy,lz)
            scrollAns('q8flag',q8index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question7\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
         elif q8flag == on:
            q8flag = off
            q5flag = on
            line.translate(0,lay,lz)
            scrollAns('q5flag',q5index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question8\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)

      else:
         vrut.starttimer(scroller,rate)

   elif timernum == taskpoint:
      what = detWhat()
      print('what',what)
      showTarget(what,2)
      out = str(initpost[count])+'\t'+str(what)+'\t'
      file.write(out)
      file.flush()
      arrow.curtain(vrut.OPEN)
      #startpoint = time.time()
      vrut.starttimer(waitENTER,0.001)

   elif timernum == waitENTER:
      if sid.buttons() == sid.BUTTON9:
         arrow.curtain(vrut.CLOSE)
         data = tracker.get()
         elapstime = time.time() - startpoint
         out                                                          =
str(data[3])+'\t'+str(data[4])+'\t'+str(data[5])+'\t'+str(data[6]*RAD2
DEG)+'\t'+str(elapstime)+'\t'
         file.write(out)
         file.flush()
         startpoint = time.time()
         out = str(time.time()-begintime) + '\t' + str(sid.BUTTON9)
+ '\tenter_pointing\n'
         keys.write(out)
         keys.flush()
         vrut.starttimer(showFirstQuest,1)
      else:
         vrut.starttimer(waitENTER,rate)




#
*****************************************************************
*********************

# Setting up output files
exp_time = time.localtime(time.time())
timestring = time.strftime('%b%d_%H%M',exp_time)
filename                    =                    'N:\Jessica\Thesis
SIM\Data\Sim\Phase2\Phase2b_SIM'+str(timestring)+'.dat'
file = open(filename,'w')
out                                                               =
'Posture\tTarget\tQuat0\tQuat1\tQuat2\tQuat3\tTime\tQ1\tQ2\
tQ3\tQ4\tTime1\tQ5\tQ6\tQ7\tQ8\tTime2\n'
file.write(out)
file.flush()
# second file out
filename2                   =                    'N:\Jessica\Thesis
SIM\Data\Sim\Phase2\Phase2b_SIM'+str(timestring)+'_allkeys.dat'
keys = open(filename2,'w')


vrut.callback(vrut.KEYBOARD_EVENT,'myKeyboard')
vrut.callback(vrut.TIMER_EVENT,'mytimer')

reminder.curtain(vrut.OPEN)
```

Step 3: Measuring 3-D Knowledge of Station Layout and Routes

This program was the same for Control and SIM groups

```
import vrut
import time
import math
import types
import string
import sid

# 6/7/2001  This is an altered version of Phase 3 program with
wand.
#        it will include the new protocol/sequences and the wand
scrolling.
# 6/27/2001 eliminating second set of questions.
# 6/29/2001 Intersense sensor is broken.  Thus in order to continue
programming
#       I had to comment out all the sensor lines with #$.  To later
erase,
#       search for #$ comments and delete those.  -- jjm

#
*****************************************************************
*********************
# KEY:
#
*****************************************************************
*********************

#    1                          direction nomenclature
#    |
# 2 ----- 0 into the page --  4
#    |                  out of the page -- 5
#    3
#
# Target difficulty:
#          0: modules 0 or 1
#          1: modules 4 or 5
#          2: modules 2 or 3
# FarNode:          I    MIGHT    HAVE    GOTTEN    THIS
BACKWARDS... 4/2/2001
#     0: modules 1, 3, or 5
#          1: modules 0, 2, or 4
# Posture:  relative to module 6
#          0: roll, with nose at 4 & feet at 1
#          1: pitch, (positive) with nose at 1 & feet at 4
#          2: roll, with nose at 4 & feet at 2
#          3: pitch, (negative) with nose at 3 & feet at 5
#
# Module labelling:
#          0: Columbus, European Module      == Experiment
#          1: Stowage Module              == EVA
#          2: Zarya, Russian Control Module   == Storage
#          3: Destiny, U.S. Module          == Health Fitness
#          4: Habitation Module            == Habitation
#          5: Kibo, Japanese Module         == Centrifuge
#          6: Zvezda, Russian Service Module  == Control
#          Nodes
#          7: FarNode 1, hatch at end of zvezda (blue)          #
6/4/2001  had these reversed
#          8: FarNode 0, hatch at other end of zvezda (yellow)
#
#                                                        Constants
*****************************************************************
************


on = 1
off = 0
RAD2DEG = 57.295779513082323
DEG2RAD = 1/57.295779513082323
rate = 0.3
maxtrials = 24

HMD = on

tdif = 0
fnod = 1
pstr = 2

qax = .3
qay = .435
qby = .15
qcy = -.15
qdy = -.435
qaz = 2.05
lay = .275
lby = 0
lcy =-.275
ldy = -.57
lz  = 2.0
ansscfac = 4
scfac = 2.25

#
*****************************************************************
*********************


if HMD == on:
   vrut.go(vrut.HMD | vrut.STEREO)
else:
   vrut.go(vrut.CONSOLE)
   vrut.translate(vrut.HEAD_POS, 0,-1.82,0)

#
*****************************************************************
*********************


# Sensor
tracker = vrut.addsensor('isensemulti_beta')   # just rotations??
vrut.tracker()
tracker.command(6)  # RESET1    ... see multi.py in AndyL
tracker.reset()

#
*****************************************************************
*********************
# Loading objects and their initial positions

station = vrut.addchild('n:\Jessica\Thesis SIM\Models\Station2.wrl')
station.curtain(vrut.CLOSE)
```

```
astronaut          =                 vrut.addchild('n:\Jessica\Thesis
SIM\Models\Astronaut.wrl')
astronaut.curtain(vrut.CLOSE)
arrow              =                 vrut.addchild('n:\Jessica\Thesis
SIM\Models\Cross.wrl',vrut.HEAD)
arrow.translate(0,0,.75)
arrow.curtain(vrut.CLOSE)

firstQuestion          =             vrut.addchild('n:\Jessica\Thesis
SIM\Models\Firstquestion.wrl',vrut.HEAD)
firstQuestion.translate(0,0,1.5)
firstQuestion.scale(scfac,scfac,scfac)
firstQuestion.curtain(vrut.CLOSE)
secondQuestion          =            vrut.addchild('n:\Jessica\Thesis
SIM\Models\Secondquestion.wrl',vrut.HEAD)
secondQuestion.translate(0,0,1.5)
secondQuestion.scale(scfac,scfac,scfac)
secondQuestion.curtain(vrut.CLOSE)
line               =                 vrut.addchild('n:\Jessica\Thesis
SIM\Models\Underline.wrl',vrut.HEAD)
line.translate(0,lay,lz)
line.scale(scfac,scfac,scfac)
line.curtain(vrut.CLOSE)

ansblue            =                 vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansbluehatch.wrl',vrut.HEAD)
ansyellw           =                 vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyellowhatch.wrl',vrut.HEAD)
ansblue.translate(qax,qby,qaz)
ansblue.scale(ansscfac,ansscfac,ansscfac)
ansyellw.translate(qax,qby,qaz)
ansyellw.scale(ansscfac,ansscfac,ansscfac)
ansblue.curtain(vrut.CLOSE)
ansyellw.curtain(vrut.CLOSE)

ansupright           =               vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansupright.wrl',vrut.HEAD)
ansrdown             =               vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansrightdown.wrl',vrut.HEAD)
ansupdown            =               vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansupsidedown.wrl',vrut.HEAD)
ansldown             =               vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansleftdown.wrl',vrut.HEAD)
ansnoknow            =               vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansupright.translate(qax,qay,qaz)
ansupright.scale(ansscfac,ansscfac,ansscfac)
ansrdown.translate(qax,qay,qaz)
ansrdown.scale(ansscfac,ansscfac,ansscfac)
ansupdown.translate(qax,qay,qaz)
ansupdown.scale(ansscfac,ansscfac,ansscfac)
ansldown.translate(qax,qay,qaz)
ansldown.scale(ansscfac,ansscfac,ansscfac)
ansnoknow.translate(qax,qay,qaz)
ansnoknow.scale(ansscfac,ansscfac,ansscfac)
ansupright.curtain(vrut.CLOSE)
ansrdown.curtain(vrut.CLOSE)
ansupdown.curtain(vrut.CLOSE)
ansldown.curtain(vrut.CLOSE)
ansnoknow.curtain(vrut.CLOSE)

ans3pit90f           =               vrut.addchild('n:\Jessica\Thesis
SIM\Models\Anspitch90f.wrl',vrut.HEAD)
ans3pit90b           =               vrut.addchild('n:\Jessica\Thesis
SIM\Models\Anspitch90b.wrl',vrut.HEAD)
ans3yaw180           =               vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyaw180.wrl',vrut.HEAD)
ans3yaw90l           =               vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyaw90l.wrl',vrut.HEAD)
```

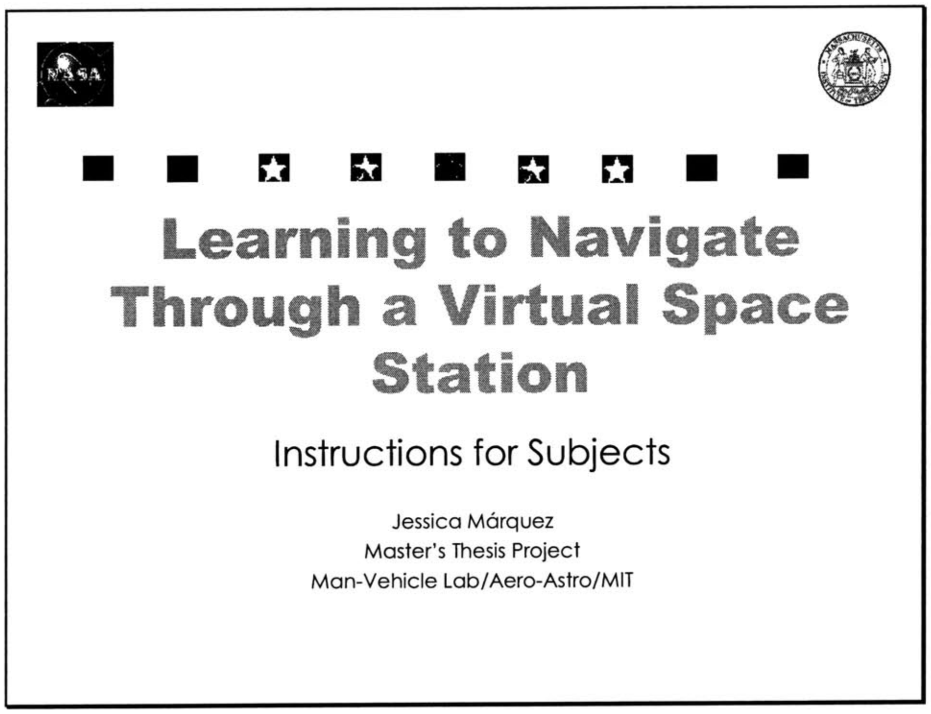
```
ans3yaw90r          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansyaw90r.wrl',vrut.HEAD)
ans3norot           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansnorotate.wrl',vrut.HEAD)
ans3noknow          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans3pit90f.translate(qax,qcy,qaz)
ans3pit90f.scale(ansscfac,ansscfac,ansscfac)
ans3pit90b.translate(qax,qcy,qaz)
ans3pit90b.scale(ansscfac,ansscfac,ansscfac)
ans3yaw180.translate(qax,qcy,qaz)
ans3yaw180.scale(ansscfac,ansscfac,ansscfac)
ans3yaw90l.translate(qax,qcy,qaz)
ans3yaw90l.scale(ansscfac,ansscfac,ansscfac)
ans3yaw90r.translate(qax,qcy,qaz)
ans3yaw90r.scale(ansscfac,ansscfac,ansscfac)
ans3norot.translate(qax,qcy,qaz)
ans3norot.scale(ansscfac,ansscfac,ansscfac)
ans3noknow.translate(qax,qcy,qaz)
ans3noknow.scale(ansscfac,ansscfac,ansscfac)
ans3pit90f.curtain(vrut.CLOSE)
ans3pit90b.curtain(vrut.CLOSE)
ans3yaw180.curtain(vrut.CLOSE)
ans3yaw90l.curtain(vrut.CLOSE)
ans3yaw90r.curtain(vrut.CLOSE)
ans3norot.curtain(vrut.CLOSE)
ans3noknow.curtain(vrut.CLOSE)

ansr180             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll180.wrl',vrut.HEAD)
ansr90l             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll90l.wrl',vrut.HEAD)
ansr90r             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll90r.wrl',vrut.HEAD)
ansr0               =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansroll0.wrl',vrut.HEAD)
ans4noknow          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansr0.translate(qax,qdy,qaz)
ansr0.scale(ansscfac,ansscfac,ansscfac)
ansr90l.translate(qax,qdy,qaz)
ansr90l.scale(ansscfac,ansscfac,ansscfac)
ansr90r.translate(qax,qdy,qaz)
ansr90r.scale(ansscfac,ansscfac,ansscfac)
ansr180.translate(qax,qdy,qaz)
ansr180.scale(ansscfac,ansscfac,ansscfac)
ans4noknow.translate(qax,qdy,qaz)
ans4noknow.scale(ansscfac,ansscfac,ansscfac)
ansr180.curtain(vrut.CLOSE)
ansr90l.curtain(vrut.CLOSE)
ansr90r.curtain(vrut.CLOSE)
ansr0.curtain(vrut.CLOSE)
ans4noknow.curtain(vrut.CLOSE)

anshealth           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
anshab              =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ansexp              =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
anseva              =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
anscont             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ansclose            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ansstore            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
anscentr            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans5noknow          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ansexp.translate(qax,qay,qaz)
ansexp.scale(ansscfac,ansscfac,ansscfac)
anseva.translate(qax,qay,qaz)
anseva.scale(ansscfac,ansscfac,ansscfac)
ansstore.translate(qax,qay,qaz)
ansstore.scale(ansscfac,ansscfac,ansscfac)
anshealth.translate(qax,qay,qaz)
anshealth.scale(ansscfac,ansscfac,ansscfac)
anshab.translate(qax,qay,qaz)
anshab.scale(ansscfac,ansscfac,ansscfac)
anscentr.translate(qax,qay,qaz)
anscentr.scale(ansscfac,ansscfac,ansscfac)
anscont.translate(qax,qay,qaz)
anscont.scale(ansscfac,ansscfac,ansscfac)
ansclose.translate(qax,qay,qaz)
ansclose.scale(ansscfac,ansscfac,ansscfac)
ans5noknow.translate(qax,qay,qaz)
ans5noknow.scale(ansscfac,ansscfac,ansscfac)
anshealth.curtain(vrut.CLOSE)
anshab.curtain(vrut.CLOSE)
ansexp.curtain(vrut.CLOSE)
anseva.curtain(vrut.CLOSE)
anscont.curtain(vrut.CLOSE)
ansclose.curtain(vrut.CLOSE)
ansstore.curtain(vrut.CLOSE)
anscentr.curtain(vrut.CLOSE)
ans5noknow.curtain(vrut.CLOSE)

ans2health          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
ans2hab             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans2exp             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
ans2eva             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
ans2cont            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans2close           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans2store           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
ans2centr           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans6noknow          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans2exp.translate(qax,qby,qaz)
ans2exp.scale(ansscfac,ansscfac,ansscfac)
ans2eva.translate(qax,qby,qaz)
ans2eva.scale(ansscfac,ansscfac,ansscfac)
ans2store.translate(qax,qby,qaz)
ans2store.scale(ansscfac,ansscfac,ansscfac)
ans2health.translate(qax,qby,qaz)
ans2health.scale(ansscfac,ansscfac,ansscfac)
ans2hab.translate(qax,qby,qaz)
ans2hab.scale(ansscfac,ansscfac,ansscfac)
ans2centr.translate(qax,qby,qaz)
ans2centr.scale(ansscfac,ansscfac,ansscfac)
ans2cont.translate(qax,qby,qaz)
ans2cont.scale(ansscfac,ansscfac,ansscfac)
ans2close.translate(qax,qby,qaz)
ans2close.scale(ansscfac,ansscfac,ansscfac)
ans6noknow.translate(qax,qby,qaz)
ans6noknow.scale(ansscfac,ansscfac,ansscfac)
ans2health.curtain(vrut.CLOSE)
```

```
ans2hab.curtain(vrut.CLOSE)
ans2exp.curtain(vrut.CLOSE)
ans2eva.curtain(vrut.CLOSE)
ans2cont.curtain(vrut.CLOSE)
ans2close.curtain(vrut.CLOSE)
ans2store.curtain(vrut.CLOSE)
ans2centr.curtain(vrut.CLOSE)
ans6noknow.curtain(vrut.CLOSE)

ans3health          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
ans3hab             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans3exp             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
ans3eva             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
ans3cont            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans3close           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans3store           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
ans3centr           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans7noknow          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans3exp.translate(qax,qcy,qaz)
ans3exp.scale(ansscfac,ansscfac,ansscfac)
ans3eva.translate(qax,qcy,qaz)
ans3eva.scale(ansscfac,ansscfac,ansscfac)
ans3store.translate(qax,qcy,qaz)
ans3store.scale(ansscfac,ansscfac,ansscfac)
ans3health.translate(qax,qcy,qaz)
ans3health.scale(ansscfac,ansscfac,ansscfac)
ans3hab.translate(qax,qcy,qaz)
ans3hab.scale(ansscfac,ansscfac,ansscfac)
ans3centr.translate(qax,qcy,qaz)
ans3centr.scale(ansscfac,ansscfac,ansscfac)
ans3cont.translate(qax,qcy,qaz)
ans3cont.scale(ansscfac,ansscfac,ansscfac)
ans3close.translate(qax,qcy,qaz)
ans3close.scale(ansscfac,ansscfac,ansscfac)
ans7noknow.translate(qax,qcy,qaz)
ans7noknow.scale(ansscfac,ansscfac,ansscfac)
ans3health.curtain(vrut.CLOSE)
ans3hab.curtain(vrut.CLOSE)
ans3exp.curtain(vrut.CLOSE)
ans3eva.curtain(vrut.CLOSE)
ans3cont.curtain(vrut.CLOSE)
ans3close.curtain(vrut.CLOSE)
ans3store.curtain(vrut.CLOSE)
ans3centr.curtain(vrut.CLOSE)
ans7noknow.curtain(vrut.CLOSE)

ans4health          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhealth.wrl',vrut.HEAD)
ans4hab             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modhabitation.wrl',vrut.HEAD)
ans4exp             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modexperiment.wrl',vrut.HEAD)
ans4eva             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modeva.wrl',vrut.HEAD)
ans4cont            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcontrol.wrl',vrut.HEAD)
ans4close           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modclosedhatch.wrl',vrut.HEAD)
ans4store           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modstorage.wrl',vrut.HEAD)
ans4centr           =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Modcentrifuge.wrl',vrut.HEAD)
ans8noknow          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Ansdontknow.wrl',vrut.HEAD)
ans4exp.translate(qax,qdy,qaz)
ans4exp.scale(ansscfac,ansscfac,ansscfac)
ans4eva.translate(qax,qdy,qaz)
ans4eva.scale(ansscfac,ansscfac,ansscfac)
ans4store.translate(qax,qdy,qaz)
ans4store.scale(ansscfac,ansscfac,ansscfac)
ans4health.translate(qax,qdy,qaz)
ans4health.scale(ansscfac,ansscfac,ansscfac)
ans4hab.translate(qax,qdy,qaz)
ans4hab.scale(ansscfac,ansscfac,ansscfac)
ans4centr.translate(qax,qdy,qaz)
ans4centr.scale(ansscfac,ansscfac,ansscfac)
ans4cont.translate(qax,qdy,qaz)
ans4cont.scale(ansscfac,ansscfac,ansscfac)
ans4close.translate(qax,qdy,qaz)
ans4close.scale(ansscfac,ansscfac,ansscfac)
ans8noknow.translate(qax,qdy,qaz)
ans8noknow.scale(ansscfac,ansscfac,ansscfac)
ans4health.curtain(vrut.CLOSE)
ans4hab.curtain(vrut.CLOSE)
ans4exp.curtain(vrut.CLOSE)
ans4eva.curtain(vrut.CLOSE)
ans4cont.curtain(vrut.CLOSE)
ans4close.curtain(vrut.CLOSE)
ans4store.curtain(vrut.CLOSE)
ans4centr.curtain(vrut.CLOSE)
ans8noknow.curtain(vrut.CLOSE)

columbus            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Columbus.wrl',vrut.HEAD)
stowage             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Stowage.wrl',vrut.HEAD)
zarya               =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Zarya.wrl',vrut.HEAD)
destiny             =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Destiny.wrl',vrut.HEAD)
habitation          =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Habitation.wrl',vrut.HEAD)
kibo                =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Kibo.wrl',vrut.HEAD)
reminder            =          vrut.addchild('n:\Jessica\Thesis
SIM\Models\Reminder.wrl', vrut.HEAD)
columbus.scale(0.25,.25,.25)
stowage.scale(0.25,.25,.25)
zarya.scale(0.25,.25,.25)
destiny.scale(0.25,.25,.25)
habitation.scale(0.25,.25,.25)
kibo.scale(0.25,.25,.25)
columbus.translate(0,-.1,.5)
stowage.translate(0,-.1,.5)
zarya.translate(0,-.1,.5)
destiny.translate(0,-.1,.5)
habitation.translate(0,-.1,.5)
kibo.translate(0,-.1,.5)
reminder.translate(0,-.4,2)
columbus.curtain(vrut.CLOSE)
stowage.curtain(vrut.CLOSE)
zarya.curtain(vrut.CLOSE)
destiny.curtain(vrut.CLOSE)
habitation.curtain(vrut.CLOSE)
kibo.curtain(vrut.CLOSE)
#reminder.curtain(vrut.CLOSE)

# posture: 0 == upright, 1 == right shoulder down, 2 == upside
down, 3 == left shoulder down (only rolls!!)
```

```
posture = [0,-90,180,90]
trialset                                                              =
[[0,0],[2,3],[1,5],[2,1],[3,4],[2,2],[1,3],[1,4],[1,2],[3,5],[0,1],[2,0],
        [0,2],[1,1],[2,5],[3,3],[1,0],[2,4],[0,3],[3,2],[3,1],[0,4],[3,0],[0,5]]

global q1flag,q2flag,q3flag,q4flag,q5flag,q6flag,q7flag,q8flag
global
q1index,q2index,q3index,q4index,q5index,q6index,q7index,q8index
global count, set1flag, begintime
q1flag = off
q2flag = off
q3flag = off
q4flag = off
q5flag = off
q6flag = off
q7flag = off
q8flag = off
q1index = 0
q2index = 0
q3index = 0
q4index = 0
q5index = 0
q6index = 0
q7index = 0
q8index = 0
count = 0
set1flag = on

# timernums
start_trial = 1
closeTarget = 2
scroller = 3
showFirstQuest = 4
showSecondQuest = 5
waitENTER = 6

#
*****************************************************************
**********************

def showTarget(modulenum, tclose):
    if modulenum == 0:
        columbus.curtain(vrut.OPEN)
    elif modulenum == 1:
        stowage.curtain(vrut.OPEN)
    elif modulenum == 2:
        zarya.curtain(vrut.OPEN)
    elif modulenum == 3:
        destiny.curtain(vrut.OPEN)
    elif modulenum == 4:
        habitation.curtain(vrut.OPEN)
    elif modulenum == 5:
        kibo.curtain(vrut.OPEN)
    vrut.starttimer(closeTarget,tclose)

def closeAll():
    columbus.curtain(vrut.CLOSE)
    stowage.curtain(vrut.CLOSE)
    zarya.curtain(vrut.CLOSE)
    destiny.curtain(vrut.CLOSE)
    habitation.curtain(vrut.CLOSE)
    kibo.curtain(vrut.CLOSE)

def closeOthers(flag):
    #global q1flag,q2flag,q3flag,q4flag,q5flag,q6flag,q7flag,q8flag
    if flag == 'q2flag':
        ansblue.curtain(vrut.CLOSE)
        ansyellw.curtain(vrut.CLOSE)
    elif flag == 'q1flag':
        ansupright.curtain(vrut.CLOSE)
        ansrdown.curtain(vrut.CLOSE)
        ansupdown.curtain(vrut.CLOSE)
        ansldown.curtain(vrut.CLOSE)
        ansnoknow.curtain(vrut.CLOSE)
    elif flag == 'q3flag':
        ans3pit90f.curtain(vrut.CLOSE)
        ans3pit90b.curtain(vrut.CLOSE)
        ans3yaw180.curtain(vrut.CLOSE)
        ans3yaw90l.curtain(vrut.CLOSE)
        ans3yaw90r.curtain(vrut.CLOSE)
        ans3norot.curtain(vrut.CLOSE)
        ans3noknow.curtain(vrut.CLOSE)
    elif flag == 'q4flag':
        ansr180.curtain(vrut.CLOSE)
        ansr90l.curtain(vrut.CLOSE)
        ansr90r.curtain(vrut.CLOSE)
        ansr0.curtain(vrut.CLOSE)
        ans4noknow.curtain(vrut.CLOSE)
    elif flag == 'q5flag':
        anshealth.curtain(vrut.CLOSE)
        anshab.curtain(vrut.CLOSE)
        ansexp.curtain(vrut.CLOSE)
        anseva.curtain(vrut.CLOSE)
        anscont.curtain(vrut.CLOSE)
        ansclose.curtain(vrut.CLOSE)
        ansstore.curtain(vrut.CLOSE)
        anscentr.curtain(vrut.CLOSE)
        ans5noknow.curtain(vrut.CLOSE)
    elif flag == 'q6flag':
        ans2health.curtain(vrut.CLOSE)
        ans2hab.curtain(vrut.CLOSE)
        ans2exp.curtain(vrut.CLOSE)
        ans2eva.curtain(vrut.CLOSE)
        ans2cont.curtain(vrut.CLOSE)
        ans2close.curtain(vrut.CLOSE)
        ans2store.curtain(vrut.CLOSE)
        ans2centr.curtain(vrut.CLOSE)
        ans6noknow.curtain(vrut.CLOSE)
    elif flag == 'q7flag':
        ans3health.curtain(vrut.CLOSE)
        ans3hab.curtain(vrut.CLOSE)
        ans3exp.curtain(vrut.CLOSE)
        ans3eva.curtain(vrut.CLOSE)
        ans3cont.curtain(vrut.CLOSE)
        ans3close.curtain(vrut.CLOSE)
        ans3store.curtain(vrut.CLOSE)
        ans3centr.curtain(vrut.CLOSE)
        ans7noknow.curtain(vrut.CLOSE)
    elif flag == 'q8flag':
        ans4health.curtain(vrut.CLOSE)
        ans4hab.curtain(vrut.CLOSE)
        ans4exp.curtain(vrut.CLOSE)
        ans4eva.curtain(vrut.CLOSE)
        ans4cont.curtain(vrut.CLOSE)
        ans4close.curtain(vrut.CLOSE)
        ans4store.curtain(vrut.CLOSE)
        ans4centr.curtain(vrut.CLOSE)
        ans8noknow.curtain(vrut.CLOSE)

def scrollAns(flag,which):
    if flag == 'q2flag':
        print(which)
        closeOthers('q2flag')
        if which == 0:
            ansblue.curtain(vrut.OPEN)
        elif which == 1:
            ansyellw.curtain(vrut.OPEN)
        else:
            print('error in q2flag')
        elif flag == 'q1flag':
            closeOthers('q1flag')
            if which == 0:
                ansupright.curtain(vrut.OPEN)
            elif which == 1:
                ansrdown.curtain(vrut.OPEN)
            elif which == 2:
                ansupdown.curtain(vrut.OPEN)
            elif which == 3:
                ansldown.curtain(vrut.OPEN)
            elif which == 4:
                ansnoknow.curtain(vrut.OPEN)
            else:
                print('error in q1flag')
        elif flag == 'q3flag':
            closeOthers('q3flag')
            if which == 0:
                ans3pit90f.curtain(vrut.OPEN)
            elif which == 1:
                ans3pit90b.curtain(vrut.OPEN)
            elif which == 2:
                ans3yaw180.curtain(vrut.OPEN)
            elif which == 3:
                ans3yaw90l.curtain(vrut.OPEN)
            elif which == 4:
                ans3yaw90r.curtain(vrut.OPEN)
            elif which == 5:
                ans3norot.curtain(vrut.OPEN)
            elif which == 6:
                ans3noknow.curtain(vrut.OPEN)
            else:
                print('error in q3flag')
        elif flag == 'q4flag':
            closeOthers('q4flag')
            if which == 0:
                ansr0.curtain(vrut.OPEN)
            elif which == 1:
                ansr90l.curtain(vrut.OPEN)
            elif which == 2:
                ansr90r.curtain(vrut.OPEN)
            elif which == 3:
                ansr180.curtain(vrut.OPEN)
            elif which == 4:
                ans4noknow.curtain(vrut.OPEN)
            else:
                print('error in q4flag')
        elif flag == 'q5flag':
            closeOthers('q5flag')
            if which == 0:
                ansexp.curtain(vrut.OPEN)
            elif which == 1:
                anseva.curtain(vrut.OPEN)
            elif which == 2:
                ansstore.curtain(vrut.OPEN)
            elif which == 3:
                anshealth.curtain(vrut.OPEN)
            elif which == 4:
                anshab.curtain(vrut.OPEN)
            elif which == 5:
                anscentr.curtain(vrut.OPEN)
            elif which == 6:
                anscont.curtain(vrut.OPEN)
            elif which == 7:
                ansclose.curtain(vrut.OPEN)
            elif which == 8:
                ans5noknow.curtain(vrut.OPEN)
            else:
                print('error in q5flag')
        elif flag == 'q6flag':
            closeOthers('q6flag')
            print('which', which)
            if which == 0:
                ans2exp.curtain(vrut.OPEN)
            elif which == 1:
                ans2eva.curtain(vrut.OPEN)
            elif which == 2:
                ans2store.curtain(vrut.OPEN)
            elif which == 3:
                ans2health.curtain(vrut.OPEN)
            elif which == 4:
                ans2hab.curtain(vrut.OPEN)
            elif which == 5:
                ans2centr.curtain(vrut.OPEN)
            elif which == 6:
                ans2cont.curtain(vrut.OPEN)
            elif which == 7:
                ans2close.curtain(vrut.OPEN)
            elif which == 8:
                ans6noknow.curtain(vrut.OPEN)
        elif flag == 'q7flag':
            closeOthers('q7flag')
            if which == 0:
                ans3exp.curtain(vrut.OPEN)
            elif which == 1:
                ans3eva.curtain(vrut.OPEN)
            elif which == 2:
                ans3store.curtain(vrut.OPEN)
            elif which == 3:
                ans3health.curtain(vrut.OPEN)
            elif which == 4:
                ans3hab.curtain(vrut.OPEN)
            elif which == 5:
                ans3centr.curtain(vrut.OPEN)
            elif which == 6:
                ans3cont.curtain(vrut.OPEN)
            elif which == 7:
                ans3close.curtain(vrut.OPEN)
            elif which == 8:
                ans7noknow.curtain(vrut.OPEN)
            else:
                print('error in q7flag')
        elif flag == 'q8flag':
            closeOthers('q8flag')
            if which == 0:
                ans4exp.curtain(vrut.OPEN)
            elif which == 1:
                ans4eva.curtain(vrut.OPEN)
            elif which == 2:
                ans4store.curtain(vrut.OPEN)
            elif which == 3:
                ans4health.curtain(vrut.OPEN)
            elif which == 4:
                ans4hab.curtain(vrut.OPEN)
            elif which == 5:
                ans4centr.curtain(vrut.OPEN)
            elif which == 6:
                ans4cont.curtain(vrut.OPEN)
            elif which == 7:
                ans4close.curtain(vrut.OPEN)
            elif which == 8:
                ans8noknow.curtain(vrut.OPEN)
            else:
                print('error in q8flag')

    def closeEverything():
        global set1flag
        firstQuestion.curtain(vrut.CLOSE)
    secondQuestion.curtain(vrut.CLOSE)
    line.curtain(vrut.CLOSE)
    if set1flag == on:
        ansblue.curtain(vrut.CLOSE)
        ansyellw.curtain(vrut.CLOSE)
        ansupright.curtain(vrut.CLOSE)
        ansrdown.curtain(vrut.CLOSE)
        ansupdown.curtain(vrut.CLOSE)
        ansldown.curtain(vrut.CLOSE)
        ansnoknow.curtain(vrut.CLOSE)
        ans3pit90f.curtain(vrut.CLOSE)
        ans3pit90b.curtain(vrut.CLOSE)
        ans3yaw180.curtain(vrut.CLOSE)
        ans3yaw90l.curtain(vrut.CLOSE)
        ans3yaw90r.curtain(vrut.CLOSE)
        ans3norot.curtain(vrut.CLOSE)
        ans3noknow.curtain(vrut.CLOSE)
        ansr180.curtain(vrut.CLOSE)
        ansr90l.curtain(vrut.CLOSE)
        ansr90r.curtain(vrut.CLOSE)
        ansr0.curtain(vrut.CLOSE)
        ans4noknow.curtain(vrut.CLOSE)
    elif set1flag == off:
        anshealth.curtain(vrut.CLOSE)
        anshab.curtain(vrut.CLOSE)
        ansexp.curtain(vrut.CLOSE)
        anseva.curtain(vrut.CLOSE)
        anscont.curtain(vrut.CLOSE)
        ansclose.curtain(vrut.CLOSE)
        ansstore.curtain(vrut.CLOSE)
        anscentr.curtain(vrut.CLOSE)
        ans5noknow.curtain(vrut.CLOSE)
        ans2health.curtain(vrut.CLOSE)
        ans2hab.curtain(vrut.CLOSE)
        ans2exp.curtain(vrut.CLOSE)
        ans2eva.curtain(vrut.CLOSE)
        ans2cont.curtain(vrut.CLOSE)
        ans2close.curtain(vrut.CLOSE)
        ans2store.curtain(vrut.CLOSE)
        ans2centr.curtain(vrut.CLOSE)
        ans6noknow.curtain(vrut.CLOSE)
        ans3health.curtain(vrut.CLOSE)
        ans3hab.curtain(vrut.CLOSE)
        ans3exp.curtain(vrut.CLOSE)
        ans3eva.curtain(vrut.CLOSE)
        ans3cont.curtain(vrut.CLOSE)
        ans3close.curtain(vrut.CLOSE)
        ans3store.curtain(vrut.CLOSE)
        ans3centr.curtain(vrut.CLOSE)
        ans7noknow.curtain(vrut.CLOSE)
        ans4health.curtain(vrut.CLOSE)
        ans4hab.curtain(vrut.CLOSE)
        ans4exp.curtain(vrut.CLOSE)
        ans4eva.curtain(vrut.CLOSE)
        ans4cont.curtain(vrut.CLOSE)
        ans4close.curtain(vrut.CLOSE)
        ans4store.curtain(vrut.CLOSE)
        ans4centr.curtain(vrut.CLOSE)
        ans8noknow.curtain(vrut.CLOSE)

def myKeyboard(key):
    global count,begintime
    if key == ' ':
        reminder.curtain(vrut.CLOSE)
        station.curtain(vrut.OPEN)
        vrut.starttimer(start_trial,0.001)
        begintime = time.time()
        out = str(time.time()-begintime) + '\t' + str(count) +
'\tcount\n'
        keys.write(out)
        keys.flush()
    elif key == '1':
        module = trialset[count][1]
        showTarget(module,1)
    elif key == 'r':
        tracker.command(6)  # RESET1    ... see multi.py in AndyL
        tracker.reset()

def mytimer(timernum):
    global q1flag,q2flag,q3flag,q4flag,q5flag,q6flag,q7flag,q8flag
    global
q1index,q2index,q3index,q4index,q5index,q6index,q7index,q8index
    global count,set1flag,startpoint,begintime
    if timernum == start_trial:
        arrow.curtain(vrut.OPEN)
        set1flag = on
        ori = trialset[count][0]
        vrut.reset(vrut.BODY_ORI)
        vrut.reset(vrut.HEAD_POS)
        vrut.rotate(vrut.BODY_ORI,0,0,posture[ori])
        module = trialset[count][1]
        showTarget(module,2)
        q1flag = on
        out = str(ori) + '\t' + str(module) + '\t'
        file.write(out)
        file.flush()
        startpoint = time.time()
        vrut.starttimer(waitENTER,rate)

    elif timernum == closeTarget:
        closeAll()

    elif timernum == waitENTER:
        if sid.buttons() == sid.BUTTON9:
            arrow.curtain(vrut.CLOSE)
            data = tracker.get()
            elapstime = time.time() - startpoint
            out                                                    =
str(data[3])+'\t'+str(data[4])+'\t'+str(data[5])+'\t'+str(data[6]*RAD2
DEG)+'\t'+str(elapstime)+'\t'
            file.write(out)
            file.flush()
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON9)
+ '\tenter_pointing\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(showFirstQuest,1)
        else:
            vrut.starttimer(waitENTER,rate)

    elif timernum == showFirstQuest:
        firstQuestion.curtain(vrut.OPEN)
        line.curtain(vrut.OPEN)
        startpoint = time.time()
        scrollAns('q1flag',q1index)
        scrollAns('q2flag',q2index)
        scrollAns('q3flag',q3index)
        scrollAns('q4flag',q4index)
        vrut.starttimer(scroller,rate)

    elif timernum == showSecondQuest:
        secondQuestion.curtain(vrut.OPEN)
        line.curtain(vrut.OPEN)
        startpoint = time.time()
        print('2nd question',q5index,q6index,q7index,q8index)
        scrollAns('q5flag',q5index)
        scrollAns('q6flag',q6index)
        scrollAns('q7flag',q7index)
        scrollAns('q8flag',q8index)
        vrut.starttimer(scroller,rate)

    elif timernum == scroller:
        if sid.buttons() == sid.BUTTON1:
            # check for which question
            # scroll up (add)
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON1)
+ '\tscroll_up\n'
            keys.write(out)
            keys.flush()
            if q2flag == on:
                if q2index == 0:
                    q2index = 1
                else:
                    q2index = 0
                scrollAns('q2flag',q2index)
            elif q1flag == on:
                if q1index == 4:
                    q1index = 0
                else:
                    q1index = q1index + 1
                scrollAns('q1flag',q1index)
            elif q3flag == on:
                if q3index == 6:
                    q3index = 0
                else:
                    q3index = q3index + 1
                scrollAns('q3flag',q3index)
            elif q4flag == on:
                if q4index == 4:
                    q4index = 0
                else:
                    q4index = q4index + 1
                scrollAns('q4flag',q4index)
            elif q5flag == on:
                if q5index == 8:
                    q5index = 0
                else:
                    q5index = q5index + 1
                scrollAns('q5flag',q5index)
            elif q6flag == on:
                if q6index == 8:
                    q6index = 0
                else:
                    q6index = q6index + 1
                scrollAns('q6flag',q6index)
            elif q7flag == on:
                if q7index == 8:
                    q7index = 0
                else:
                    q7index = q7index + 1
                scrollAns('q7flag',q7index)
            elif q8flag == on:
                if q8index == 8:
                    q8index = 0
                else:
                    q8index = q8index + 1
                scrollAns('q8flag',q8index)
            vrut.starttimer(scroller,rate)

        elif sid.buttons() == sid.BUTTON4:
            # check for which question
            # scroll down (subtract)
            out = str(time.time()-begintime) + '\t' + str(sid.BUTTON4)
+ '\tscroll_down\n'
            keys.write(out)
            keys.flush()
            if q2flag == on:
                if q2index == 1:
                    q2index = 0
                else:
                    q2index = 1
                scrollAns('q2flag',q2index)
            elif q1flag == on:
                if q1index == 0:
                    q1index = 4
                else:
                    q1index = q1index - 1
                scrollAns('q1flag',q1index)
            elif q3flag == on:
                if q3index == 0:
                    q3index = 6
                else:
                    q3index = q3index - 1
                scrollAns('q3flag',q3index)
            elif q4flag == on:
                if q4index == 0:
                    q4index = 4
                else:
                    q4index = q4index - 1
                scrollAns('q4flag',q4index)
            elif q5flag == on:
                if q5index == 0:
                    q5index = 8
                else:
                    q5index = q5index - 1
                scrollAns('q5flag',q5index)
            elif q6flag == on:
                if q6index == 0:
                    q6index = 8
                else:
                    q6index = q6index - 1
                scrollAns('q6flag',q6index)
            elif q7flag == on:
                if q7index == 0:
                    q7index = 8
                else:
                    q7index = q7index - 1
                scrollAns('q7flag',q7index)
            elif q8flag == on:
                if q8index == 0:
                    q8index = 8
                else:
                    q8index = q8index - 1
                scrollAns('q8flag',q8index)
            vrut.starttimer(scroller,rate)

        elif sid.buttons() == 768:  # pressing sid.BUTTON9 and
sid.BUTTON10 together
            print('add a out file statement!')
            elapstime = time.time() - startpoint
            closeEverything()
            out = str(time.time()-begintime) + '\t' + str(768) +
'\tenter_hide_quest_set\n'
            keys.write(out)
            keys.flush()
            if set1flag == on:
                q1flag = on
                q2flag = off
                q3flag = off
                q4flag = off
                out                                                    =
str(q1index)+'\t'+str(q2index)+'\t'+str(q3index)+'\t'+str(q4index)+'
\t'+ str(elapstime)+'\n'
                file.write(out)
                file.flush()
            vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\EndQuest.wav')
            #set1flag = off
            line.translate(0,lay,lz)
            #vrut.starttimer(showSecondQuest,.5)
        #else:
        #   q5flag = off
        #   q6flag = off
        #   q7flag = off
        #   q8flag = off
        #   q1flag = on
        #                                     out        =
str(q5index)+'\t'+str(q6index)+'\t'+str(q7index)+'\t'+str(q8index)+'
\t'+ str(elapstime)+'\n'
        #   file.write(out)
        #   file.flush()
        #                     vrut.playsound('n:\Jessica\Thesis
SIM\Experiment\EndQuest.wav')
            print('increasing count')
            if count == 23:
               print('endprogram!!!')
               vrut.translate(vrut.HEAD_POS, 0,100,0)
               file.close()
               keys.close()
            else:
               count = count + 1
               q1index = 0
               q2index = 0
               q3index = 0
               q4index = 0
               q5index = 0
               q6index = 0
               q7index = 0
               q8index = 0
               vrut.starttimer(start_trial,0.001)

    elif sid.buttons() == sid.BUTTON9:
        # enter answer  (record)
        # increase which question counter
        if q1flag == on:
            q1flag = off
            q2flag = on
            line.translate(0,lby,lz)
            scrollAns('q2flag',q2index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question1\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q2flag == on:
            q2flag = off
            q3flag = on
            line.translate(0,lcy,lz)
            scrollAns('q3flag',q3index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question2\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q3flag == on:
            q3flag = off
            q4flag = on
            line.translate(0,ldy,lz)
            scrollAns('q4flag',q4index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question3\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q4flag == on:
            q4flag = off
            q1flag = on
            line.translate(0,lay,lz)
            scrollAns('q1flag',q1index)
            out    =    str(time.time()-begintime)    +    '\t'    +
str(sid.BUTTON9) + '\tenter_question4\n'
            keys.write(out)
            keys.flush()
            vrut.starttimer(scroller,rate)
        elif q5flag == on:
            q5flag = off
            q6flag = on
            line.translate(0,lby,lz)
            scrollAns('q6flag',q6index)
            vrut.starttimer(scroller,rate)
        elif q6flag == on:
            q6flag = off
            q7flag = on
            line.translate(0,lcy,lz)
            scrollAns('q7flag',q7index)
            vrut.starttimer(scroller,rate)
        elif q7flag == on:
            q7flag = off
            q8flag = on
            line.translate(0,ldy,lz)
            scrollAns('q8flag',q8index)
            vrut.starttimer(scroller,rate)
        elif q8flag == on:
            q8flag = off
            q5flag = on
            line.translate(0,lay,lz)
            scrollAns('q5flag',q5index)
            vrut.starttimer(scroller,rate)

    else:
        vrut.starttimer(scroller,rate)

#
*****************************************************************
**********************

# Setting up output files
exp_time = time.localtime(time.time())
timestring = time.strftime('%b%d_%H%M',exp_time)
filename                    =                    'N:\Jessica\Thesis
SIM\Data\Sim\Phase3\Phase3_SIM'+str(timestring)+'.dat'
#          filename          =                   'N:\Jessica\Thesis
SIM\Data\Cntrl\Phase3\Phase3_Cntrl'+str(timestring)+'.dat'
file = open(filename,'w')
out                                                              =
'Posture\tTarget\tQuat0\tQuat1\tQuat2\tQuat3\tTime\tQ1\tQ2\
tQ3\tQ4\tTime1\n'
file.write(out)
file.flush()
# second file out
filename2                   =                    'N:\Jessica\Thesis
SIM\Data\Sim\Phase3\Phase3_SIM'+str(timestring)+'_allkeys.dat'
#          filename2          =                  'N:\Jessica\Thesis
SIM\Data\Cntrl\Phase3\Phase3_Cntrl'+str(timestring)+'_allkeys.da
t'

keys = open(filename2,'w')

vrut.callback(vrut.KEYBOARD_EVENT, 'myKeyboard')
vrut.callback(vrut.TIMER_EVENT,'mytimer')
```

APPENDIX B: SUBJECT INSTRUCTIONS

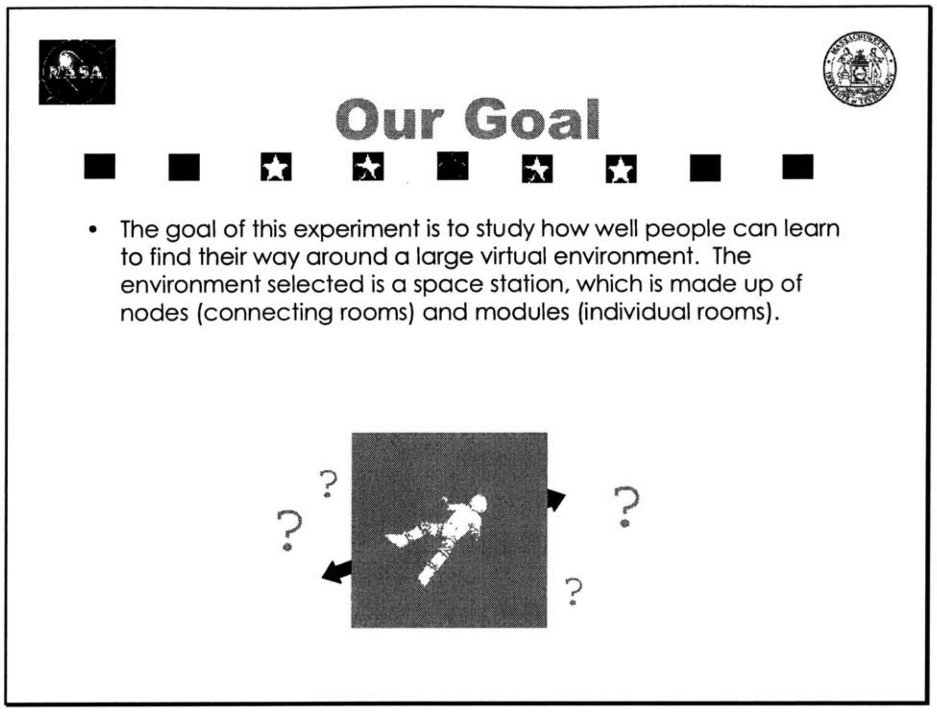
There were two sets of instructions, one for each group. Sections were read previous to every step. Most of the slides are the same for both groups except for those indicated in this appendix as pertaining to a certain group.



Learning to Navigate Through a Virtual Space Station

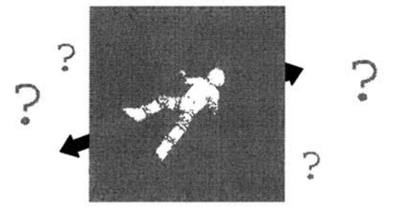
Instructions for Subjects

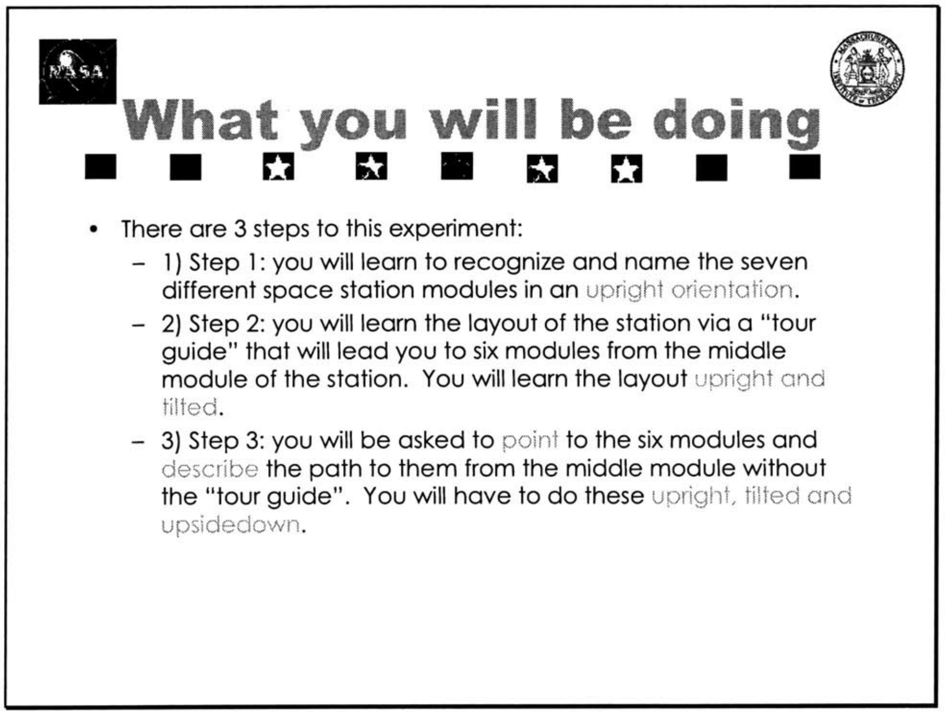
Jessica Márquez
Master's Thesis Project
Man-Vehicle Lab/Aero-Astro/MIT



Our Goal

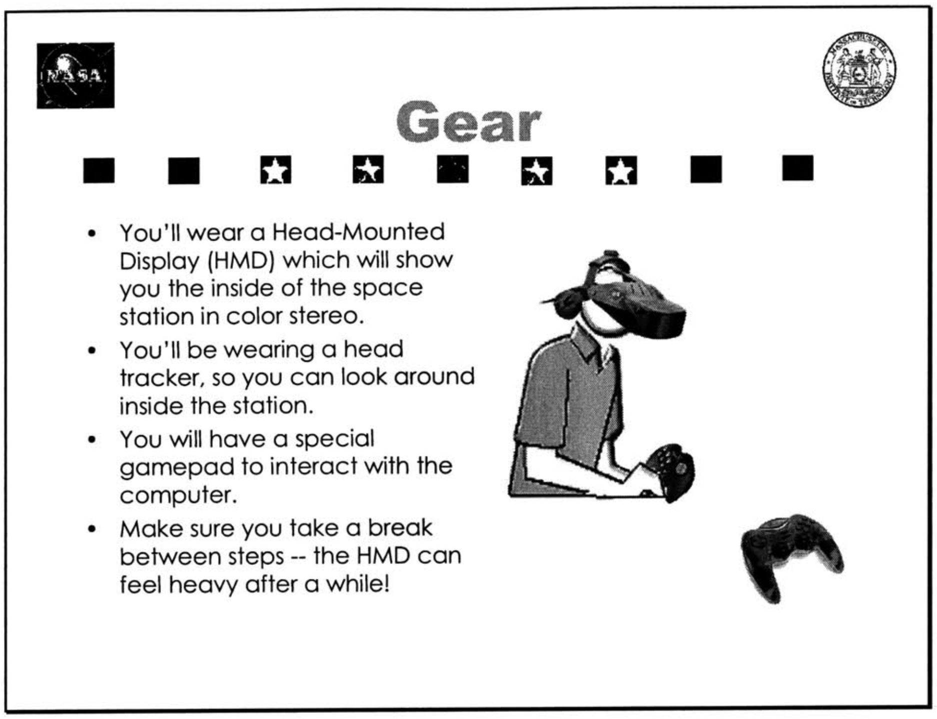
- The goal of this experiment is to study how well people can learn to find their way around a large virtual environment. The environment selected is a space station, which is made up of nodes (connecting rooms) and modules (individual rooms).





What you will be doing

- There are 3 steps to this experiment:
  - 1) Step 1: you will learn to recognize and name the seven different space station modules in an upright orientation.
  - 2) Step 2: you will learn the layout of the station via a "tour guide" that will lead you to six modules from the middle module of the station. You will learn the layout upright and tilted.
  - 3) Step 3: you will be asked to point to the six modules and describe the path to them from the middle module without the "tour guide". You will have to do these upright, tilted and upsidedown.



Gear

- You'll wear a Head-Mounted Display (HMD) which will show you the inside of the space station in color stereo.
- You'll be wearing a head tracker, so you can look around inside the station.
- You will have a special gamepad to interact with the computer.
- Make sure you take a break between steps -- the HMD can feel heavy after a while!

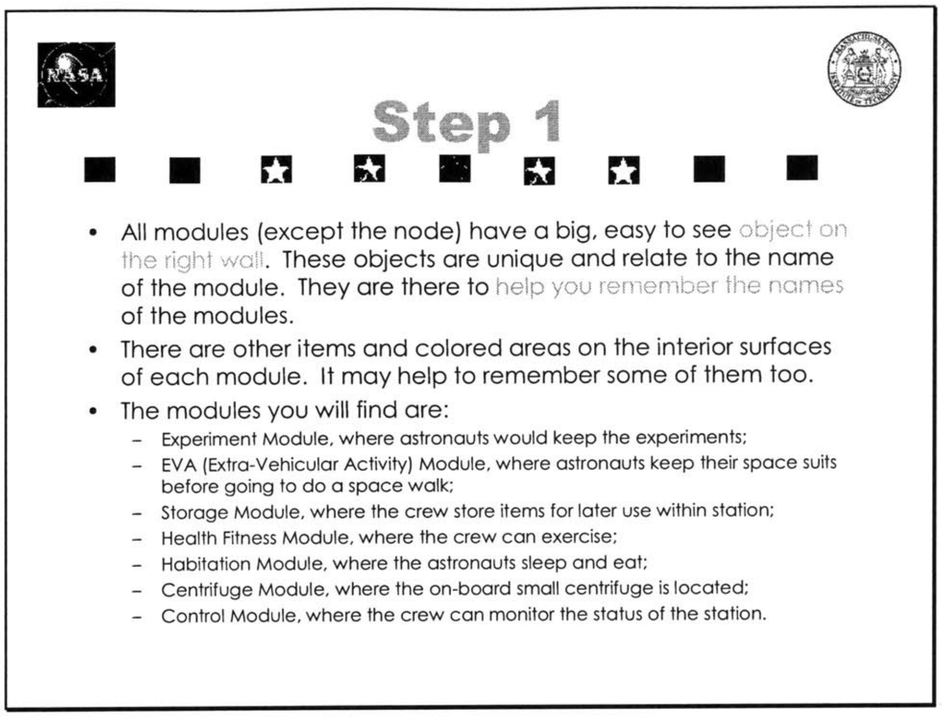
Step 1

Learning each module individually

Step 1

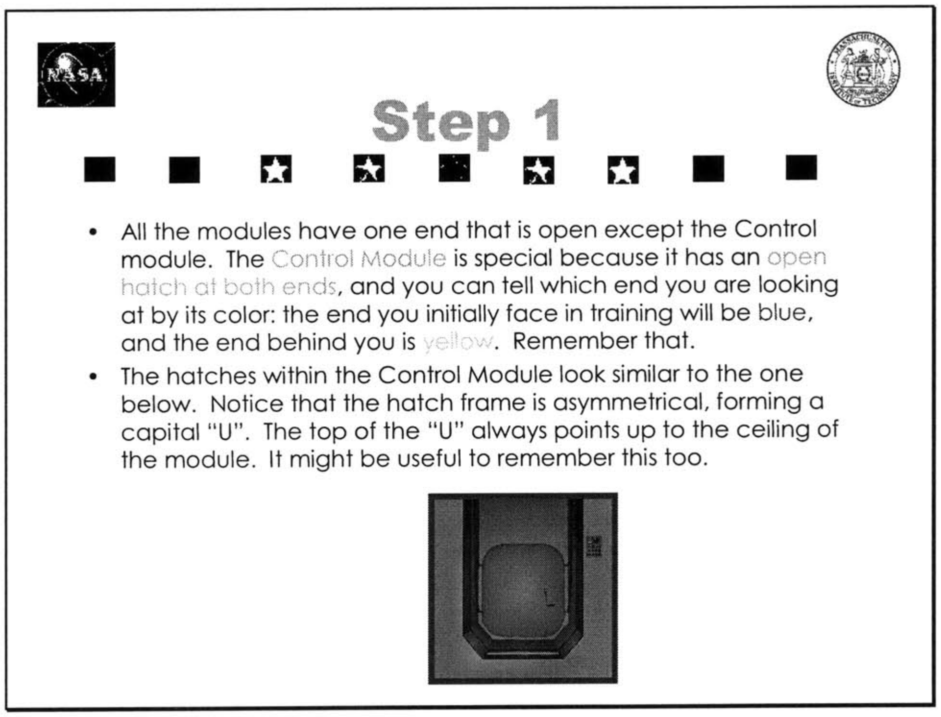
- In this step, your job is to learn to recognize each module type, and be able to tell us its name.
- You will appear in each of the seven modules and a node in an upright orientation. A node has six hatches and they connect the other modules together. Our station has two nodes, and they both look the same.
- Inside the modules or node, you can call up the name of the module at any time and move forward and backwards within that module/node. The head mounted display lets you look around.

Button to call up name (behind gamepad)
Joystick: to move forward/backward



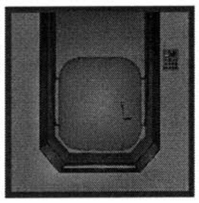
Step 1

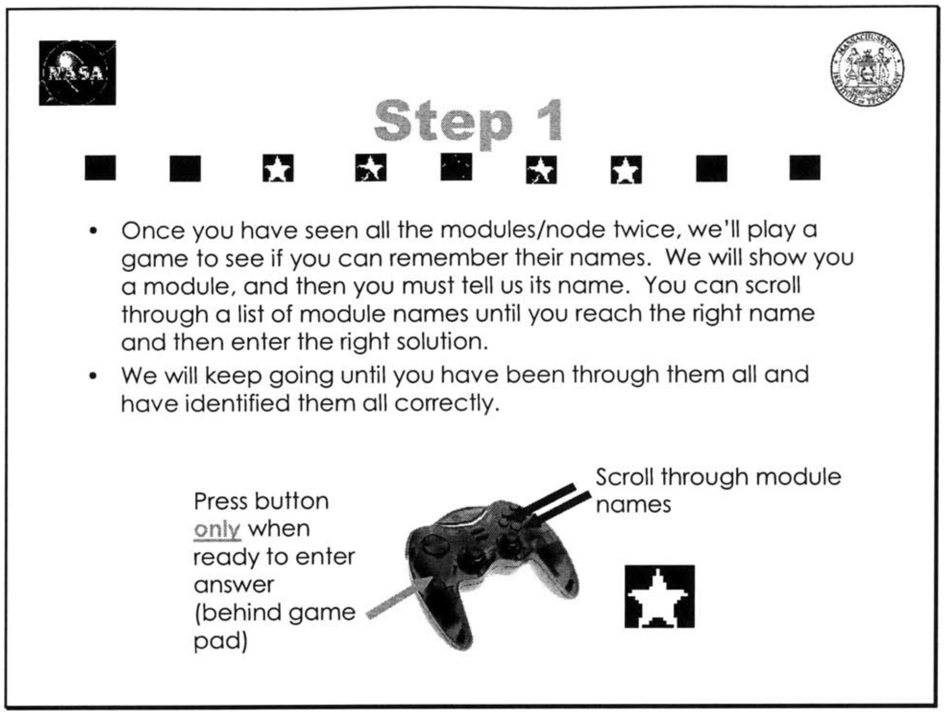
- All modules (except the node) have a big, easy to see object on the right wall. These objects are unique and relate to the name of the module. They are there to help you remember the names of the modules.
- There are other items and colored areas on the interior surfaces of each module. It may help to remember some of them too.
- The modules you will find are:
  - Experiment Module, where astronauts would keep the experiments;
  - EVA (Extra-Vehicular Activity) Module, where astronauts keep their space suits before going to do a space walk;
  - Storage Module, where the crew store items for later use within station;
  - Health Fitness Module, where the crew can exercise;
  - Habitation Module, where the astronauts sleep and eat;
  - Centrifuge Module, where the on-board small centrifuge is located;
  - Control Module, where the crew can monitor the status of the station.



Step 1

- All the modules have one end that is open except the Control module. The Control Module is special because it has an open hatch at both ends, and you can tell which end you are looking at by its color: the end you initially face in training will be blue, and the end behind you is yellow. Remember that.
- The hatches within the Control Module look similar to the one below. Notice that the hatch frame is asymmetrical, forming a capital "U". The top of the "U" always points up to the ceiling of the module. It might be useful to remember this too.



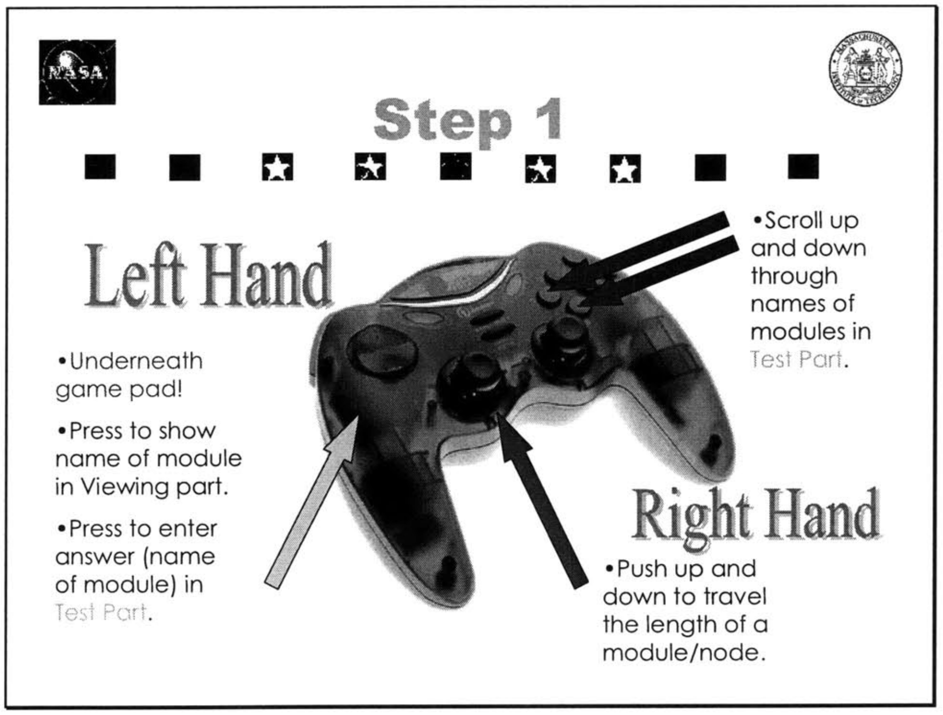


Step 1

- Once you have seen all the modules/node twice, we'll play a game to see if you can remember their names. We will show you a module, and then you must tell us its name. You can scroll through a list of module names until you reach the right name and then enter the right solution.
- We will keep going until you have been through them all and have identified them all correctly.

Press button <u>only</u> when ready to enter answer (behind game pad)

Scroll through module names



Step 1

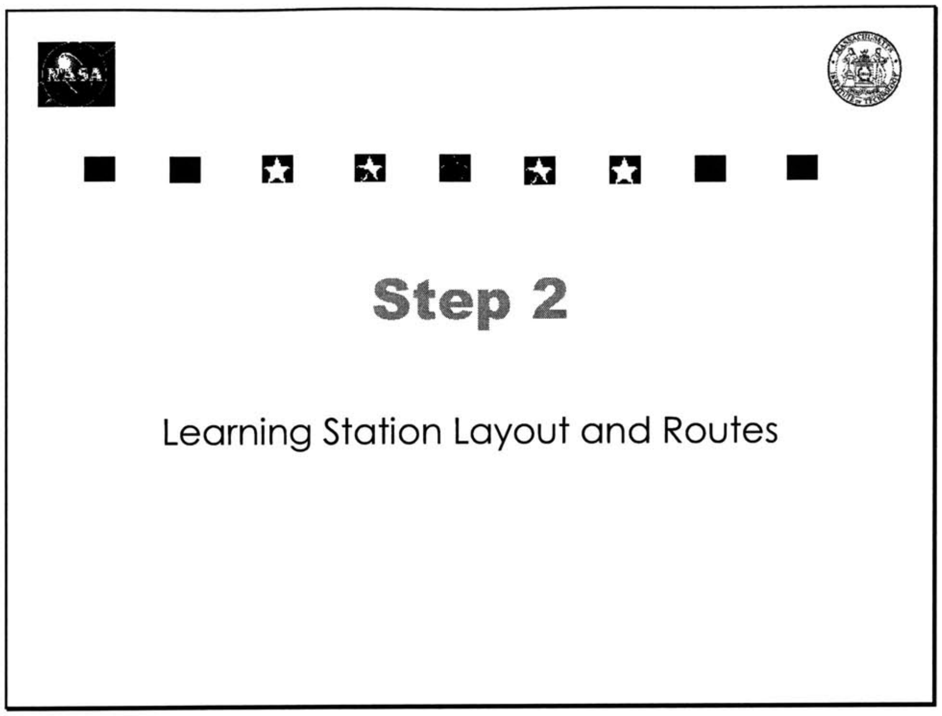
Left Hand

- Underneath game pad!
- Press to show name of module in Viewing part.
- Press to enter answer (name of module) in Test Part.

- Scroll up and down through names of modules in Test Part.

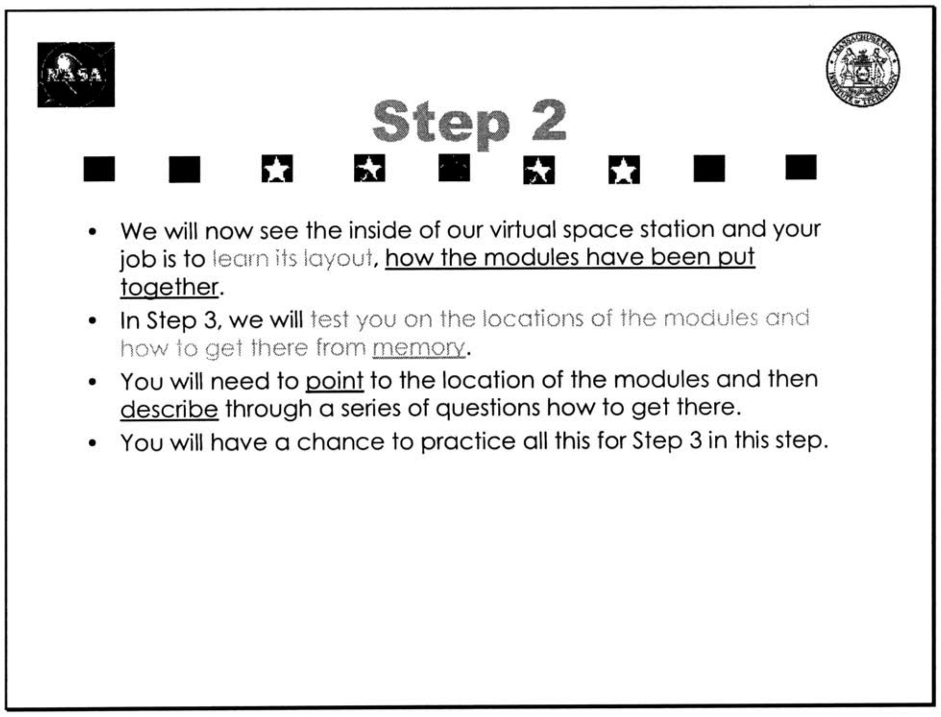
Right Hand

- Push up and down to travel the length of a module/node.



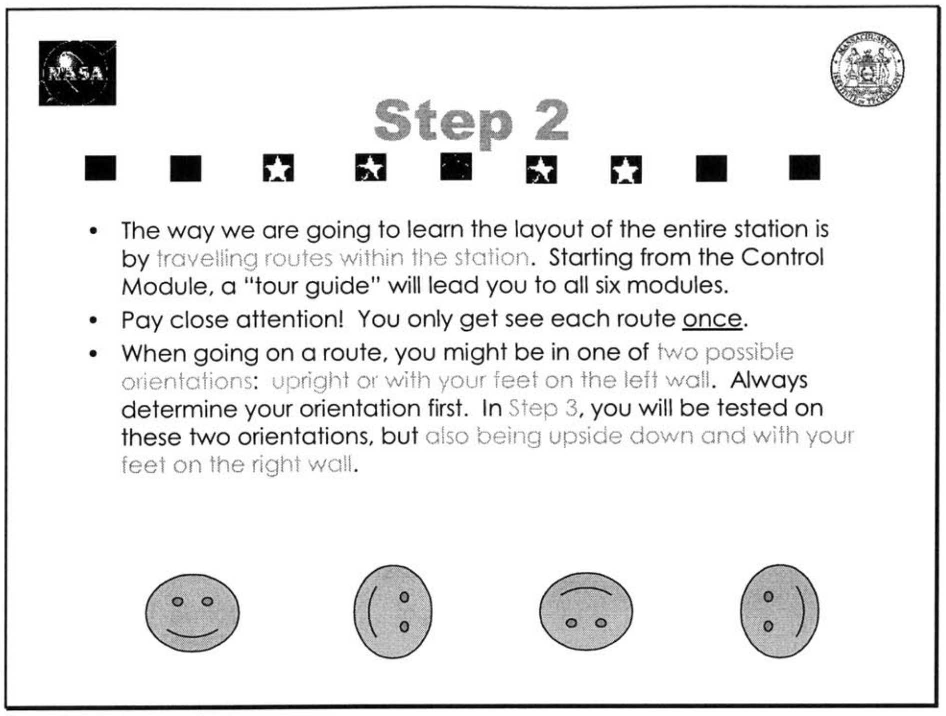
Step 2

Learning Station Layout and Routes



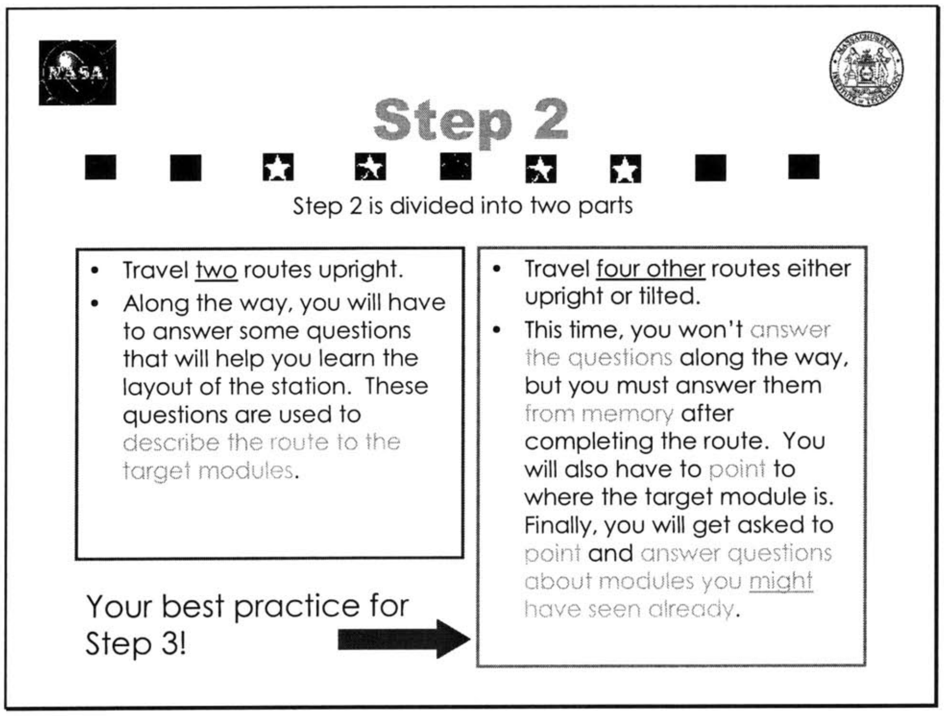
Step 2

- We will now see the inside of our virtual space station and your job is to learn its layout, how the modules have been put together.
- In Step 3, we will test you on the locations of the modules and how to get there from memory.
- You will need to point to the location of the modules and then describe through a series of questions how to get there.
- You will have a chance to practice all this for Step 3 in this step.



Step 2

- The way we are going to learn the layout of the entire station is by travelling routes within the station. Starting from the Control Module, a "tour guide" will lead you to all six modules.
- Pay close attention! You only get see each route <u>once</u>.
- When going on a route, you might be in one of two possible orientations: upright or with your feet on the left wall. Always determine your orientation first. In Step 3, you will be tested on these two orientations, but also being upside down and with your feet on the right wall.



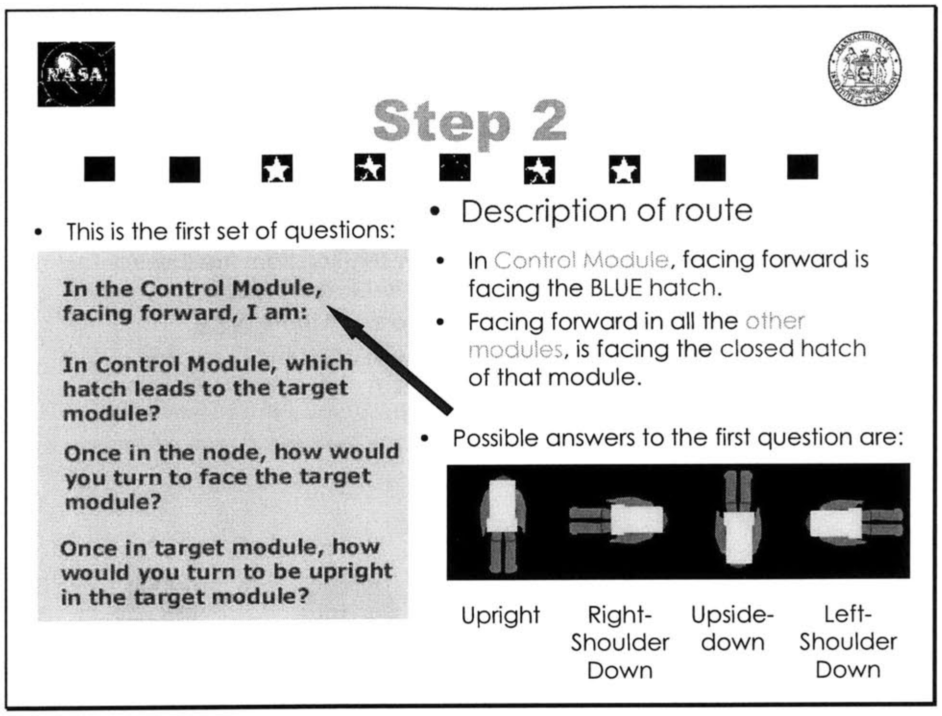
Step 2

Step 2 is divided into two parts

- Travel <u>two</u> routes upright.
- Along the way, you will have to answer some questions that will help you learn the layout of the station. These questions are used to describe the route to the target modules.

Your best practice for Step 3!

- Travel <u>four other</u> routes either upright or tilted.
- This time, you won't answer the questions along the way, but you must answer them from memory after completing the route. You will also have to point to where the target module is. Finally, you will get asked to point and answer questions about modules you <u>might</u> have seen already.



Step 2

- This is the first set of questions:

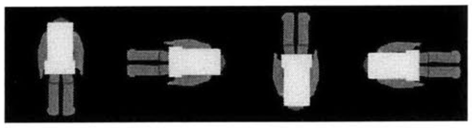
In the Control Module, facing forward, I am:

In Control Module, which hatch leads to the target module?

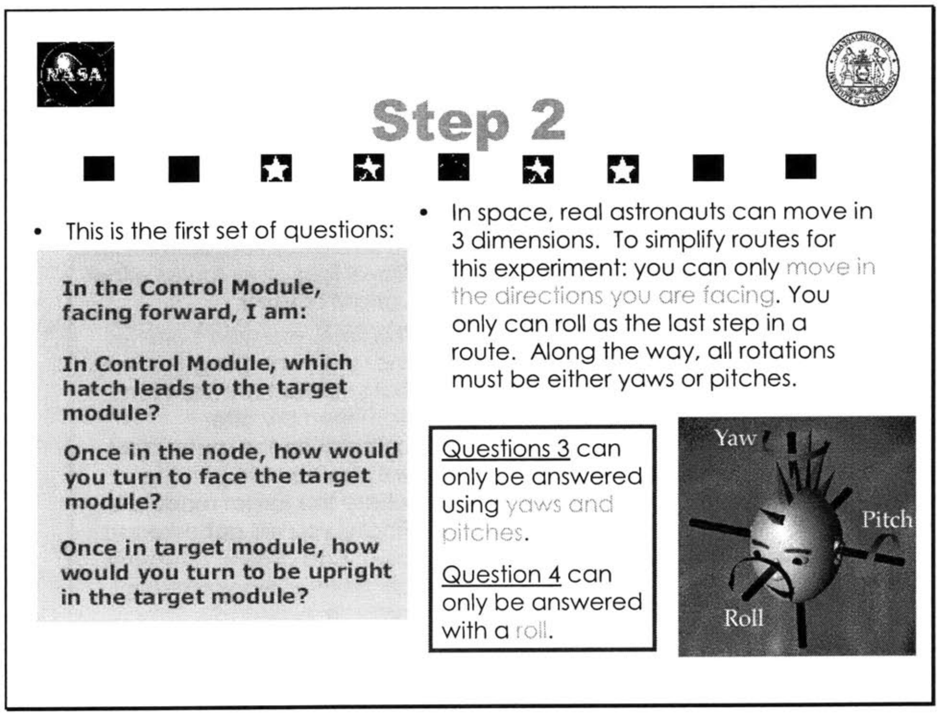
Once in the node, how would you turn to face the target module?

Once in target module, how would you turn to be upright in the target module?

- Description of route
  - In Control Module, facing forward is facing the BLUE hatch.
  - Facing forward in all the other modules, is facing the closed hatch of that module.
- Possible answers to the first question are:



Upright	Right-Shoulder Down	Upside-down	Left-Shoulder Down



Step 2

- This is the first set of questions:

In the Control Module, facing forward, I am:

In Control Module, which hatch leads to the target module?

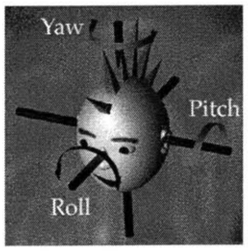
Once in the node, how would you turn to face the target module?

Once in target module, how would you turn to be upright in the target module?

- In space, real astronauts can move in 3 dimensions. To simplify routes for this experiment: you can only move in the directions you are facing. You only can roll as the last step in a route. Along the way, all rotations must be either yaws or pitches.

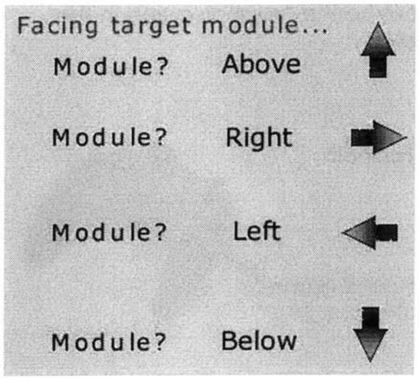
<u>Questions 3</u> can only be answered using yaws and pitches.

<u>Question 4</u> can only be answered with a roll.



Step 2

- This is the second set of questions.



Facing target module...

Module? Above

Module? Right

Module? Left

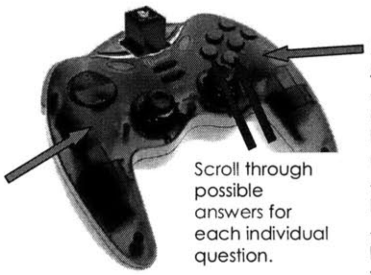
Module? Below

- These questions ask which modules are above you, to your right, left, and below you -- if you were facing a target module.
- Don't forget to look around in the nodes to be able to answer these questions.

Step 2

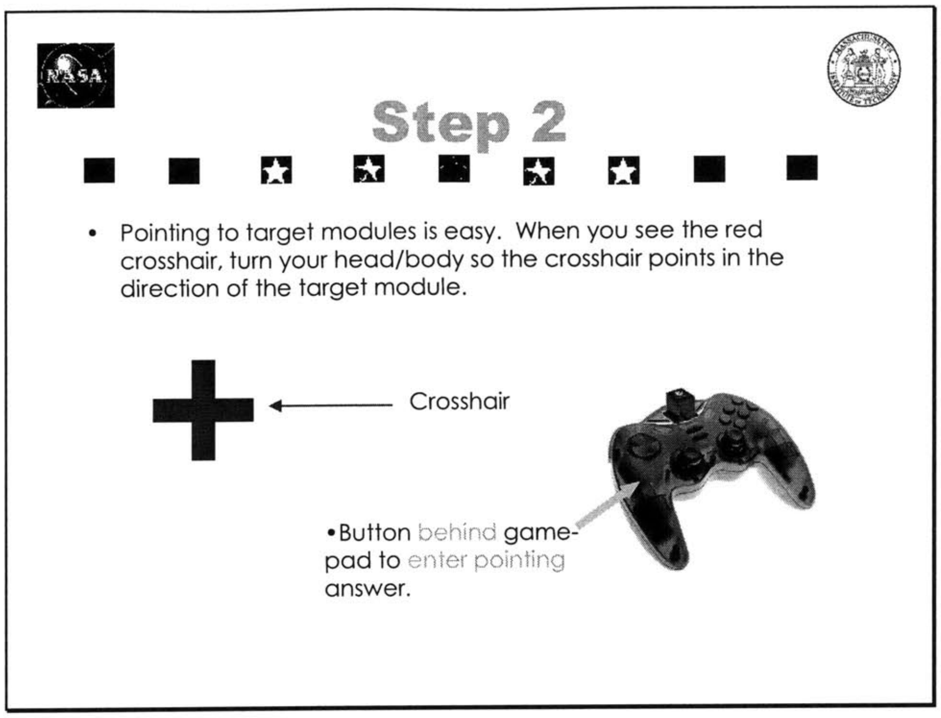
- To answer the questions, you must scroll through answers like you did in Step 1. There are two sets of questions, each containing four questions each.

Enter button (behind game-pad)
- Tells program that you have answered that individual question. Goes to the next question.
- With Hide button, lets program knows you have answered all four questions in that set.



Scroll through possible answers for each individual question.

Hide button (behind game-pad)
- Hides panel of questions (so you can look around while answering)
- Only works in first half of Step 2.
- With Enter button, lets program knows you have answered all four questions in that set.

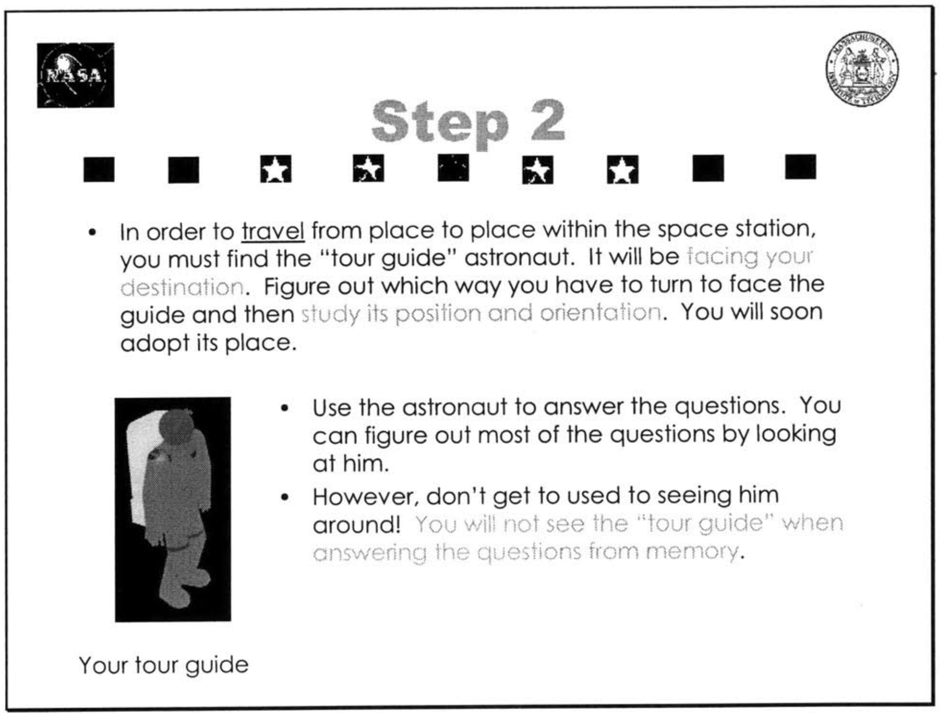


Step 2

- Pointing to target modules is easy. When you see the red crosshair, turn your head/body so the crosshair points in the direction of the target module.

- Button behind game-pad to enter pointing answer.

Control Group Slide



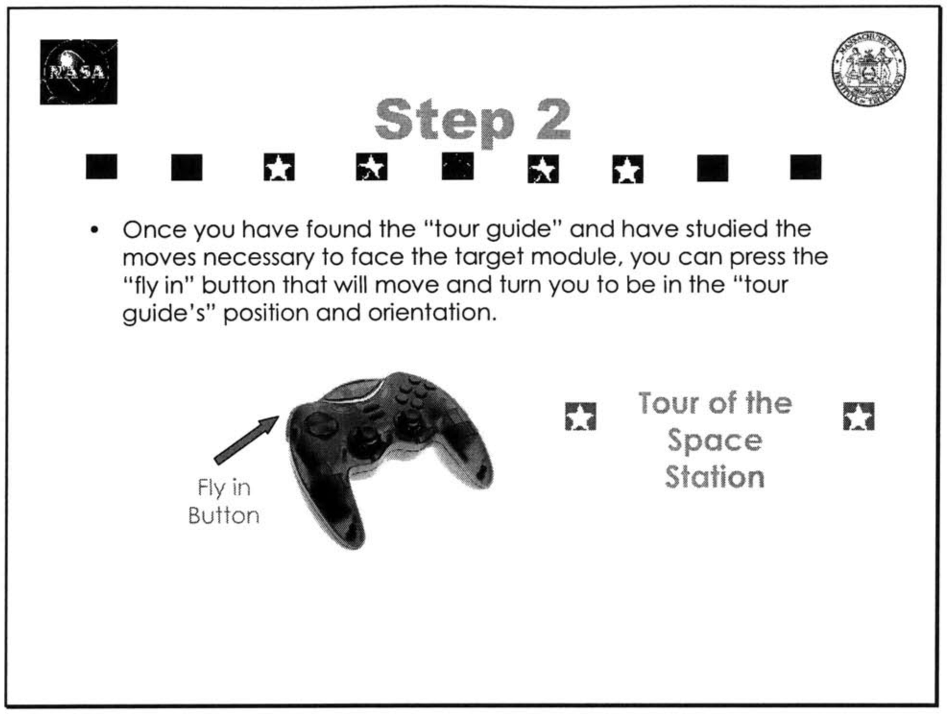
Step 2

- In order to travel from place to place within the space station, you must find the "tour guide" astronaut. It will be facing your destination. Figure out which way you have to turn to face the guide and then study its position and orientation. You will soon adopt its place.

- Use the astronaut to answer the questions. You can figure out most of the questions by looking at him.
- However, don't get to used to seeing him around! You will not see the "tour guide" when answering the questions from memory.

Your tour guide

Control Group Slide

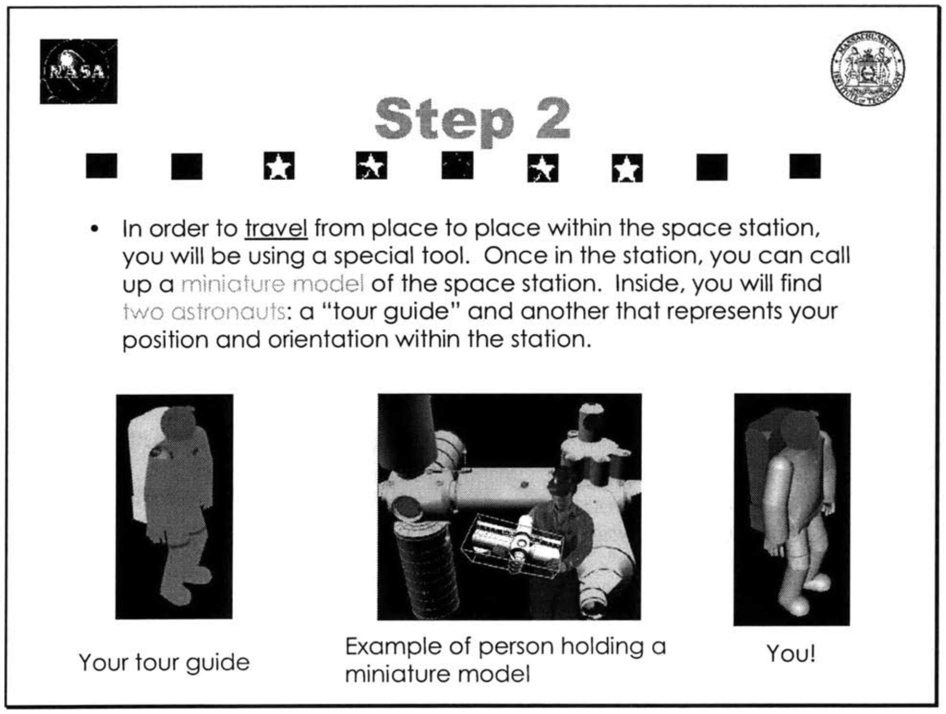


Step 2

- Once you have found the "tour guide" and have studied the moves necessary to face the target module, you can press the "fly in" button that will move and turn you to be in the "tour guide's" position and orientation.

Fly in Button
Tour of the Space Station

SIM Group Slide

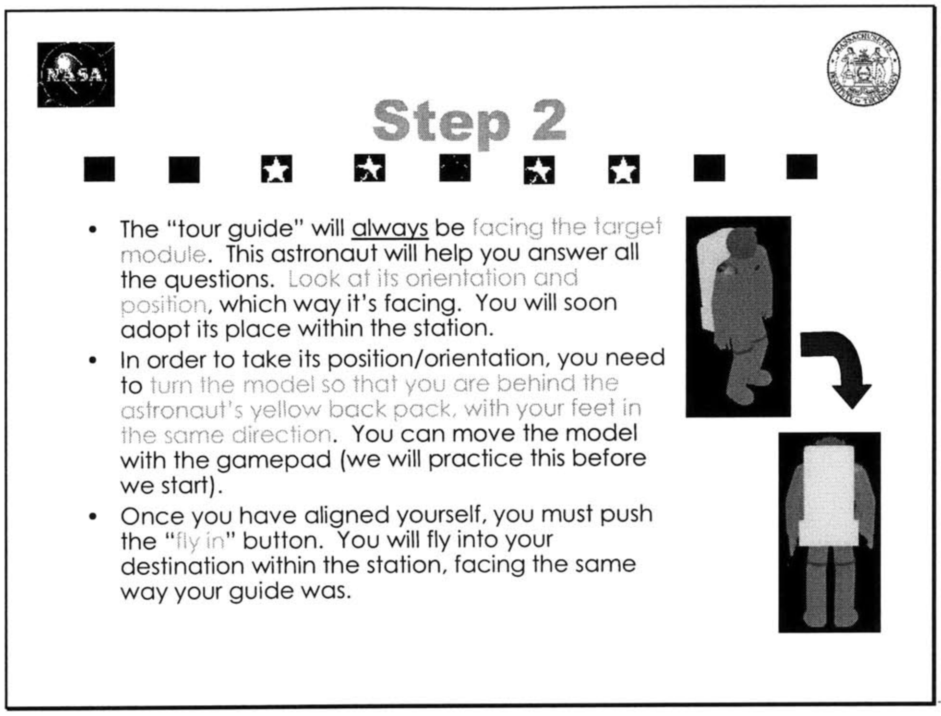


Step 2

- In order to travel from place to place within the space station, you will be using a special tool. Once in the station, you can call up a miniature model of the space station. Inside, you will find two astronauts: a "tour guide" and another that represents your position and orientation within the station.

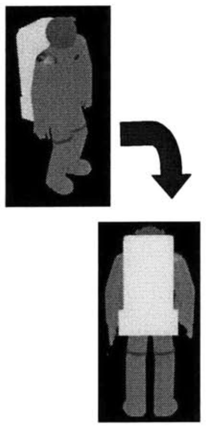
Your tour guide
Example of person holding a miniature model
You!

SIM Group Slide

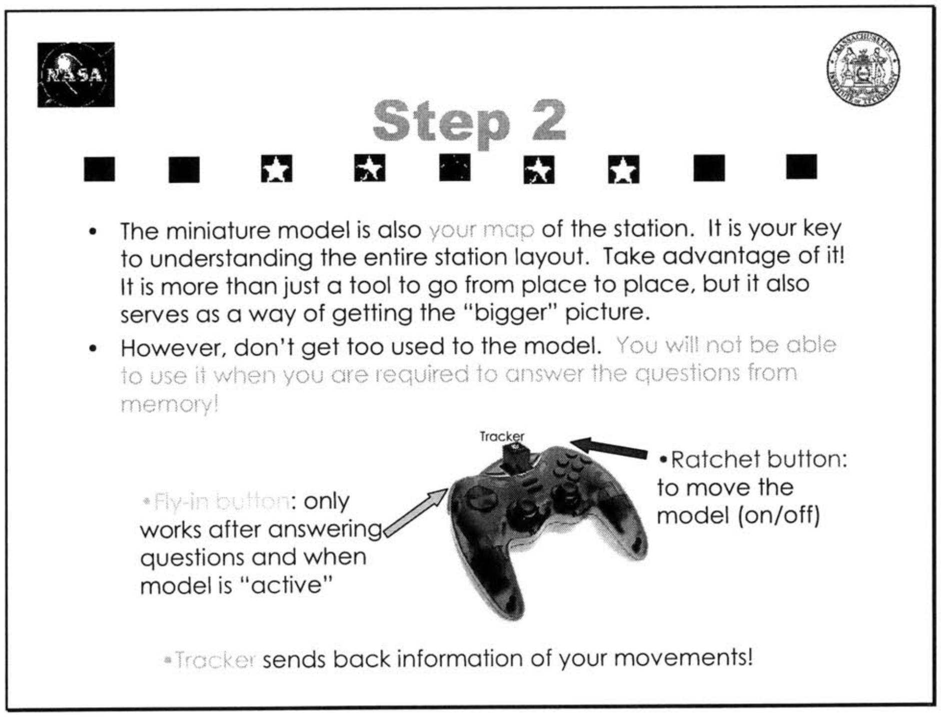


Step 2

- The "tour guide" will <u>always</u> be facing the target module. This astronaut will help you answer all the questions. Look at its orientation and position, which way it's facing. You will soon adopt its place within the station.
- In order to take its position/orientation, you need to turn the model so that you are behind the astronaut's yellow back pack, with your feet in the same direction. You can move the model with the gamepad (we will practice this before we start).
- Once you have aligned yourself, you must push the "fly in" button. You will fly into your destination within the station, facing the same way your guide was.

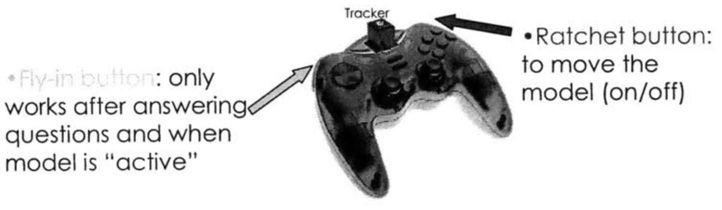


SIM Group Slide



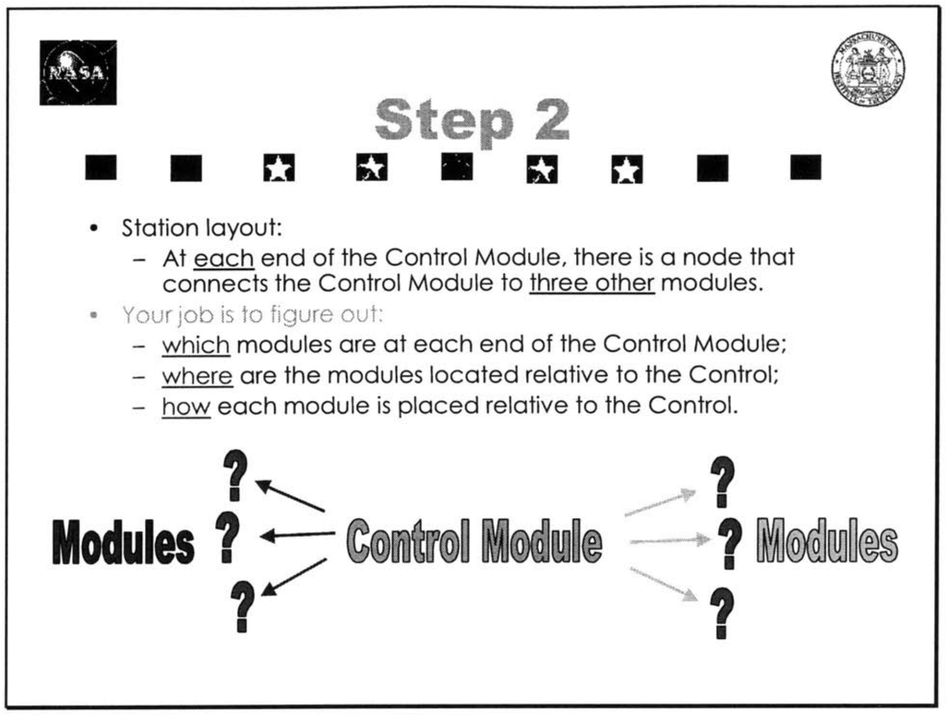
Step 2

- The miniature model is also your map of the station. It is your key to understanding the entire station layout. Take advantage of it! It is more than just a tool to go from place to place, but it also serves as a way of getting the "bigger" picture.
- However, don't get too used to the model. You will not be able to use it when you are required to answer the questions from memory!


Tracker
•Ratchet button: to move the model (on/off)
•Fly-in button: only works after answering questions and when model is "active"

•Tracker sends back information of your movements!

Both Groups Slide

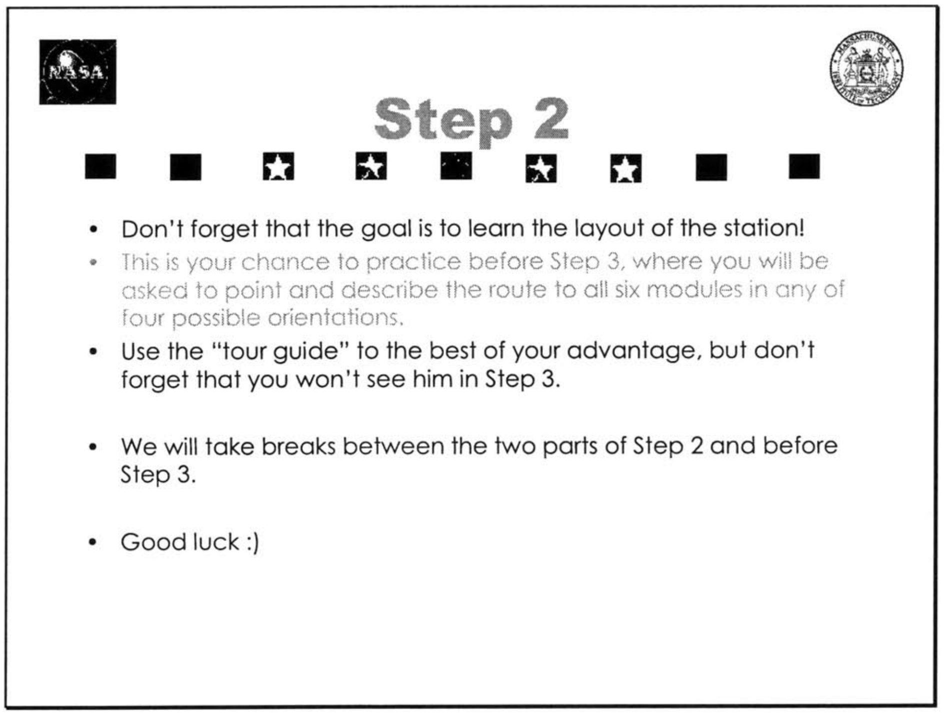


Step 2

- Station layout:
  - At each end of the Control Module, there is a node that connects the Control Module to three other modules.
- Your job is to figure out:
  - which modules are at each end of the Control Module;
  - where are the modules located relative to the Control;
  - how each module is placed relative to the Control.

Modules
?
?
?
Control Module
?
?
?
Modules

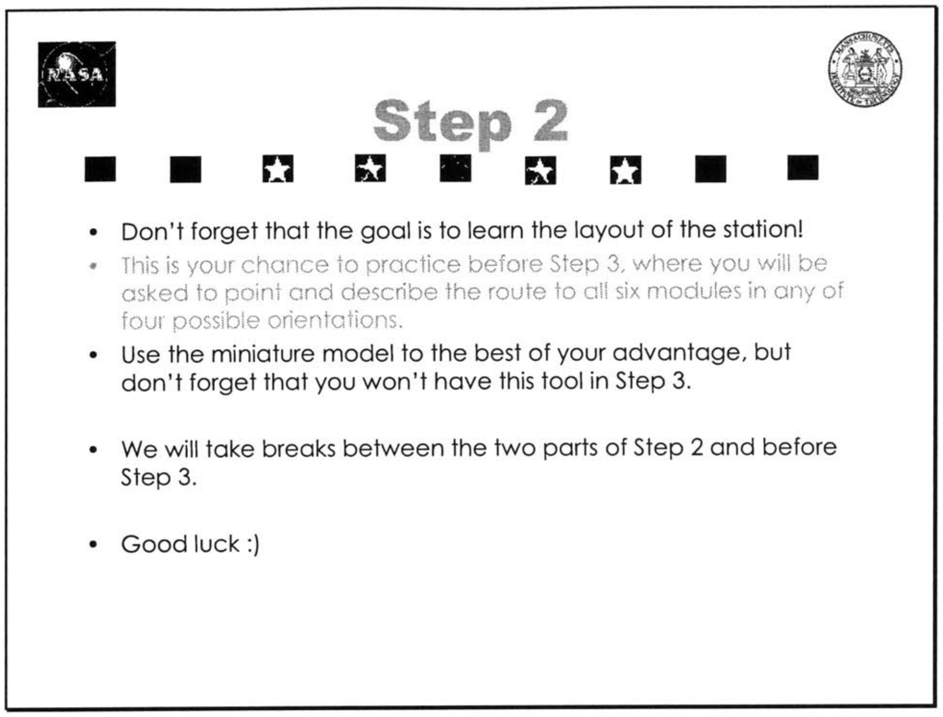
Control Group Slide



Step 2

- Don't forget that the goal is to learn the layout of the station!
- This is your chance to practice before Step 3, where you will be asked to point and describe the route to all six modules in any of four possible orientations.
- Use the "tour guide" to the best of your advantage, but don't forget that you won't see him in Step 3.
- We will take breaks between the two parts of Step 2 and before Step 3.
- Good luck :)

SIM Group Slide

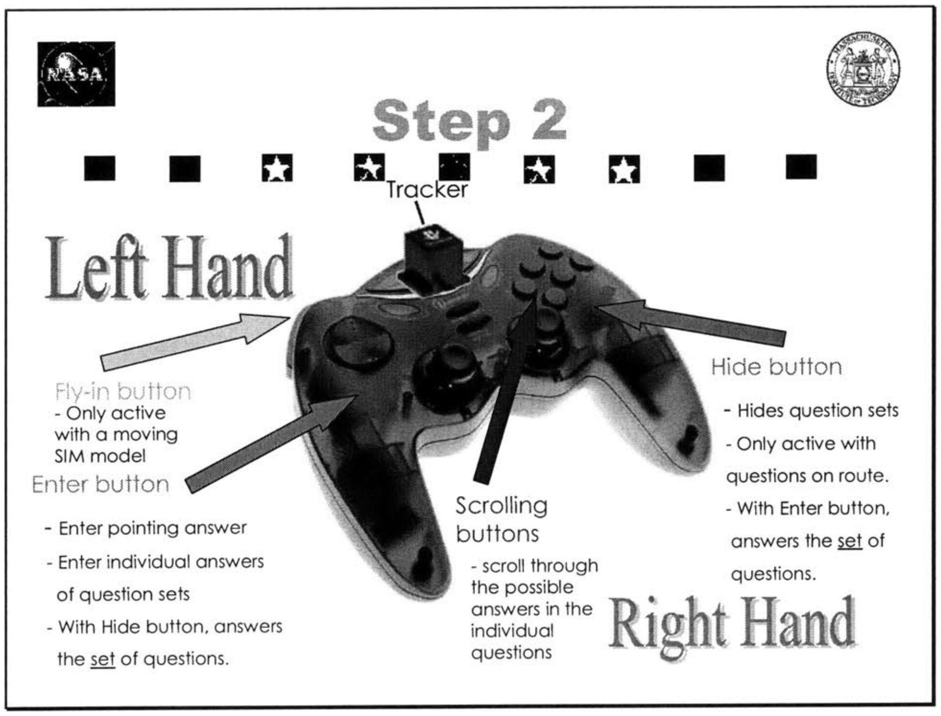


Step 2

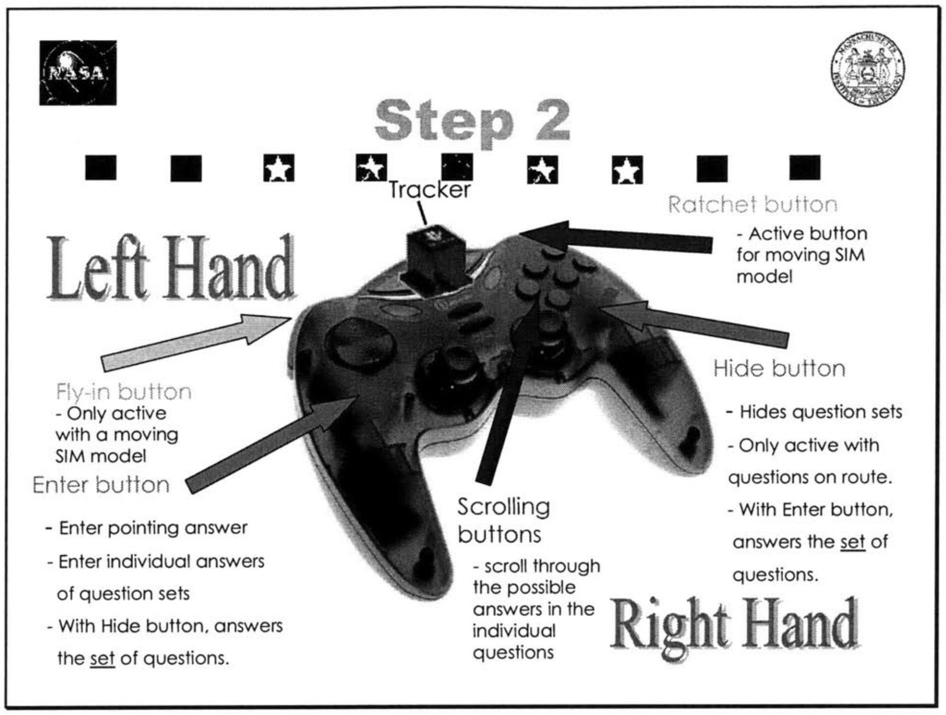
- Don't forget that the goal is to learn the layout of the station!
- This is your chance to practice before Step 3, where you will be asked to point and describe the route to all six modules in any of four possible orientations.
- Use the miniature model to the best of your advantage, but don't forget that you won't have this tool in Step 3.
- We will take breaks between the two parts of Step 2 and before Step 3.
- Good luck :)

Control Group Slide

Step 2


Tracker
Left Hand
Fly-in button
- Only active with a moving SIM model
Enter button
- Enter pointing answer
- Enter individual answers of question sets
- With Hide button, answers the set of questions.
Scrolling buttons
- scroll through the possible answers in the individual questions
Hide button
- Hides question sets
- Only active with questions on route.
- With Enter button, answers the set of questions.
Right Hand

SIM Group Slide



Step 2

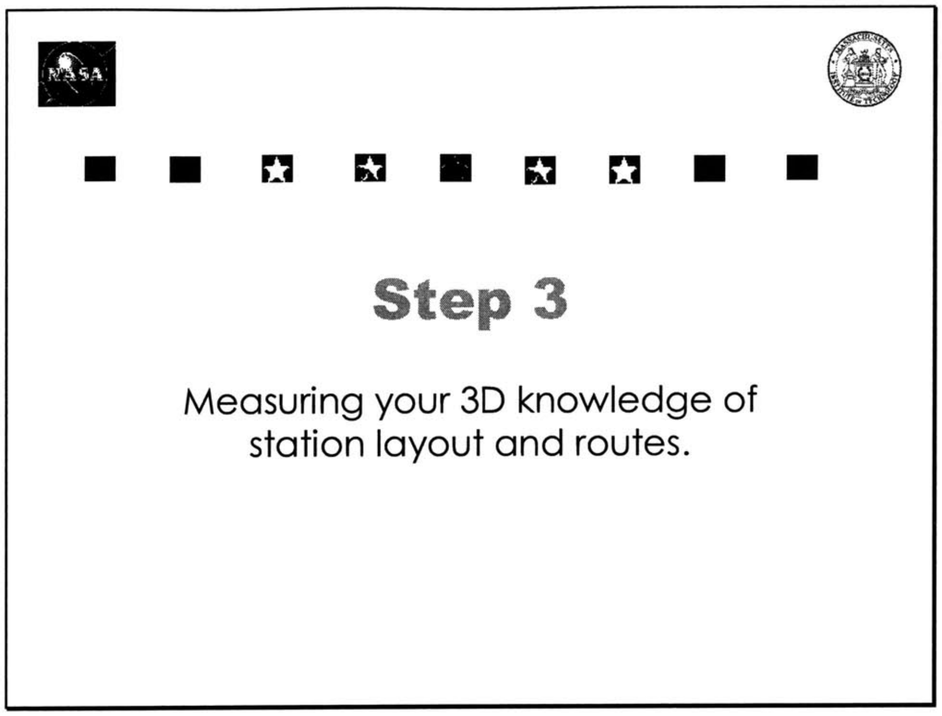
Ratchet button
- Active button for moving SIM model

Fly-in button
- Only active with a moving SIM model

Enter button
- Enter pointing answer
- Enter individual answers of question sets
- With Hide button, answers the set of questions.

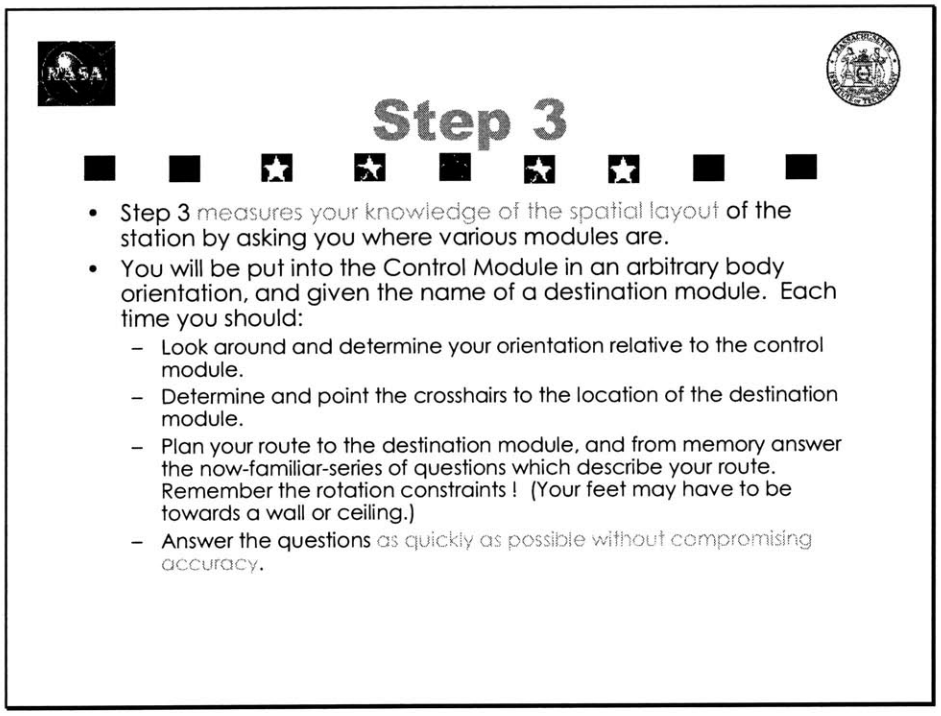
Scrolling buttons
- scroll through the possible answers in the individual questions

Hide button
- Hides question sets
- Only active with questions on route.
- With Enter button, answers the set of questions.



Step 3

Measuring your 3D knowledge of station layout and routes.

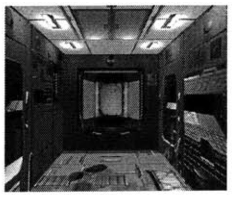


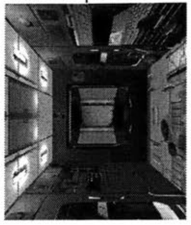
Step 3

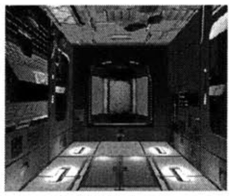
- Step 3 measures your knowledge of the spatial layout of the station by asking you where various modules are.
- You will be put into the Control Module in an arbitrary body orientation, and given the name of a destination module. Each time you should:
  - Look around and determine your orientation relative to the control module.
  - Determine and point the crosshairs to the location of the destination module.
  - Plan your route to the destination module, and from memory answer the now-familiar-series of questions which describe your route. Remember the rotation constraints ! (Your feet may have to be towards a wall or ceiling.)
  - Answer the questions as quickly as possible without compromising accuracy.

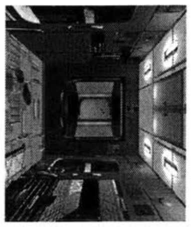
Step 3

- In Step 3 you will be placed:


Upright


Right Shoulder Down

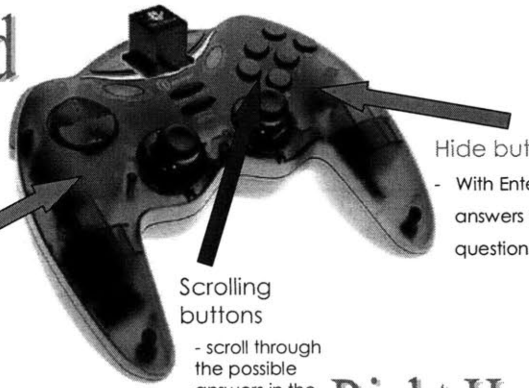

Upsidedown


Left Shoulder Down

- If you feel you have answered the pointing or route questions wrong, just keep answering with what you think is correct, regardless of your past errors.
- Good Luck! ☺

Step 3

Left Hand



Hide button
- With Enter button, answers the set of questions.

Enter button
- Enter pointing answer
- Enter individual answers of question sets
- With Hide button, answers the set of questions.

Scrolling buttons
- scroll through the possible answers in the individual questions

Right Hand

APPENDIX C: PRE-QUESTIONNAIRES

Subject Background

Name:________________
Date:_________________
Subject Number:______

Age:_______________ **Gender:** Female Male

Height & Weight:____

Occupation and field:____

Do you have a degree? If so, in what field? _______________________________

Are you: Left handed or Right handed

Do you play sports?

If so, which ones?____
… how often? (hrs/week)____

Do you play video/computer games?
If so, which ones and how often? (hrs/week)

____

____

Please describe any experience you have had with Virtual Reality systems.

____

____

____

____

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
MAN VEHICLE LABORATORY
DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS
MIT COUHES Protocol 2673

Spacecraft in Miniature Evaluation

I have been asked to participate as a subject in an evaluation of a "Spacecraft in Miniature" virtual reality display system, which tests my ability to remain spatially oriented while moving about in three dimensions in simulated spacecraft. I understand my participation is voluntary and that I am free to withdraw my consent and to discontinue participation in this study at any time without prejudice. I will be paid for my participation in a pro-rated manner at $10/hour. I understand that the investigator requests me to come in for as many of two sessions, each up to two hours in length. In the first session, I will be asked to complete a questionnaire related to my medical history, and paper-and-pencil object recognition tests. In the experiments, I will be asked to perform different types of navigation task in a "virtual" (visually simulated) environment. All experiments will be conducted while I am sitting upright in a chair, looking at a desktop computer display, or while wearing a head mounted stereoscopic video display. I control the experiment using a keyboard and cyberwand. I understand that all data developed from my participation in this study will be coded and kept confidential and that my identity will remain anonymous. The experimenter has offered to answer any questions I have concerning the procedures.

I understand that some of the visual scenes may be disorienting, and so there is a possibility I may experience some malaise, headache, nausea or other symptoms of motion sickness. If I experience unacceptable symptoms, I am free to close my eyes, ask for a break, or withdraw entirely from the experiment at any time without prejudice. If I experience any significant aftereffects, I will report them to the experimenter, and should I experience any difficulties with orientation, I will not operate an automobile.

In the unlikely event of physical injury resulting from participation in this research, I understand that medical treatment will be available from the MIT Medical Department, including first aid emergency treatment and follow-up care as needed, and that my insurance carrier may be billed for the cost of such treatment. However, no compensation can be provided for medical care apart from the foregoing. I also understand that making such medical treatment available, or providing it, does not imply that such injury is the investigator's fault. I also understand that by participating in this study I am not waiving any of my legal rights.

I understand that I may also contact the Chairman of the Committee on the Use of Humans as Experimental Subjects, MIT 253-6787, if I feel I have been treated unfairly as a subject. Further information may be obtained by calling the Institute's Insurance and Legal Affairs Office at 253-2822.

I volunteer to participate in the experiment described above.

Subject ____Date_________________

Experimenter____Date_________________

Virtual Environment Questionnaire

Do you have medical conditions that would be aggravated if you became motion sick? (yes,no) ***If you said "yes," you should not be a subject for this experiment and you should stop right now. Otherwise, please continue.***
Have you ever experienced dizzy spells? (yes,no) *If yes, can you please describe these experiences?*
Or ... motion sickness? (yes,no) *If yes, can you please explain some of your experiences?*
What is you dominant eye? (left,right) *To find your dominant eye, hold your index finger up about 10 inches from your eyes and close each eye one at a time. If you close one eye and your finger seems to move, the closed eye is dominant.*
Do you have normal peripheral vision? (yes,no) Do you have normal depth perception? (yes,no) Do you need corrective lenses? (yes,no)

Check all that apply. I have...	
☐ astigmatism ☐ near sightedness ☐ far sightedness ☐ color-blindness color(s):	☐ dyslexia type(s): ☐ blind spots where: ☐ phoria ☐ wall eye ☐ strabismus
Do you have any hearing loss? (yes,no) *If yes, please explain how you have/are lost/losing your hearing.*	
Do you have any balance problems? (yes,no) *If yes, please describe the nature of your balance problem(s)?*	
Do you have a history of chronic ear infections? (yes,no) *If yes, can you please elaborate?*	
What is your gender? M F	

APPENDIX D: POST-QUESTIONNAIRE AND MEAN ANSWERS PER GROUP

Very True was given a value of 0 and Very False, a value of 5.

Question	*Control Mean*	*Control StDev*	*SIM Mean*	*SIM StDev*
I felt I was immersed in a space station.	1.29	1.39	1.18	1.10
I felt confident I knew the names of all the modules after Step 1.	0.58	1.16	0.25	0.64
I felt confident I knew the layout of the station after Step 2.	3.21	1.50	1.86	1.60
I feel confident I know the layout of the station after Step 3.	2.88	1.79	1.86	1.65
I feel I have a good understanding of how all the modules are oriented with respect to Control module.	2.63	1.67	1.93	1.48
I always understood what orientation I was in Step 3.	1.63	1.90	0.82	1.30
I felt Step 2 was very hard.	2.33	1.44	2.50	1.70
I feel I have a good understanding of where all the modules are.	2.58	1.83	1.32	1.32
The question "Once in target module, how would you turn to be upright in the target module?" was too hard.	2.54	1.67	2.07	1.77
I felt the world was really upside-down sometimes.	2.29	1.42	2.61	1.77
I had a mental map of the entire station with me inside.	2.17	1.75	1.68	1.84
I feel I could get from any module in the station to any other module.	2.17	1.53	1.57	1.57
I feel I have a good understanding of how all the modules are oriented with respect to other modules <u>aside</u> from Control.	2.79	1.47	2.25	1.55
I found it hard to imagine myself rotated inside the station.	2.63	1.49	1.96	1.42
In general, the questions for route description were too hard.	3.32	1.23	3.00	1.21
I tried to memorize the sequences of rotations necessary to reach the target modules.	2.50	1.98	3.39	1.63
The question "Once in the node, how would you turn to face the target module?" was too hard.	3.75	0.97	3.43	1.14
I felt Step 3 was very hard.	1.79	1.47	2.39	1.55
I feel I can recall what the walls, ceilings, and floors look like for all modules.	3.17	1.25	3.04	1.45
The pointing task was too hard.	3.04	1.25	3.36	1.41
In Step 3, I decided on the route description as I went along.	1.09	1.30	0.62	0.62
I had a miniature mental map of the entire station.	2.17	1.64	1.25	1.55
I tried to memorize the sequence of landmarks necessary to reach the target modules.	2.50	1.37	2.68	1.64
I feel I can recall what the walls, ceilings, and floors look like for the Control module.	1.71	1.10	1.71	1.34
I found it hard to imagine the station rotated.	3.04	1.32	3.14	1.55
I often rely on maps to get from place to place.	1.58	1.83	1.58	1.44

If you used the miniature model, please answer these questions. Otherwise, skip these next true/false questions.				
The SIM was essential in learning the station layout.	N/A	N/A	0.29	0.43
It was very hard to use the SIM in Step 2.	N/A	N/A	3.36	1.26
It was hard to do Step 3 without the SIM.	N/A	N/A	2.04	1.61
I imagined the SIM model in Step 3.	N/A	N/A	1.43	1.40

Learning to Navigate Through a Virtual Space Station

Post-Questionnaire

Name:____________________

Subject Number:____________

I felt I was immersed in a space station.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I felt confident I knew the names of all the modules after Step 1.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I felt confident I knew the layout of the station after Step 2.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I feel confident I know the layout of the station after Step 3.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I feel I have a good understanding of how all the modules are oriented with respect to Control module.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I always understood what orientation I was in Step 3.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I felt Step 2 was very hard.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I feel I have a good understanding of where all the modules are.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
The question "Once in target module, how would you turn to be upright in the target module?" was too hard.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I felt the world was really upside-down sometimes.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I had a mental map of the entire station with me inside.	Very True ├──┼──┼──┼──┼──┼──┤ Very False

I feel I could get from any module in the station to any other module.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I feel I have a good understanding of how all the modules are oriented with respect to other modules <u>aside</u> from Control.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I found it hard to imagine myself rotated inside the station.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
In general, the questions for route description were too hard.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I tried to memorize the sequences of rotations necessary to reach the target modules.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
The question "Once in the node, how would you turn to face the target module?" was too hard.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I felt Step 3 was very hard.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I feel I can recall what the walls, ceilings, and floors look like for all modules.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
The pointing task was too hard.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
In Step 3, I decided on the route description as I went along.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I had a miniature mental map of the entire station.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I tried to memorize the sequence of landmarks necessary to reach the target modules.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I feel I can recall what the walls, ceilings, and floors look like for the Control module.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I found it hard to imagine the station rotated.	Very True ├──┼──┼──┼──┼──┼──┤ Very False
I often rely on maps to get from place to place.	Very True ├──┼──┼──┼──┼──┼──┤ Very False

If you used the miniature model, please answer these questions. Otherwise, skip these next true/false questions.	
The SIM was essential in learning the station layout.	Very True |—|—|—|—|—|—| Very False
It was very hard to use the SIM in Step 2.	Very True |—|—|—|—|—|—| Very False
It was hard to do Step 3 without the SIM.	Very True |—|—|—|—|—|—| Very False
I imagined the SIM model in Step 3.	Very True |—|—|—|—|—|—| Very False

Which objects did you use to determine your orientation in the Control Module? (Circle all those that are appropriate).

Yellow, red, green light bulbs	Handles on racks
Cameras on the hatches	Blue monitors
Brown objects	Light blue racks/instruments
Green hatch	Yellow hatch
Silver sphere	Pink shelves
Lights	White tubing
Chamfered end of hatch ("U")	Large blue computer
Vents	Brownies
Blue Hatch	Other:____________________

Please construct the station with the modules provided.

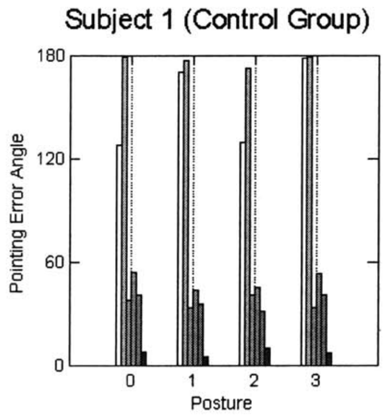
What strategies did you use to figure out the layout? ... the routes?

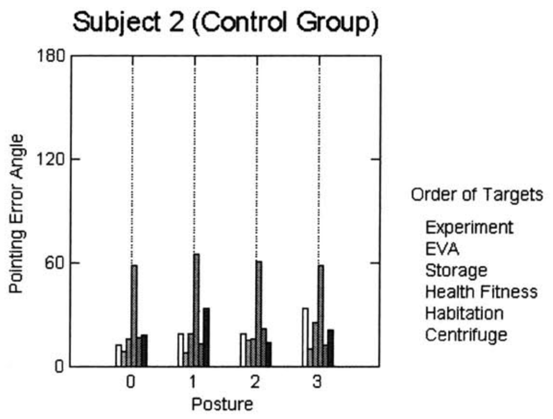
Appendix E: Subject Data For Pointing Task

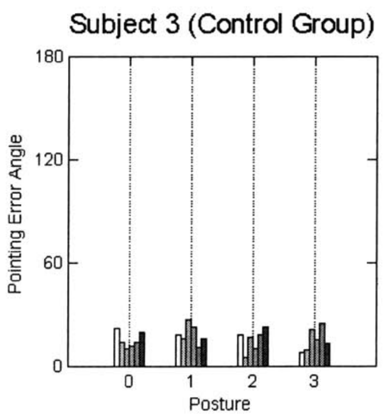
Error Angles

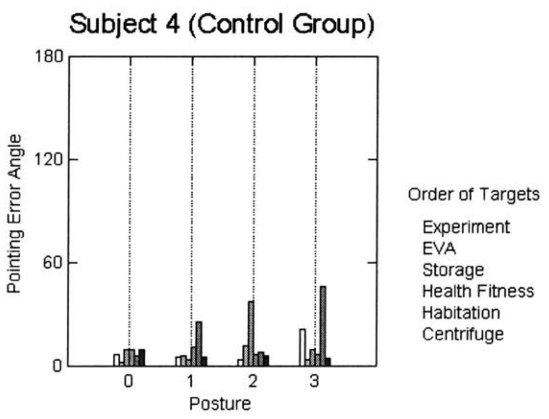
Postures

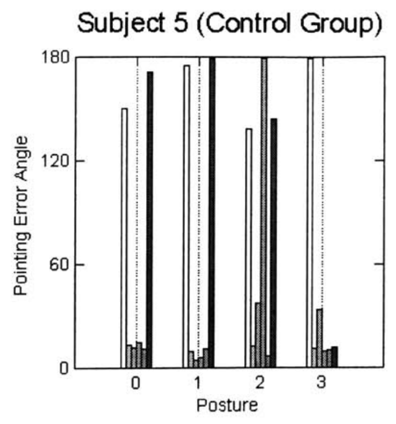
1: Upright; 2: Right Shoulder Down; 3: Upside Down; 4: Left Shoulder Down.

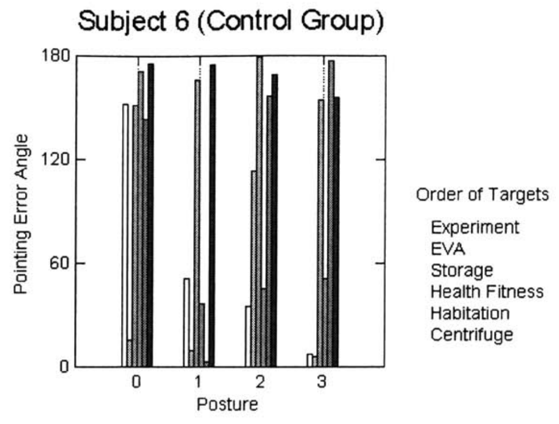


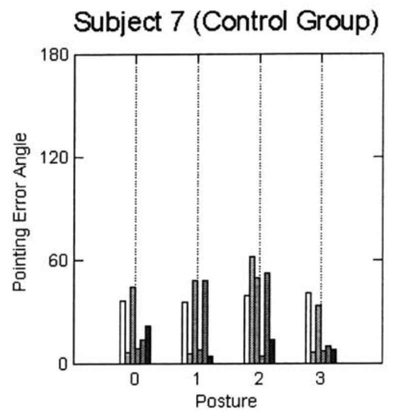


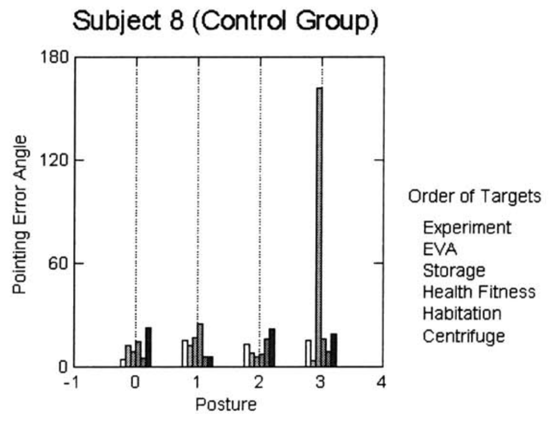


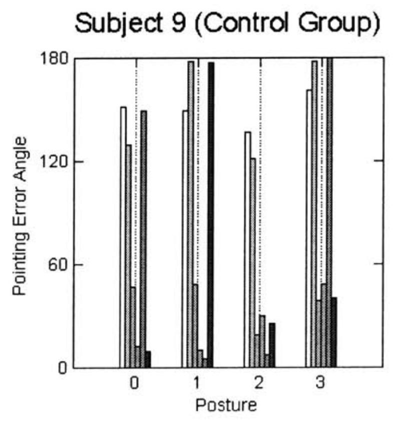


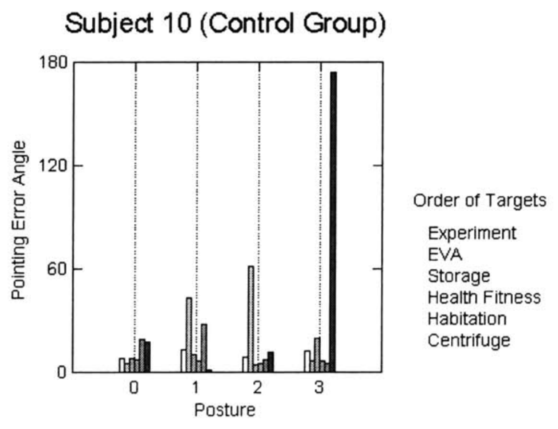

Subject 5 (Control Group)
Pointing Error Angle
180
120
60
0
0
1
2
3
Posture

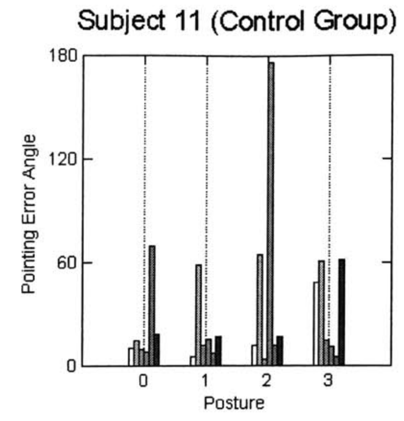

Subject 6 (Control Group)
Pointing Error Angle
180
120
60
0
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

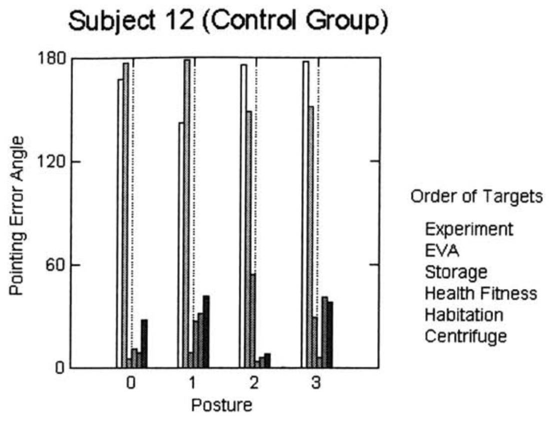

Subject 7 (Control Group)
Pointing Error Angle
180
120
60
0
0
1
2
3
Posture

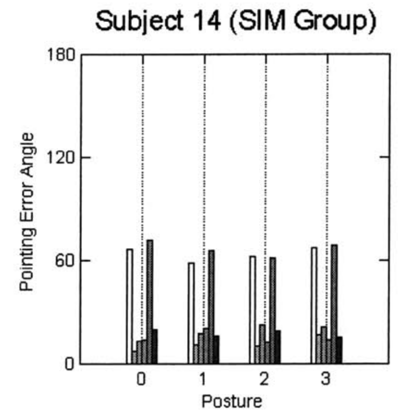

Subject 8 (Control Group)
Pointing Error Angle
180
120
60
0
-1
0
1
2
3
4
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

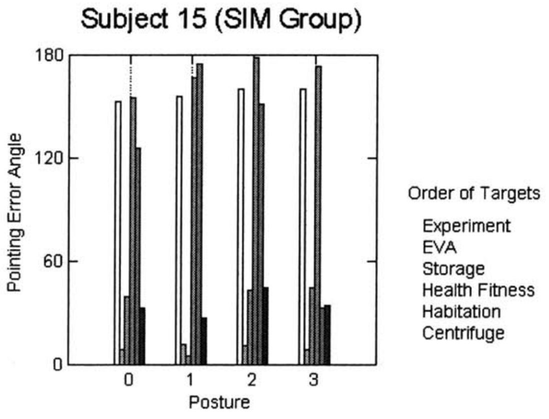

Subject 9 (Control Group)
Pointing Error Angle
180
120
60
0
0
1
2
3
Posture

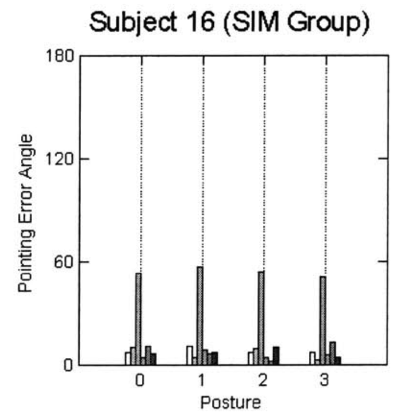

Subject 10 (Control Group)
Pointing Error Angle
180
120
60
0
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

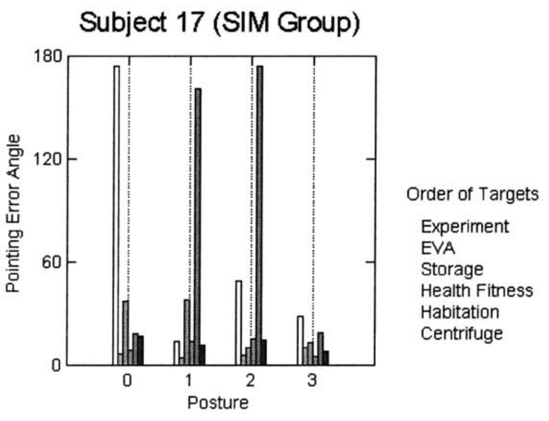

Subject 11 (Control Group)
Pointing Error Angle
180
120
60
0
0
1
2
3
Posture

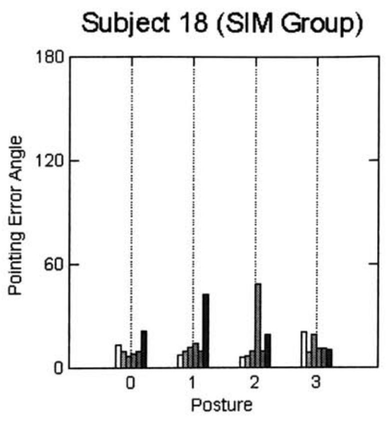

Subject 12 (Control Group)
Pointing Error Angle
180
120
60
0
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

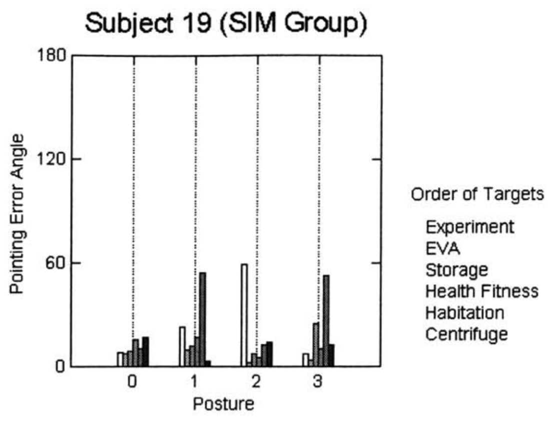

Subject 14 (SIM Group)
Pointing Error Angle
180
120
60
0
0
1
2
3
Posture

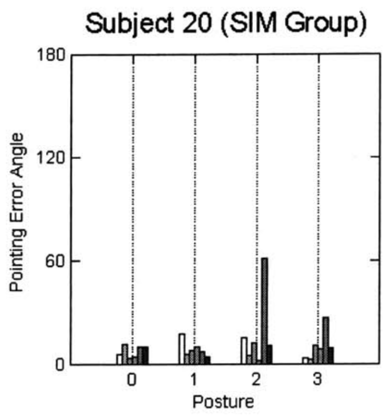

Subject 15 (SIM Group)
Pointing Error Angle
180
120
60
0
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

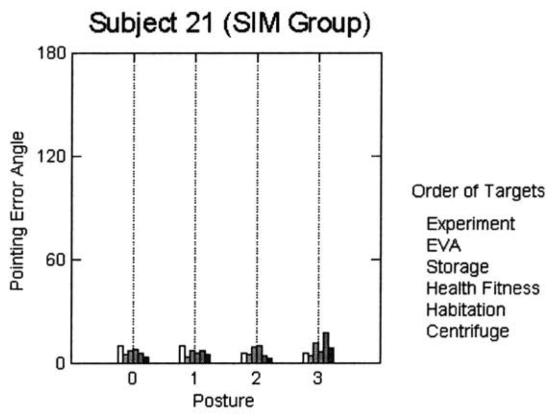

Subject 16 (SIM Group)
Pointing Error Angle
180
120
60
0
0
1
2
3
Posture

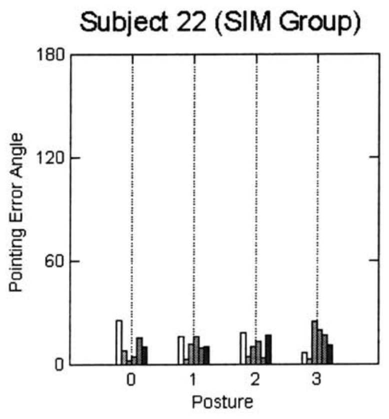

Subject 17 (SIM Group)
Pointing Error Angle
180
120
60
0
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

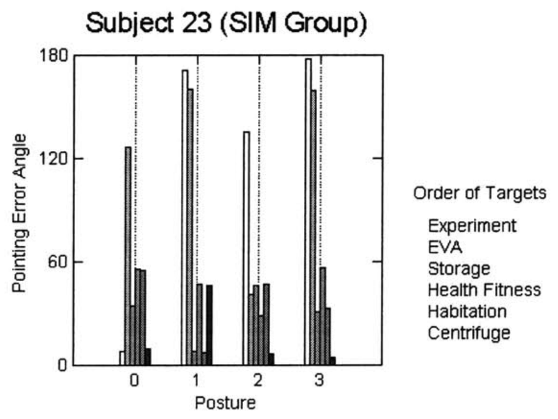

Subject 18 (SIM Group)
180
120
60
0
Pointing Error Angle
0
1
2
3
Posture

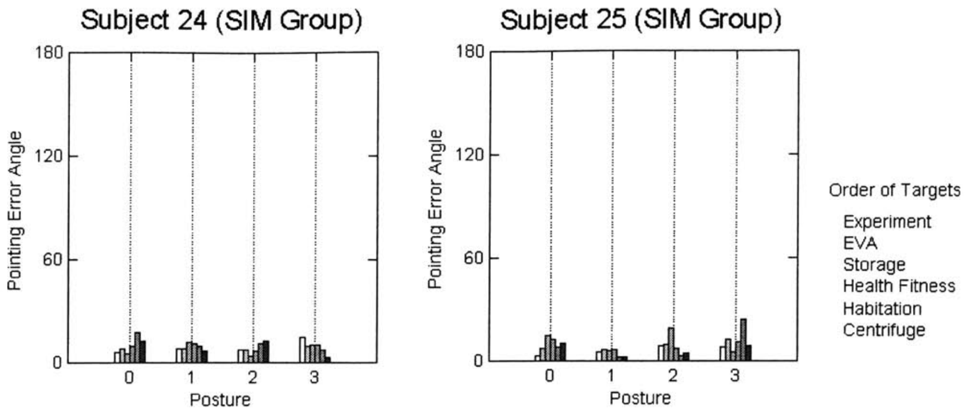

Subject 19 (SIM Group)
180
120
60
0
Pointing Error Angle
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

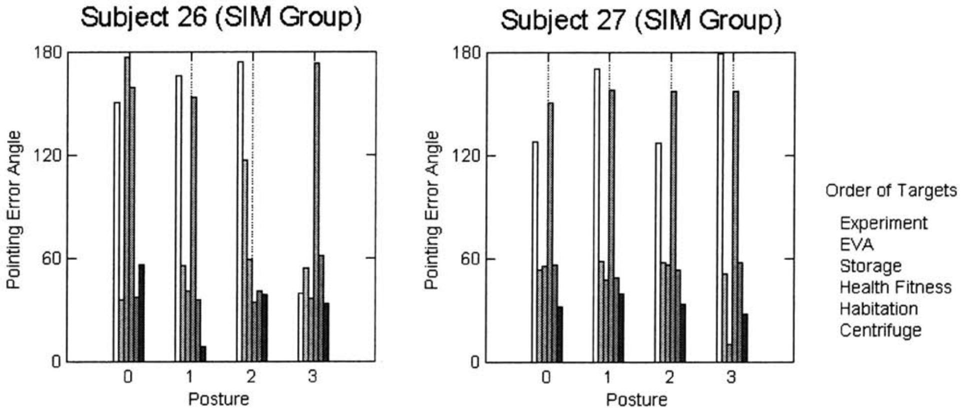

Subject 20 (SIM Group)
180
120
60
0
Pointing Error Angle
0
1
2
3
Posture


Subject 21 (SIM Group)
180
120
60
0
Pointing Error Angle
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge


Subject 22 (SIM Group)
180
120
60
0
Pointing Error Angle
0
1
2
3
Posture


Subject 23 (SIM Group)
180
120
60
0
Pointing Error Angle
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

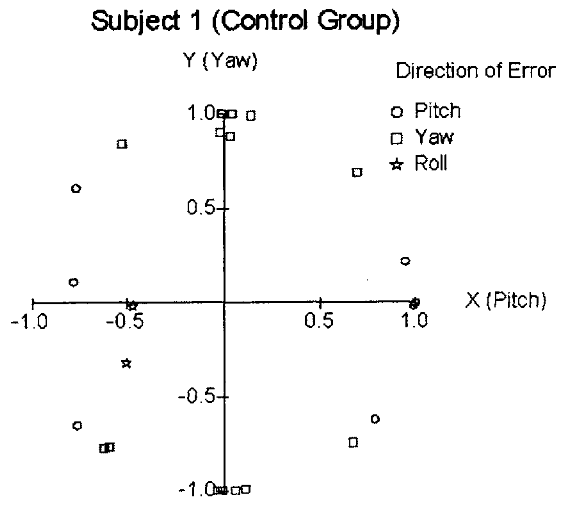


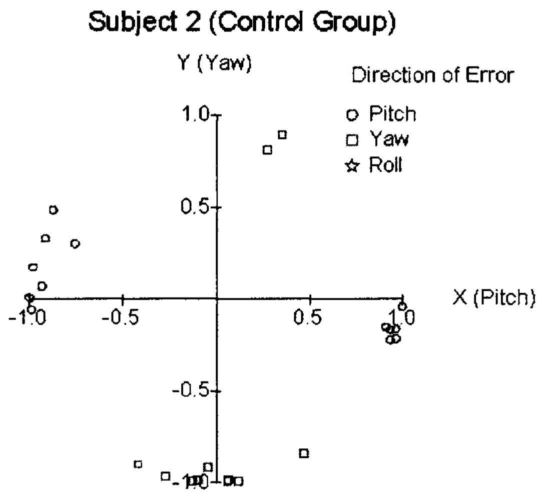


Control Subject	Comments on Direction Pointing skills
1	Poor subject; misplaced 3 modules. Fairly good mental rotations (consistent errors).
2	Good subject; misplaced 1 module. Good mental rotations.
3	Excellent subject.
4	Good/excellent subject. Mistakes are only made in other initial postures.
5	Fair subject; misplaced 2 modules. Good mental rotations (consistent errors).
6	Very poor subject; misplaced 5 modules.
7	Fair subject; misplaced 1 or 2 modules. More mistakes as changing initial posture.
8	Excellent subject; only one mistake (accidental?).
9	Very poor subject; misplaced 4 modules.
10	Good subject. Mistakes are made in other initial postures.
11	Fair subject; misplaced 2. Fair mental rotations (consistent errors & diff. postures).
12	Fair subject; misplaced 2. Fair mental rotations (consistent errors & diff. postures).

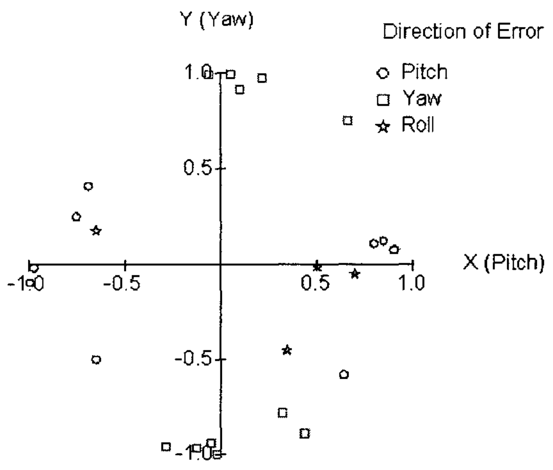
SIM Subject	Comments on Direction Pointing skills
14	Fair subject; misplaced 2. Good mental rotations (consistent errors).
15	Poor subject; misplaced 3 modules. Consistent errors & errors in different postures.
16	Good subject; misplaced 1 module. Good mental rotations.
17	Fair subject; errors in different conditions.
18	Good/excellent subject. Mistakes are only made in other initial postures.
19	Good/excellent subject. Mistakes are only made in other initial postures.
20	Excellent subject; only one mistake.
21	Excellent subject.
22	Excellent subject.
23	Poor subject; misplaced 3 modules? Poor mental rotation (inconsistent errors).
24	Excellent subject.
25	Excellent subject.
26	Very poor subject; misplaced 4? Poor mental rotations.
27	Very poor subject; misplaced 5. Fairly good mental rotations (consistent errors).

DISTRIBUTION OF ERROR VECTOR X AND Y COMPONENTS

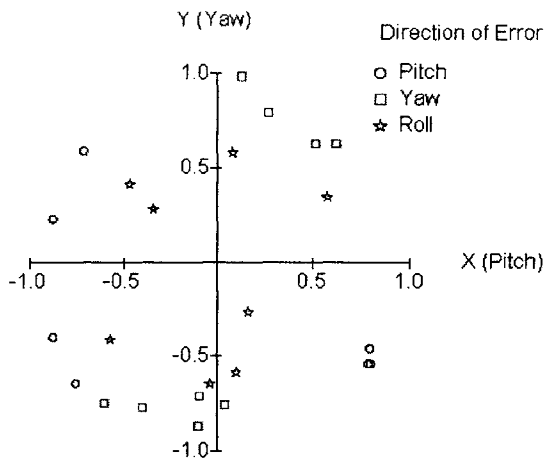




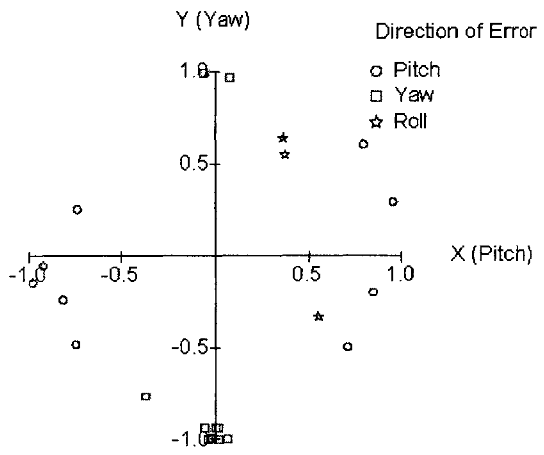
Subject 3 (Control Group)



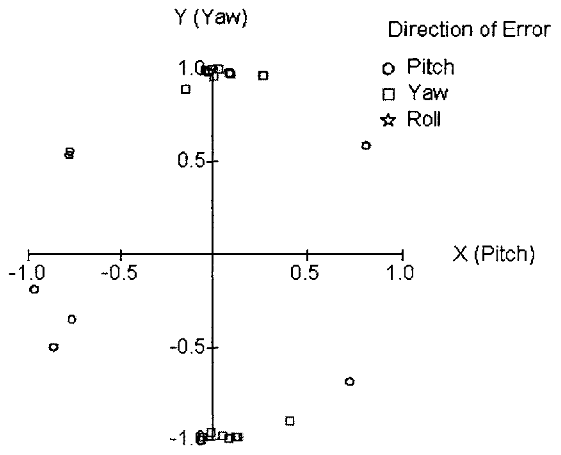
Subject 4 (Control Subject)

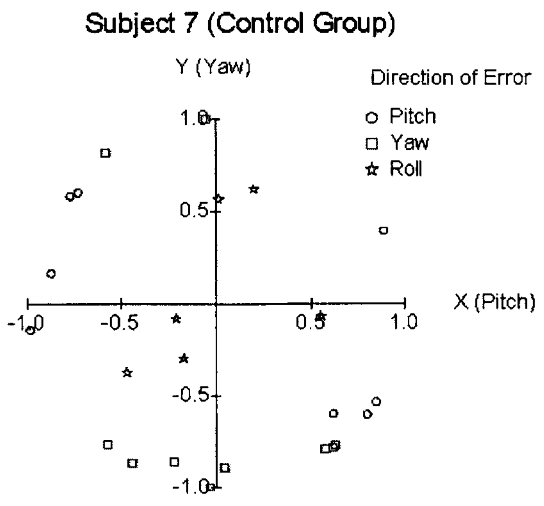


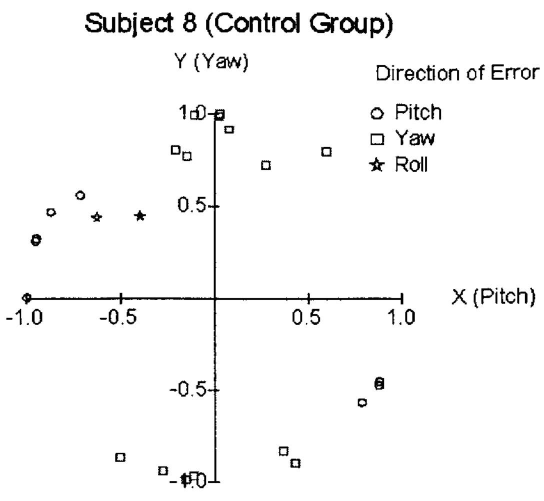
Subject 5 (Control Group)

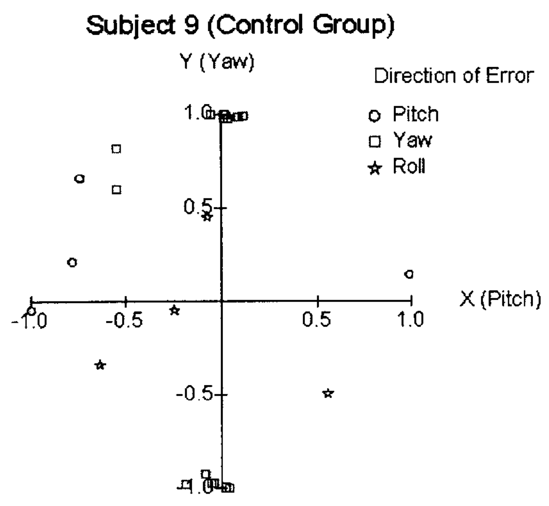


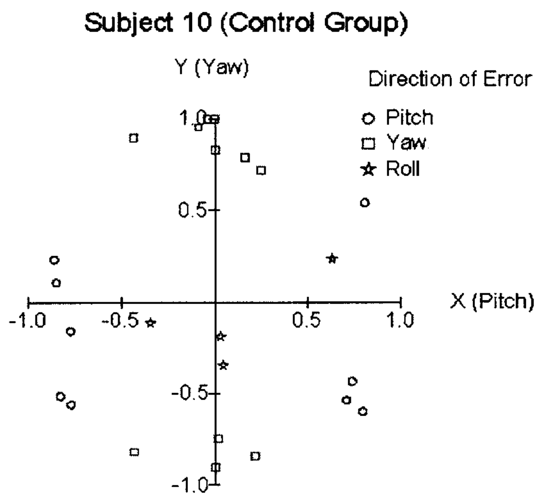
Subject 6 (Control Group)



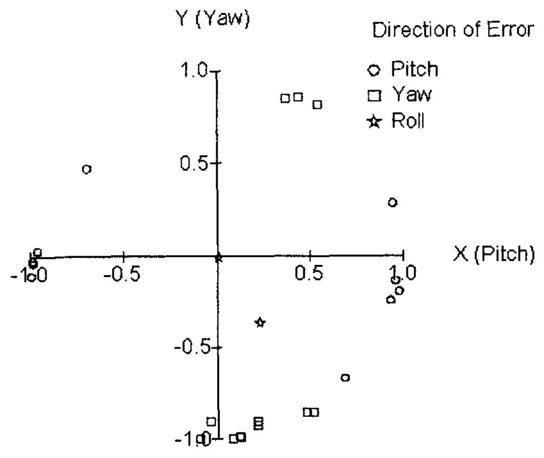

Subject 7 (Control Group)
Y (Yaw)
Direction of Error
○ Pitch
□ Yaw
☆ Roll
X (Pitch)
1.0
0.5
-0.5
-1.0
-1.0
-0.5
0.5
1.0


Subject 8 (Control Group)
Y (Yaw)
Direction of Error
○ Pitch
□ Yaw
☆ Roll
X (Pitch)
1.0
0.5
-0.5
-1.0
-1.0
-0.5
0.5
1.0

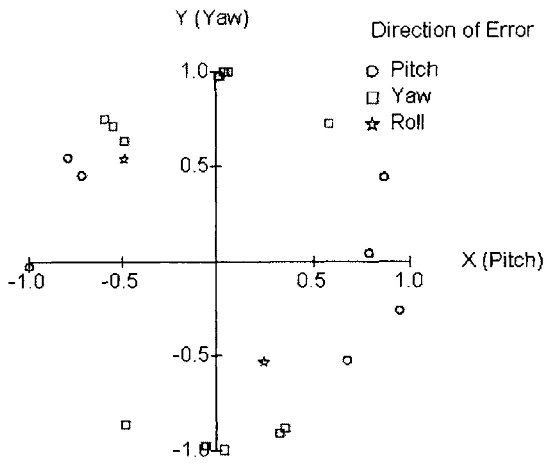

Subject 9 (Control Group)
Y (Yaw)
Direction of Error
○ Pitch
□ Yaw
☆ Roll
X (Pitch)
1.0
0.5
-0.5
-1.0
-1.0
-0.5
0.5
1.0


Subject 10 (Control Group)
Y (Yaw)
Direction of Error
○ Pitch
□ Yaw
☆ Roll
X (Pitch)
1.0
0.5
-0.5
-1.0
-1.0
-0.5
0.5
1.0

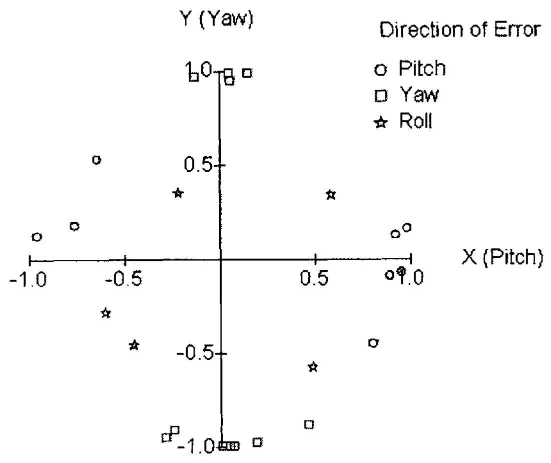
Subject 11 (Control Group)



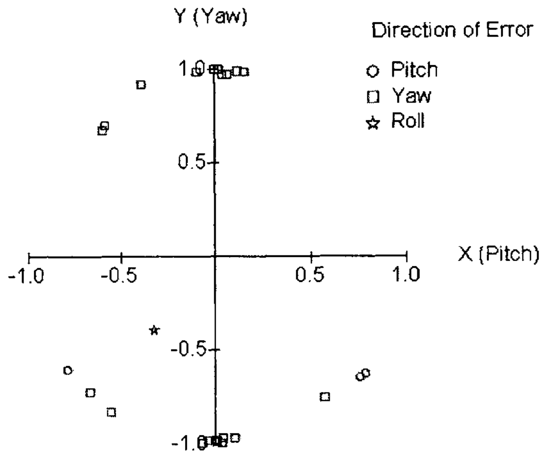
Subject 12 (Control Group)

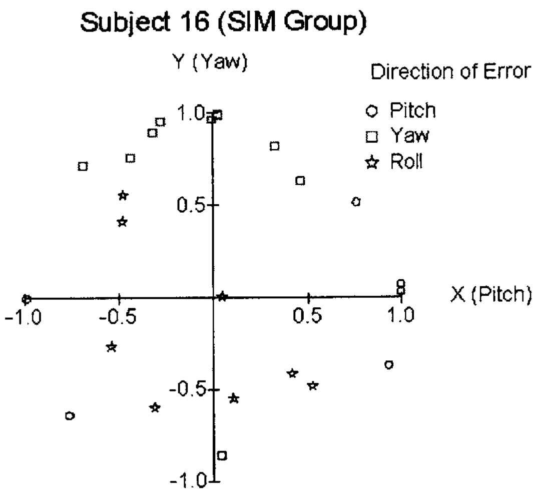


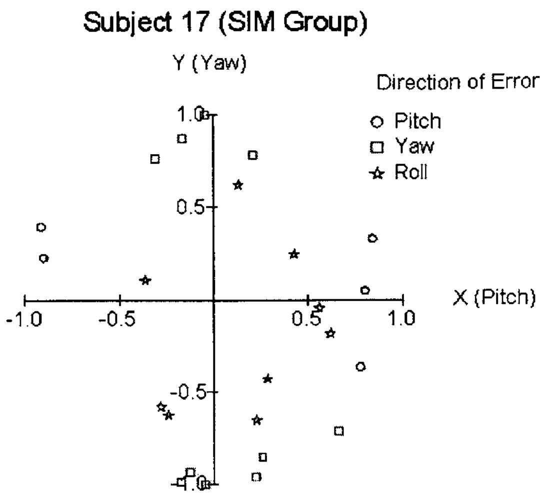
Subject 14 (SIM Group)

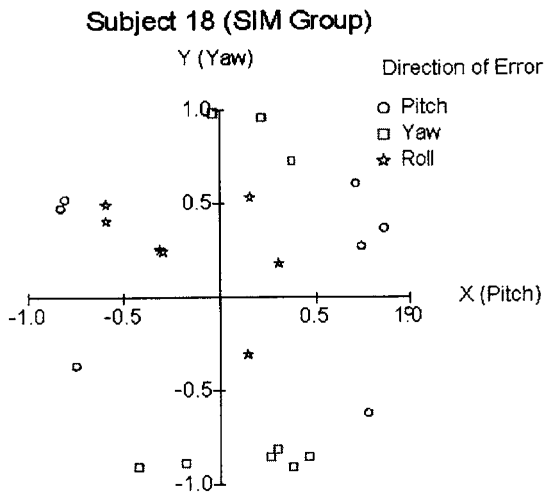


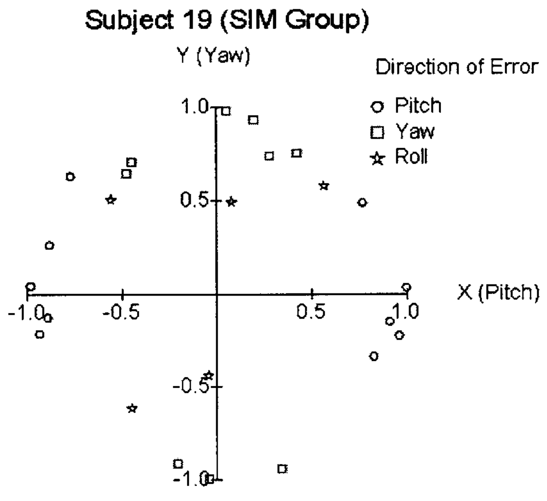
Subject 15 (SIM Group)



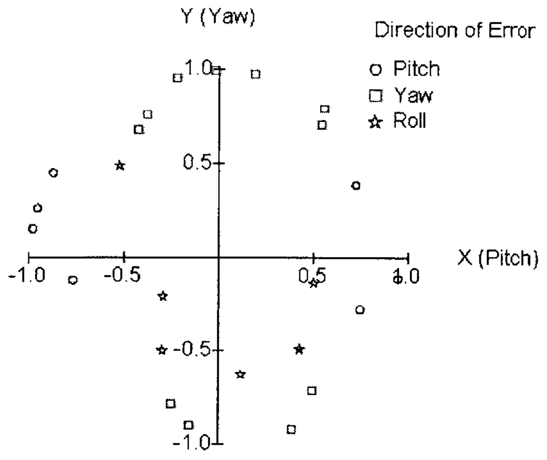

Subject 16 (SIM Group)
Y (Yaw)
Direction of Error
○ Pitch
□ Yaw
☆ Roll
X (Pitch)
1.0
0.5
-0.5
-1.0
-1.0
-0.5
0.5
1.0


Subject 17 (SIM Group)
Y (Yaw)
Direction of Error
○ Pitch
□ Yaw
☆ Roll
X (Pitch)
1.0
0.5
-0.5
-1.0
-1.0
-0.5
0.5
1.0

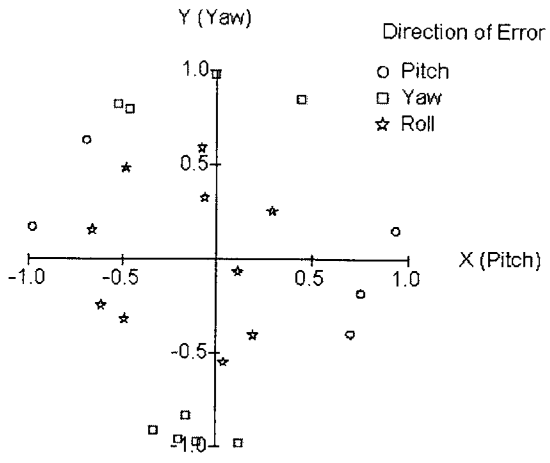

Subject 18 (SIM Group)
Y (Yaw)
Direction of Error
○ Pitch
□ Yaw
☆ Roll
X (Pitch)
1.0
0.5
-0.5
-1.0
-1.0
-0.5
0.5
1.0


Subject 19 (SIM Group)
Y (Yaw)
Direction of Error
○ Pitch
□ Yaw
☆ Roll
X (Pitch)
1.0
0.5
-0.5
-1.0
-1.0
-0.5
0.5
1.0

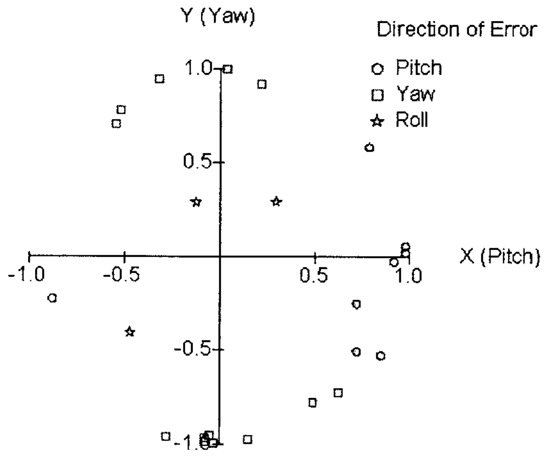
Subject 20 (SIM Group)



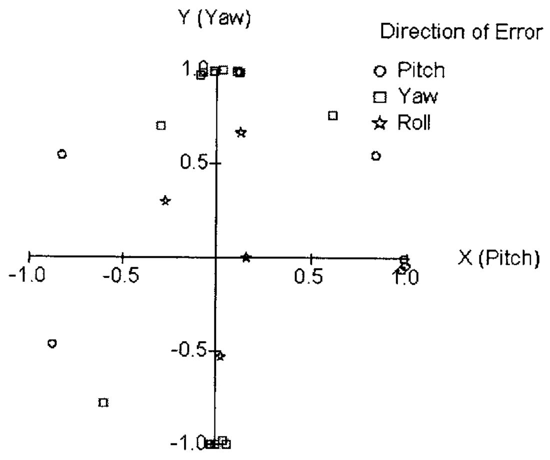
Subject 21 (SIM Group)



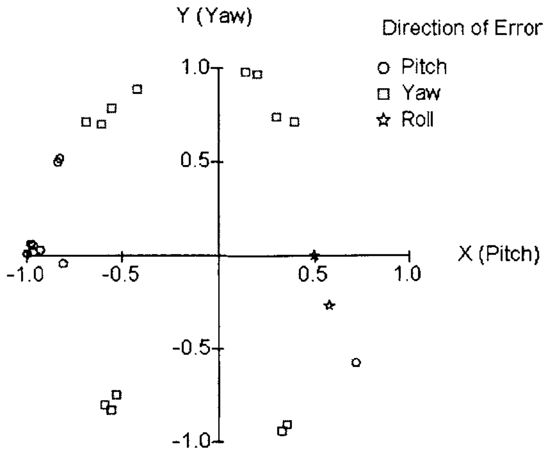
Subject 22 (SIM Group)



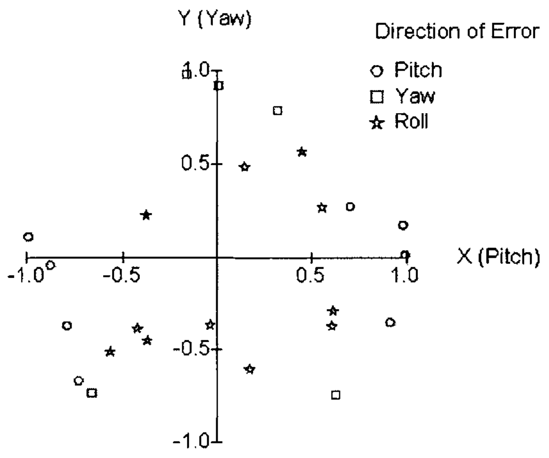
Subject 23 (SIM Group)



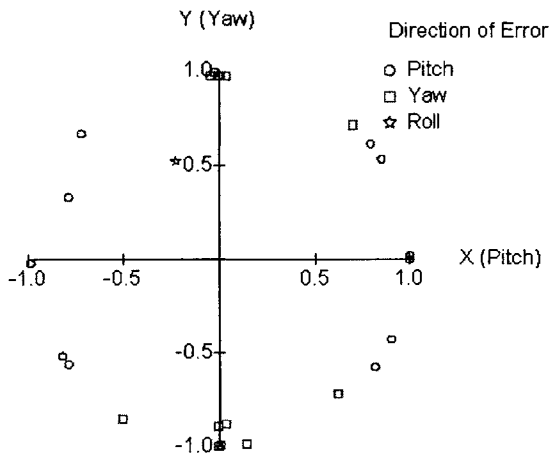
Subject 24 (SIM Group)



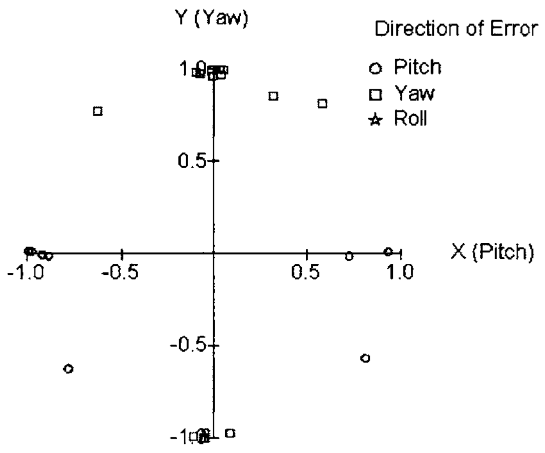
Subject 25 (SIM Group)



Subject 26 (SIM Group)



Subject 27 (SIM Group)

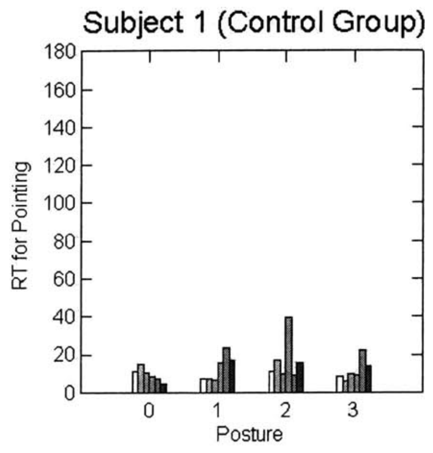


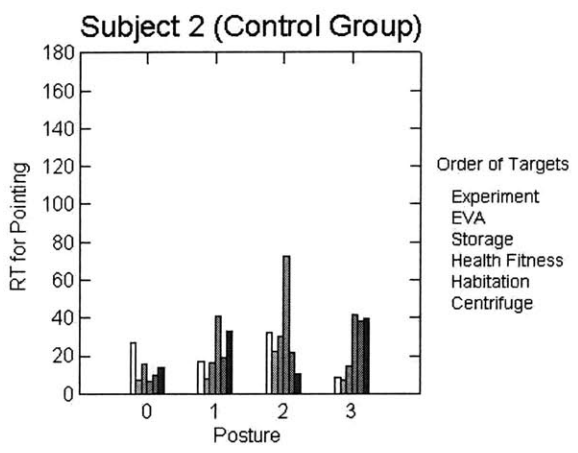
Observation: the poor subjects per group (1, 6, 9 & 15, 23, 26, 27) have higher clustering around pure yaws and/or pitches, which others have a more spread distribution around a unit circle.

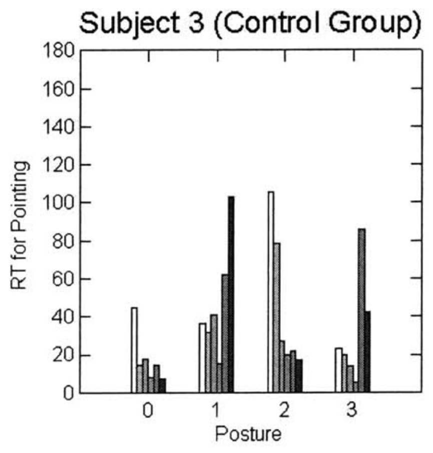
RESPONSE TIME TO POINTING TASK

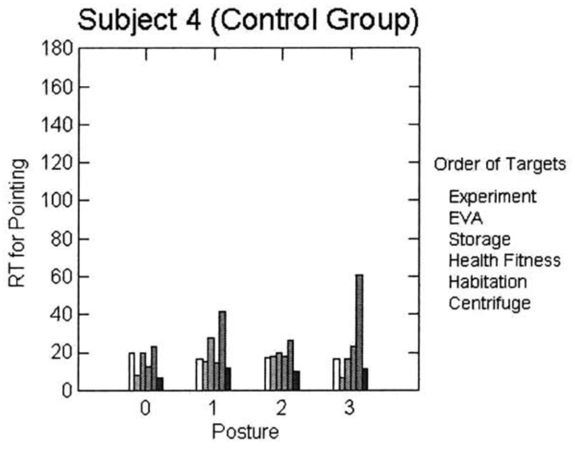
Postures

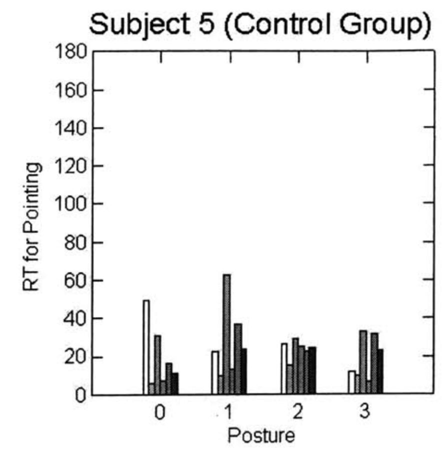
1: Upright; 2: Right Shoulder Down; 3: Upside Down; 4: Left Shoulder Down.

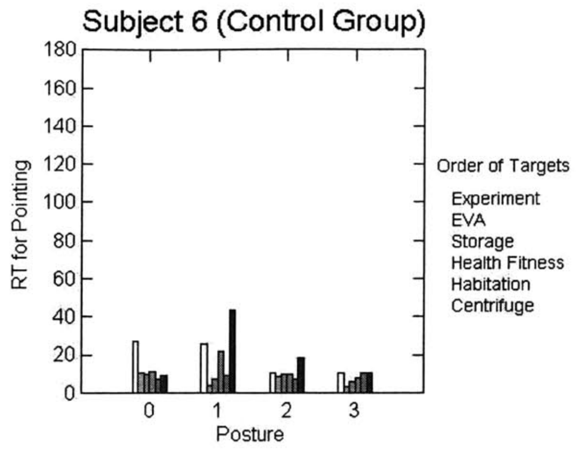


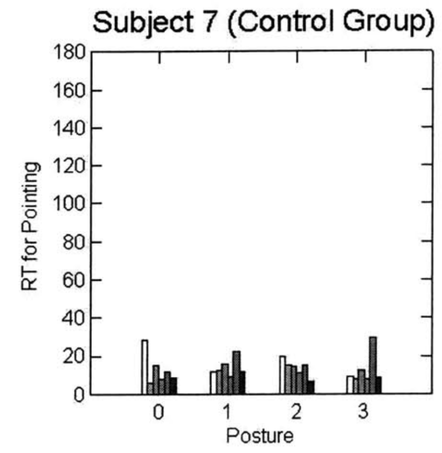


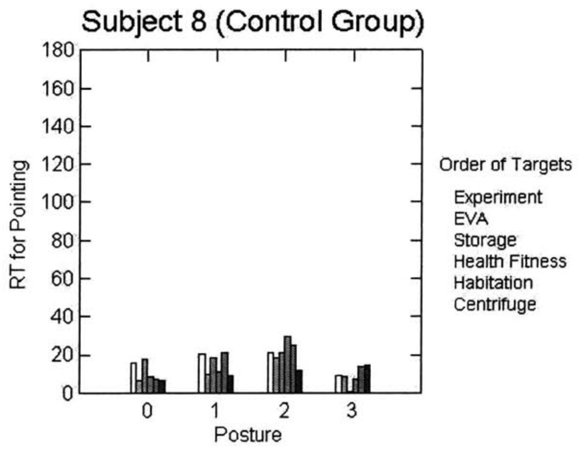


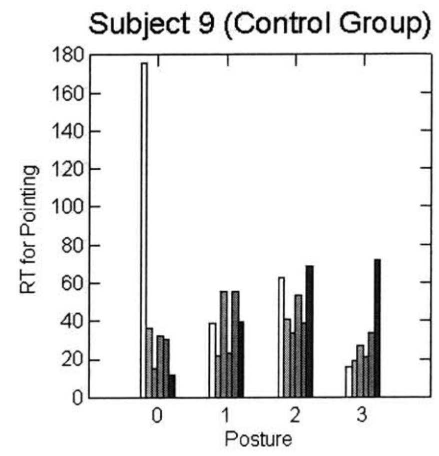


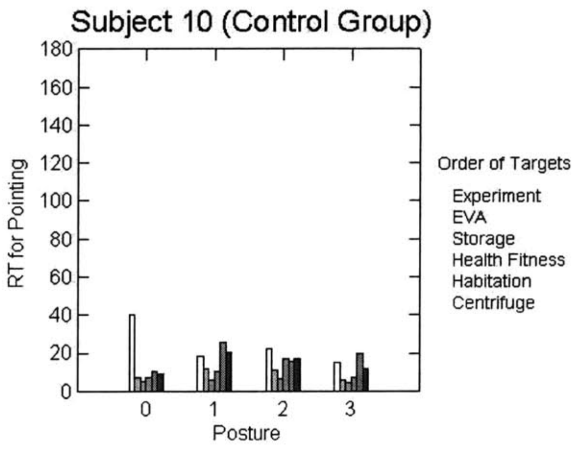

Subject 5 (Control Group)
RT for Pointing
180
160
140
120
100
80
60
40
20
0
0
1
2
3
Posture

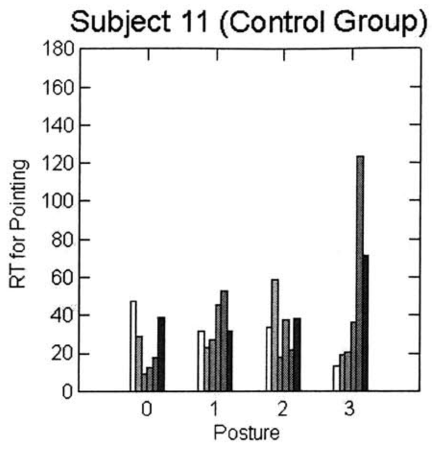

Subject 6 (Control Group)
RT for Pointing
180
160
140
120
100
80
60
40
20
0
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

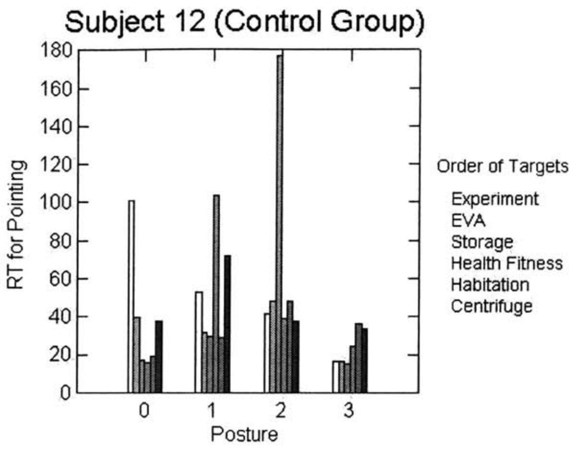

Subject 7 (Control Group)
RT for Pointing
180
160
140
120
100
80
60
40
20
0
0
1
2
3
Posture

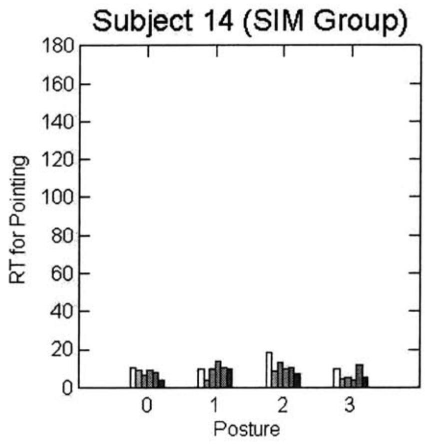

Subject 8 (Control Group)
RT for Pointing
180
160
140
120
100
80
60
40
20
0
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

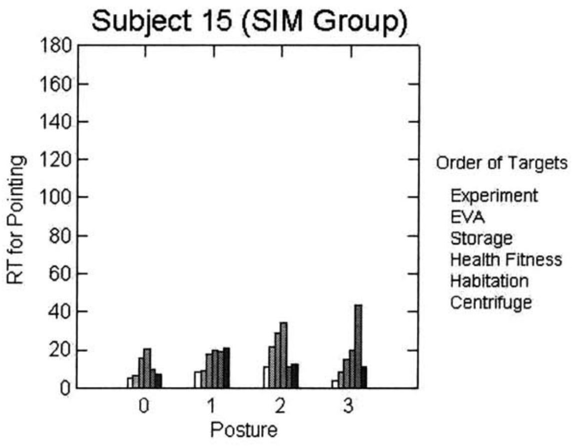

Subject 9 (Control Group)
RT for Pointing
180
160
140
120
100
80
60
40
20
0
0
1
2
3
Posture

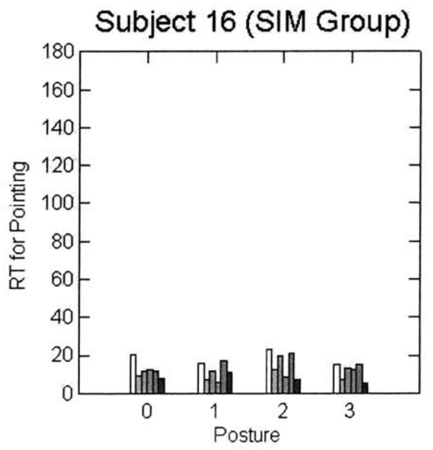

Subject 10 (Control Group)
RT for Pointing
180
160
140
120
100
80
60
40
20
0
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

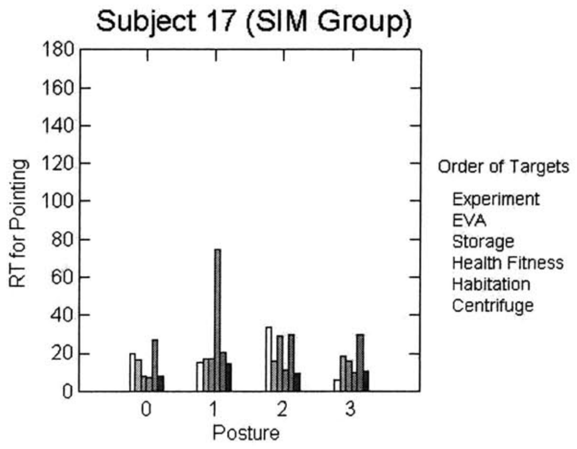

Subject 11 (Control Group)
RT for Pointing
Posture
180
160
140
120
100
80
60
40
20
0
0
1
2
3

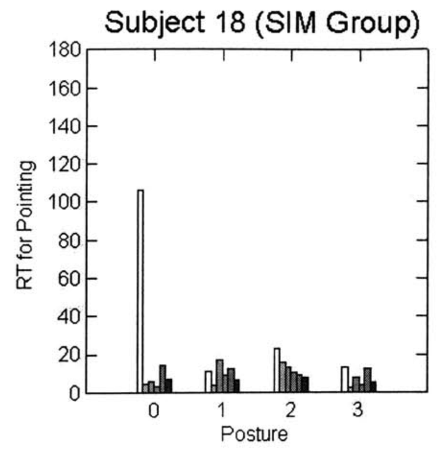

Subject 12 (Control Group)
RT for Pointing
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

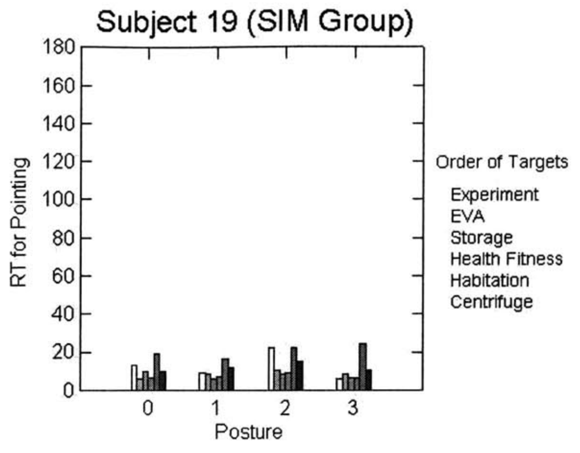

Subject 14 (SIM Group)
RT for Pointing
Posture

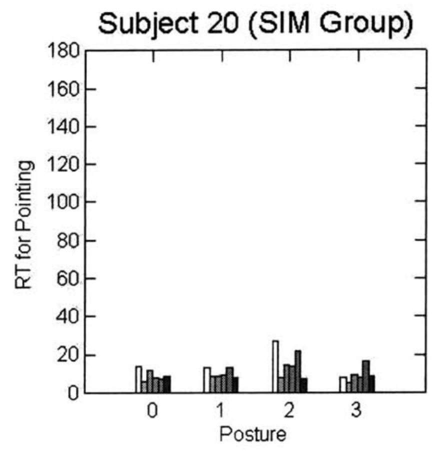

Subject 15 (SIM Group)
RT for Pointing
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

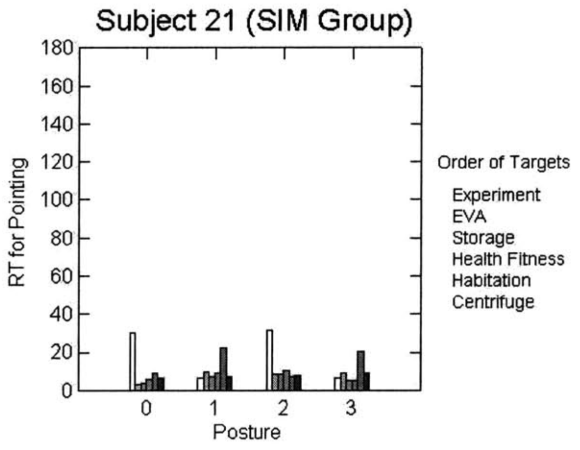

Subject 16 (SIM Group)
RT for Pointing
Posture

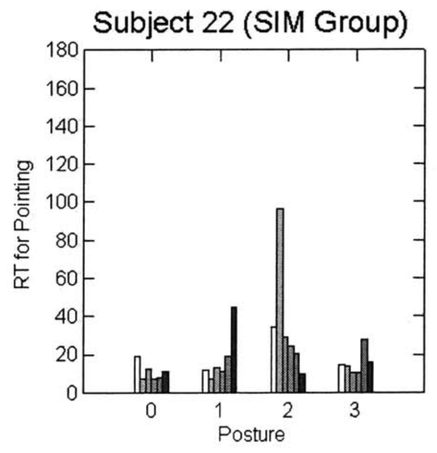

Subject 17 (SIM Group)
RT for Pointing
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

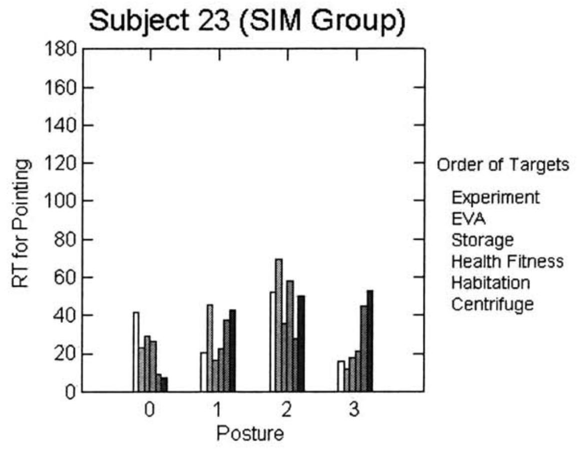

Subject 18 (SIM Group)
RT for Pointing
0
20
40
60
80
100
120
140
160
180
Posture
0
1
2
3

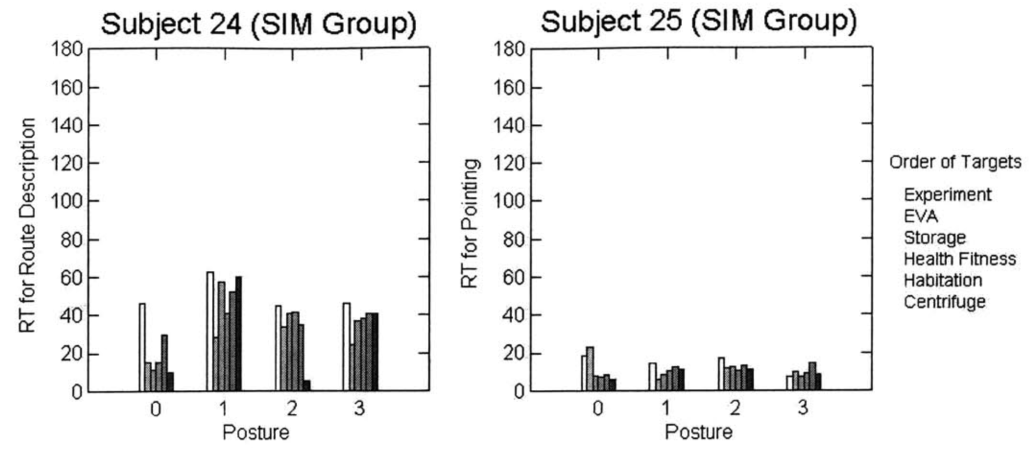

Subject 19 (SIM Group)
RT for Pointing
0
20
40
60
80
100
120
140
160
180
Posture
0
1
2
3
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

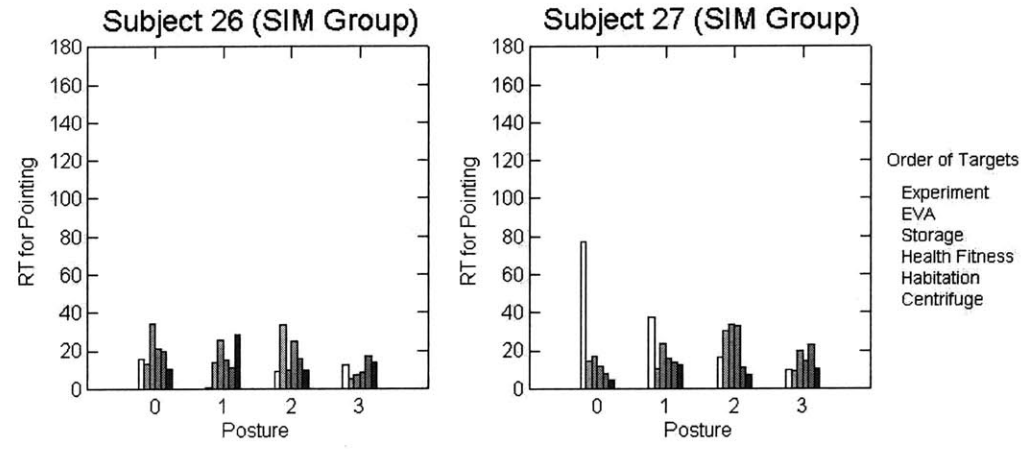

Subject 20 (SIM Group)
RT for Pointing
0
20
40
60
80
100
120
140
160
180
Posture
0
1
2
3


Subject 21 (SIM Group)
RT for Pointing
0
20
40
60
80
100
120
140
160
180
Posture
0
1
2
3
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge


Subject 22 (SIM Group)
RT for Pointing
0
20
40
60
80
100
120
140
160
180
Posture
0
1
2
3

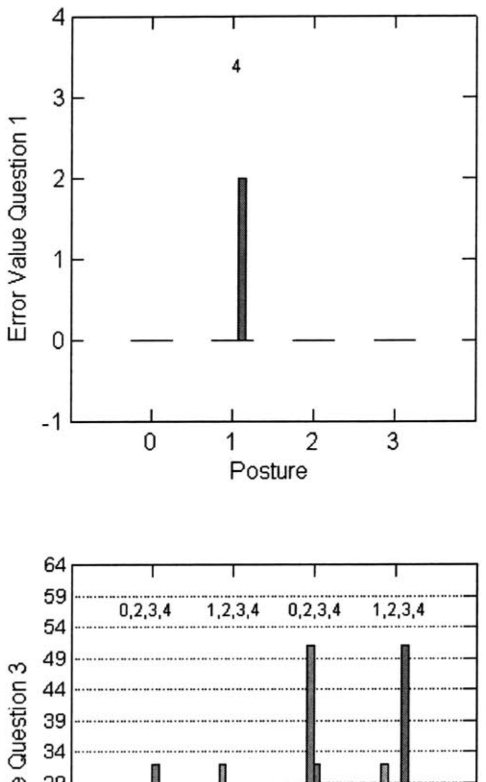

Subject 23 (SIM Group)
RT for Pointing
0
20
40
60
80
100
120
140
160
180
Posture
0
1
2
3
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

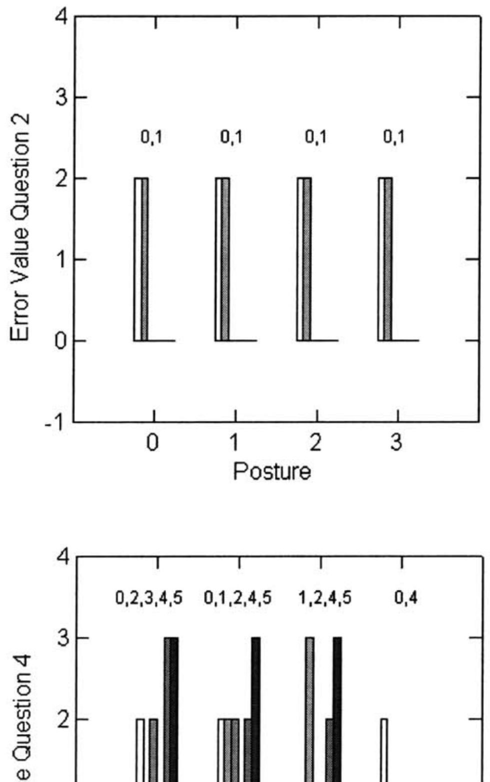

Subject 24 (SIM Group)
RT for Route Description
Posture
Subject 25 (SIM Group)
RT for Pointing
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

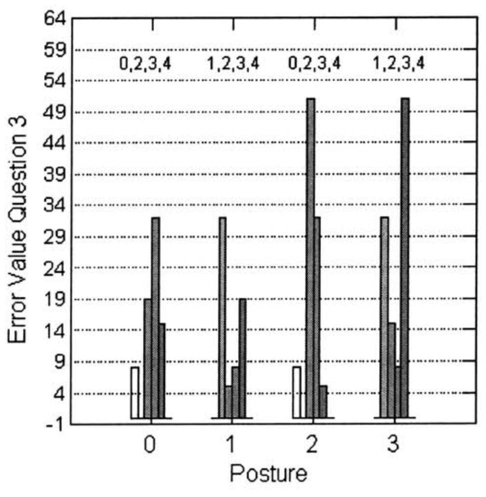

Subject 26 (SIM Group)
RT for Pointing
Posture
Subject 27 (SIM Group)
RT for Pointing
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

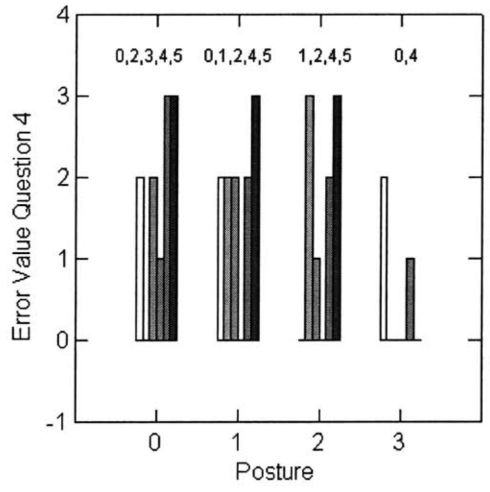
APPENDIX F: SUBJECT DATA FOR ROUTE DESCRIPTION TASK

SUBJECT 1 (CONTROL GROUP)





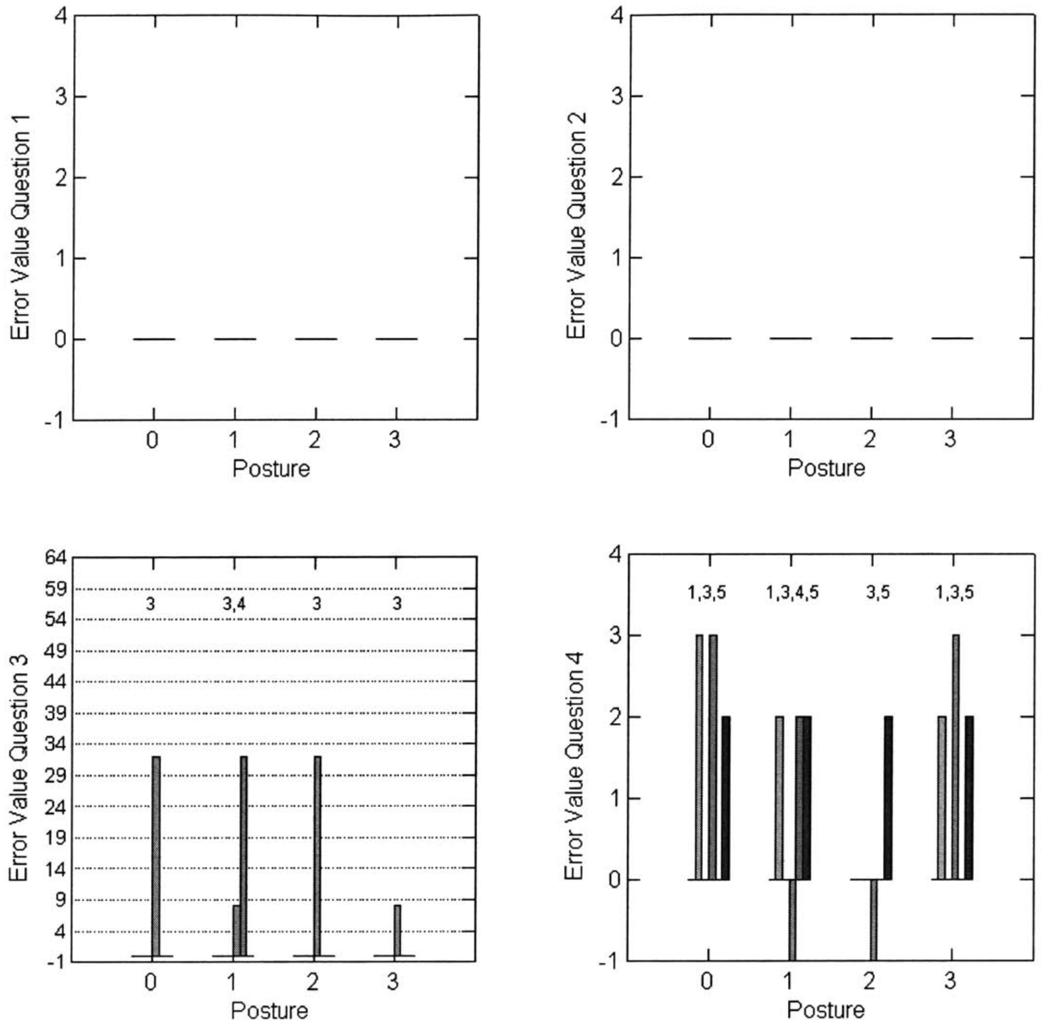




Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

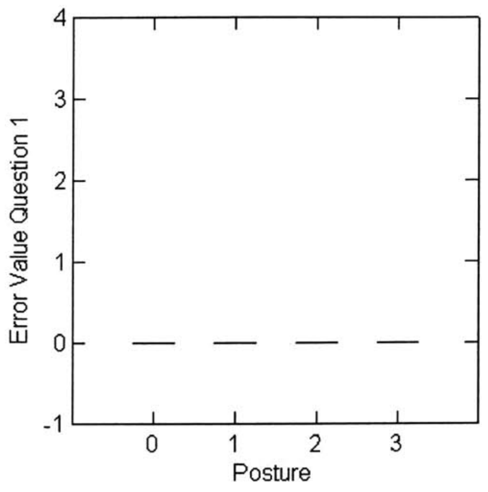
SUBJECT 2 (CONTROL GROUP)

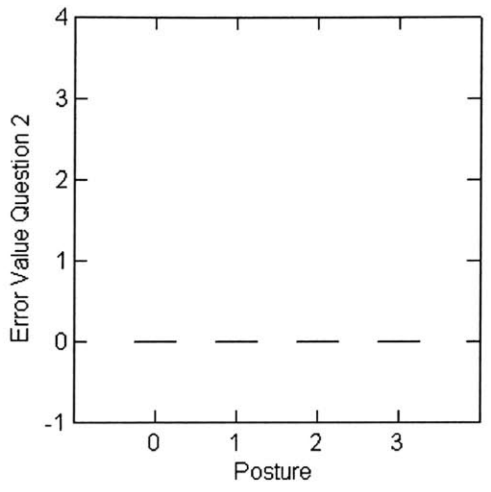


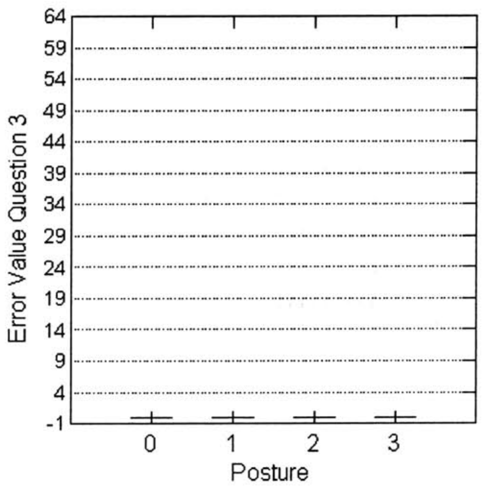
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

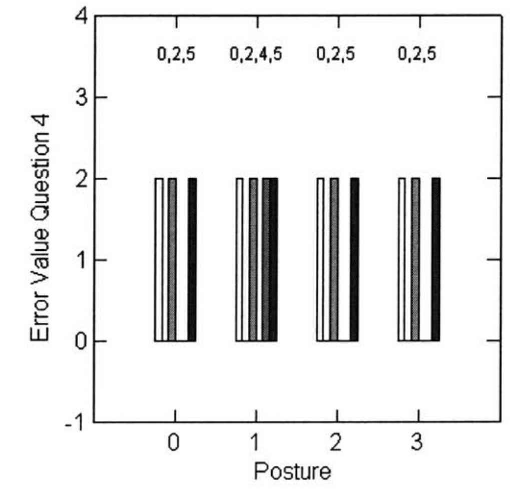
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

SUBJECT 3 (CONTROL GROUP)





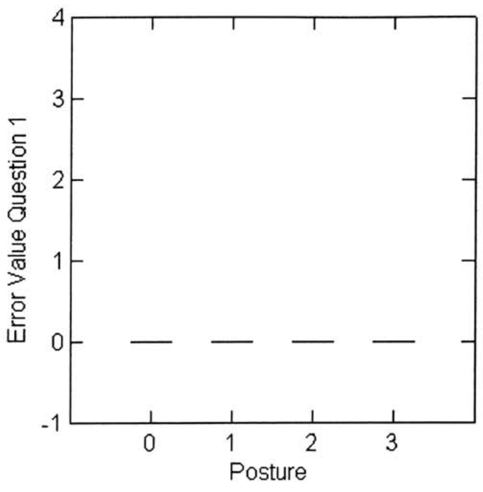


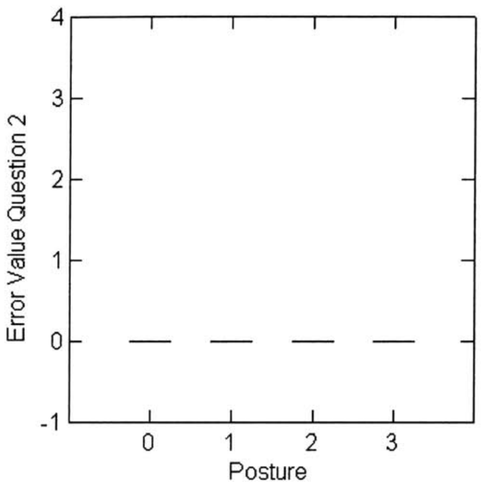


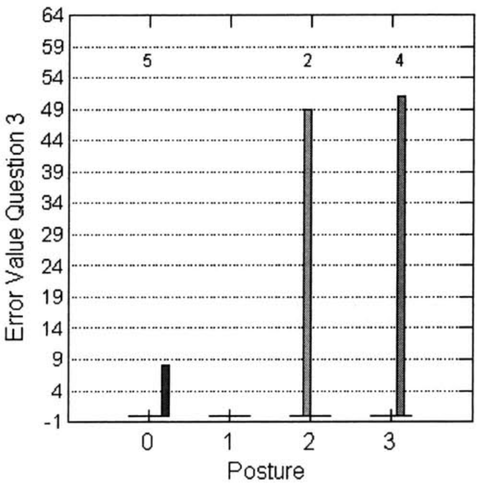
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

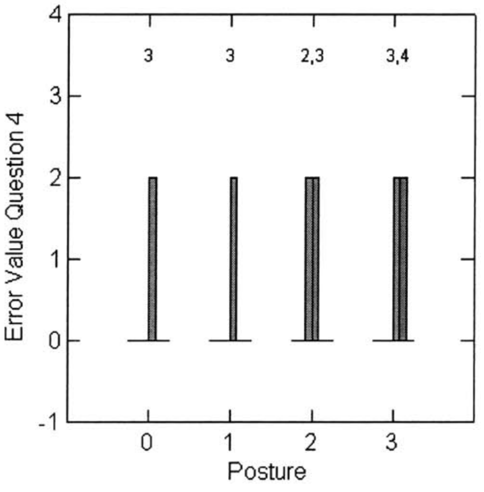
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

SUBJECT 4 (CONTROL GROUP)





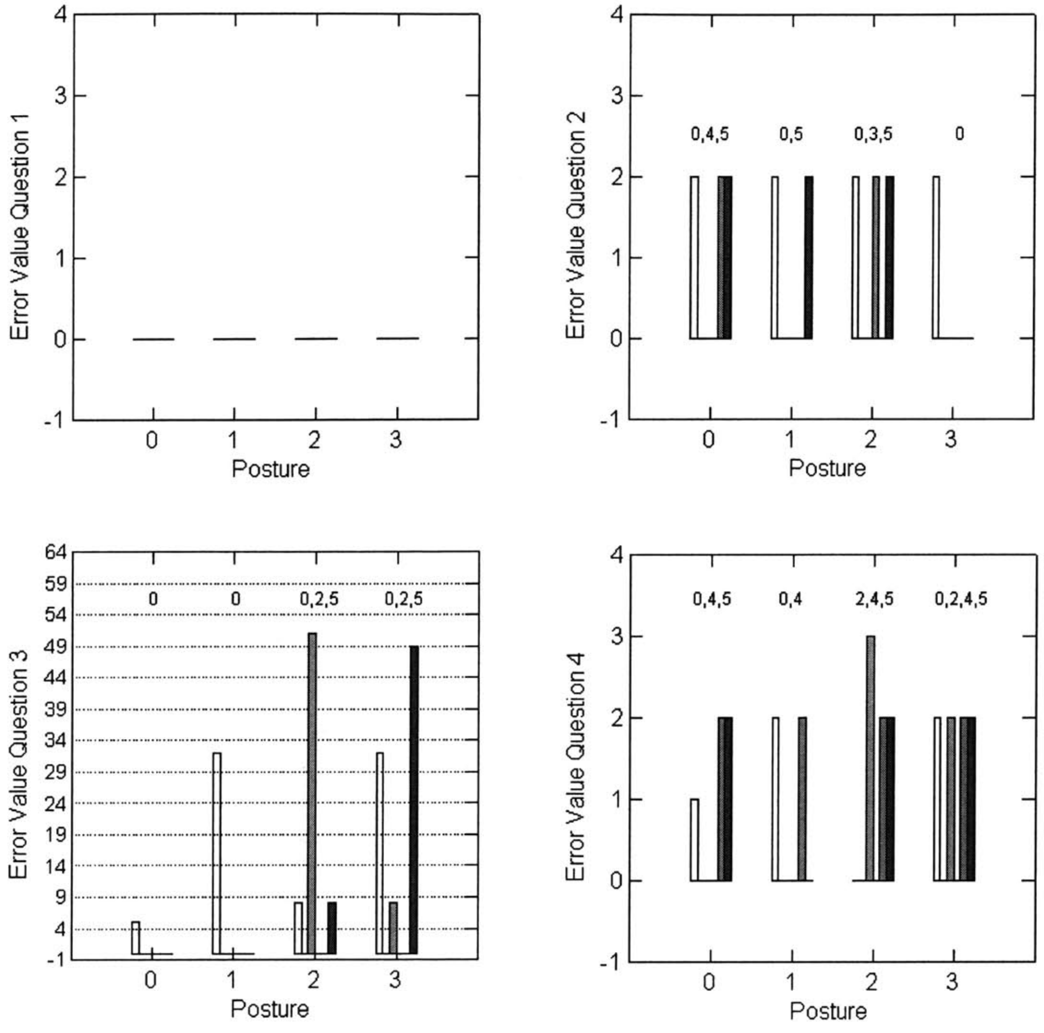




Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

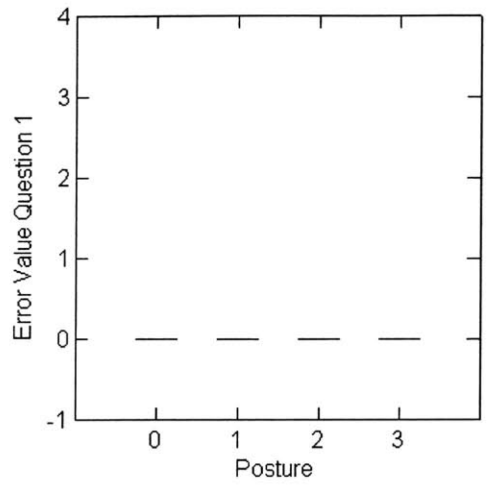
SUBJECT 5 (CONTROL GROUP)

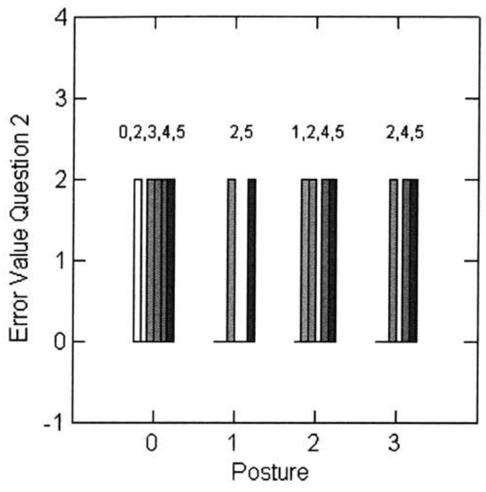


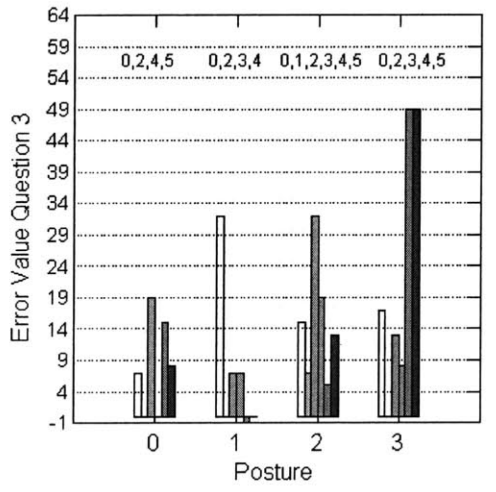
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

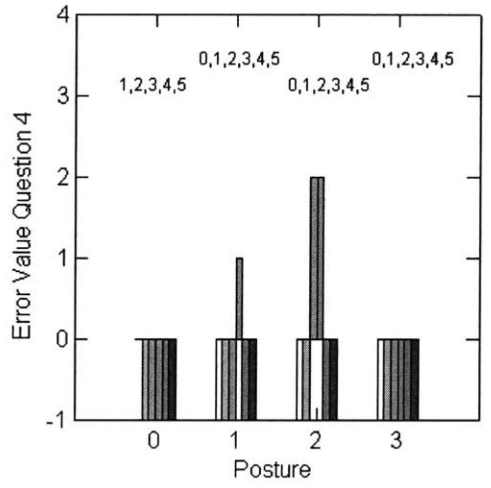
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

SUBJECT 6 (CONTROL GROUP)





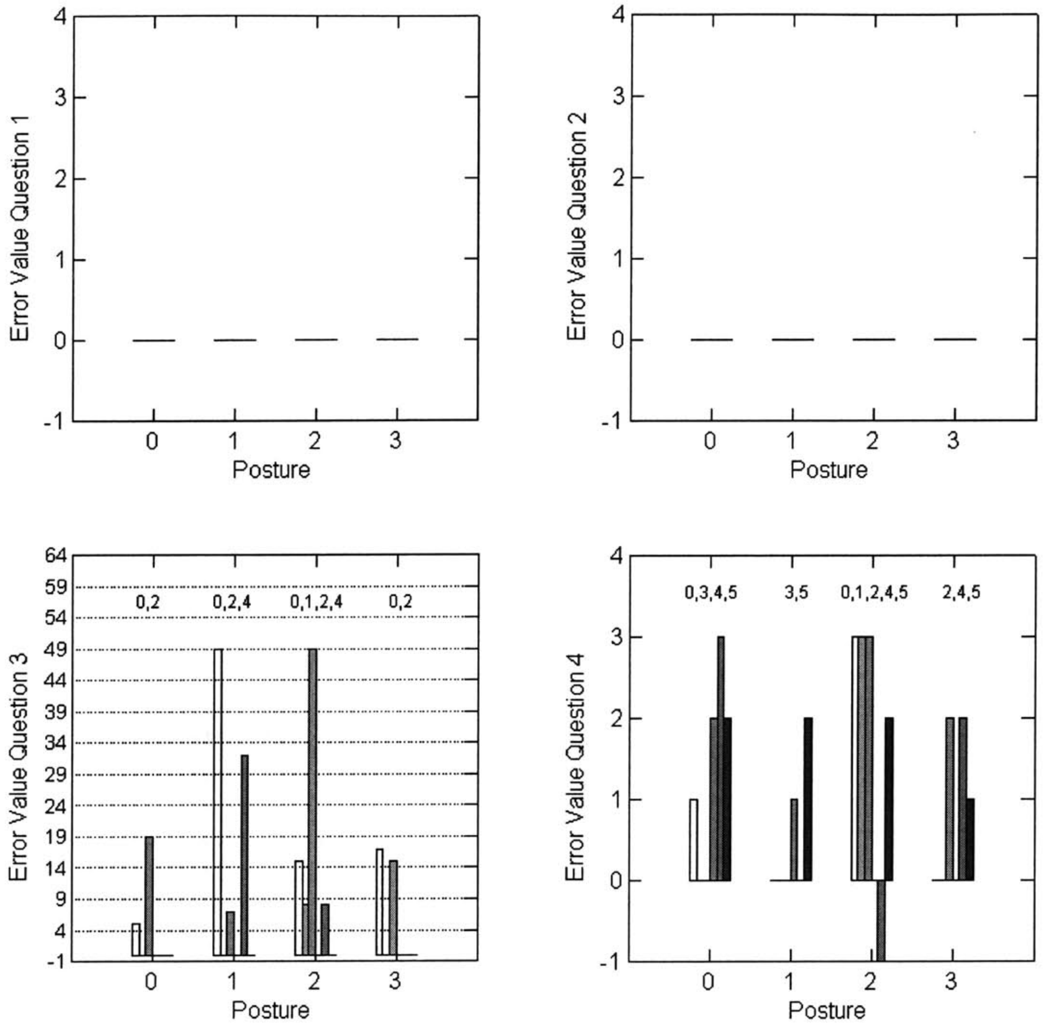




Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

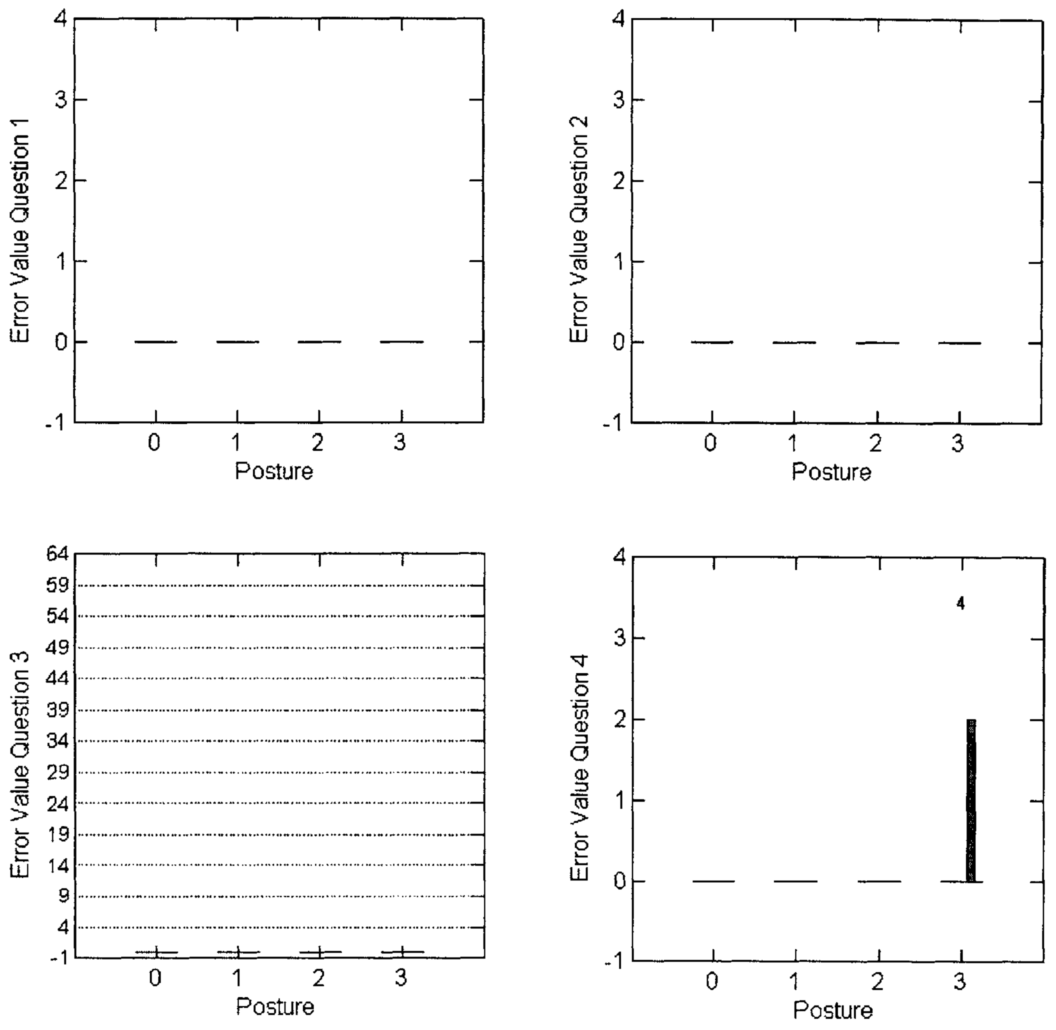
SUBJECT 7 (CONTROL GROUP)



Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

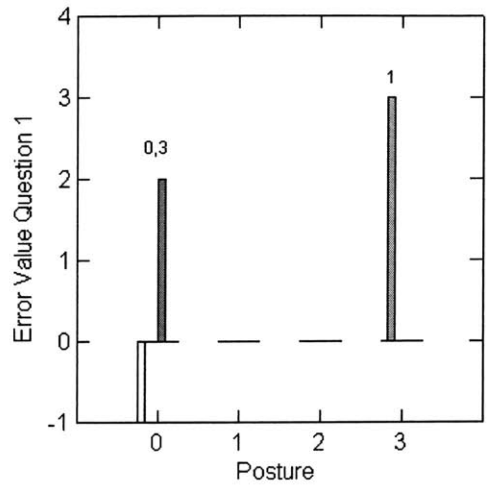
SUBJECT 8 (CONTROL GROUP)

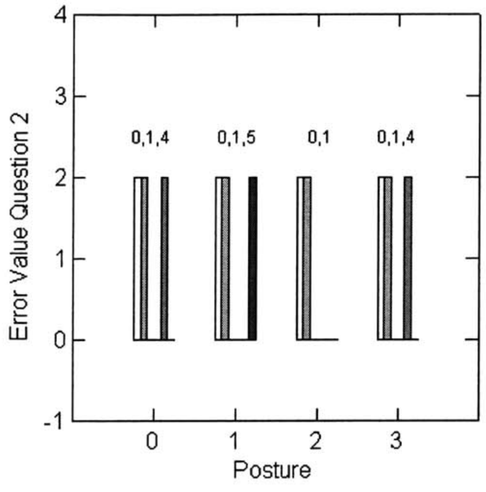


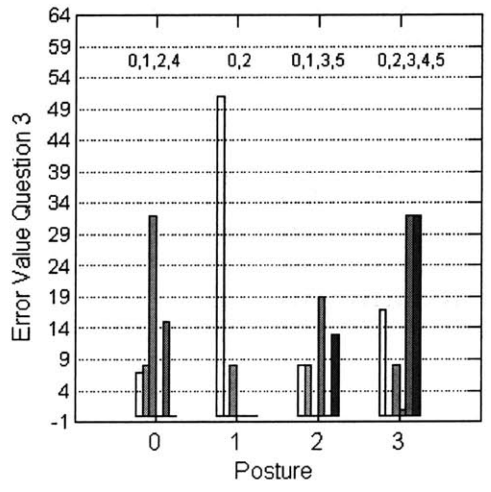
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

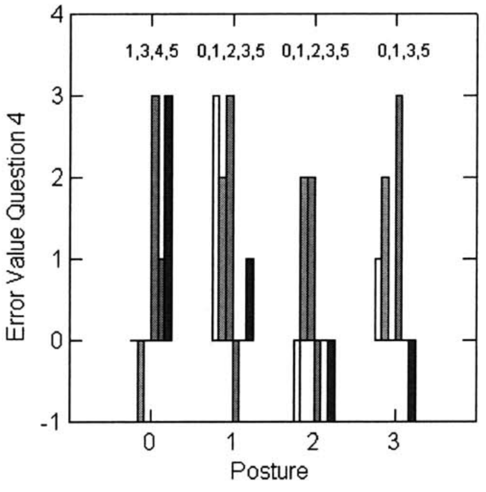
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

SUBJECT 9 (CONTROL GROUP)





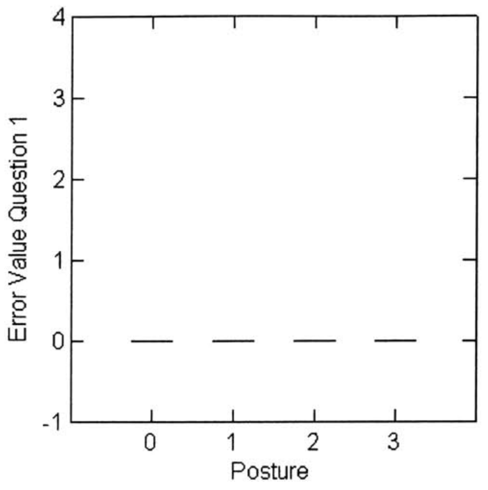


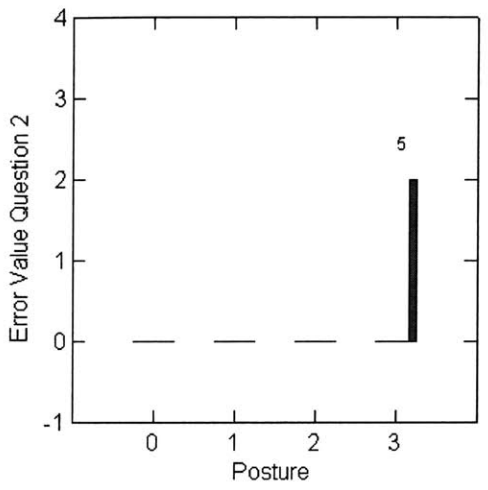


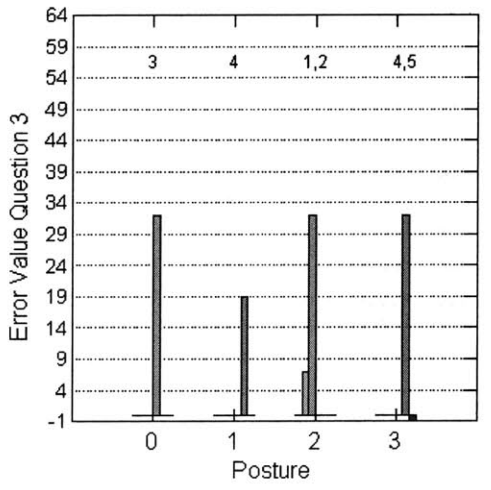
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

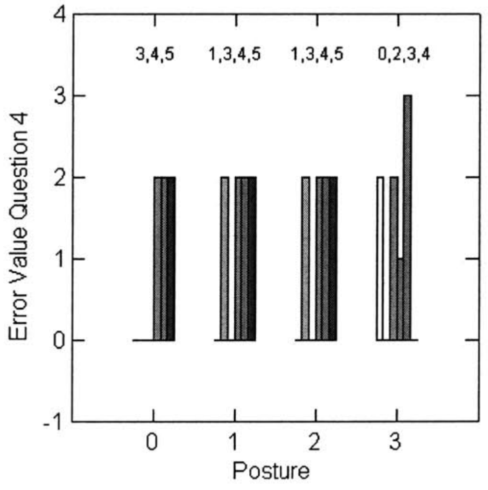
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

SUBJECT 10 (CONTROL GROUP)





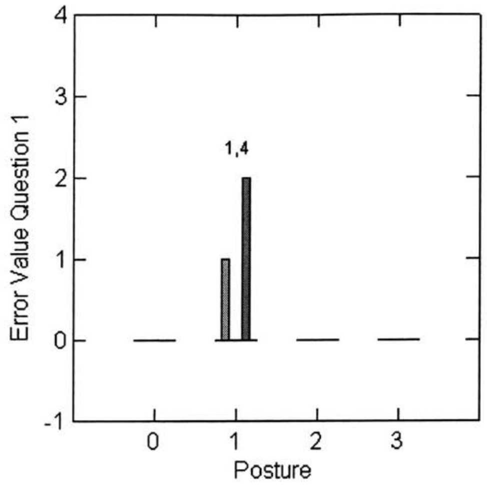


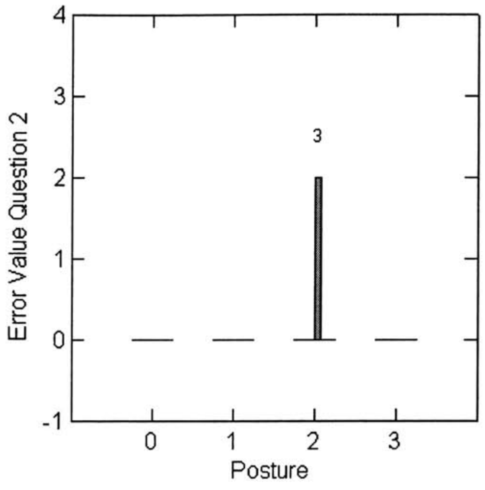


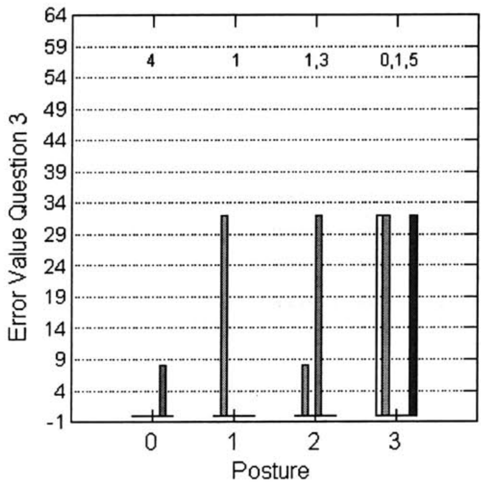
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

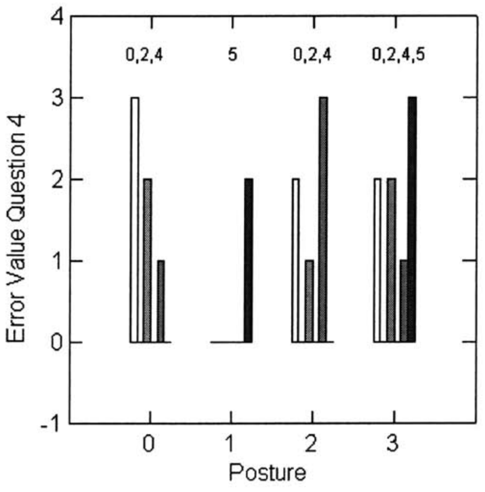
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

SUBJECT 11 (CONTROL GROUP)





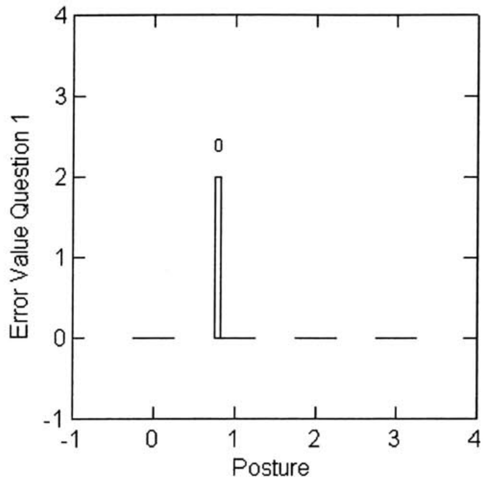


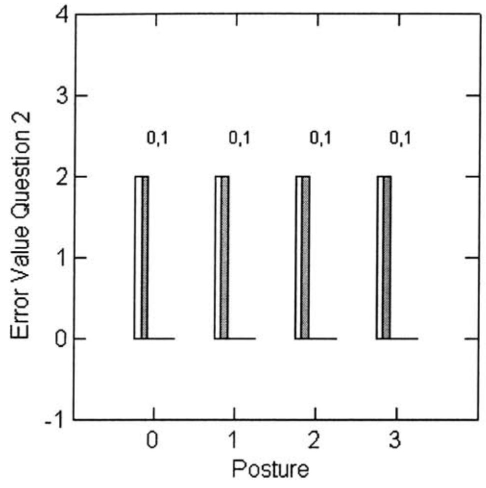


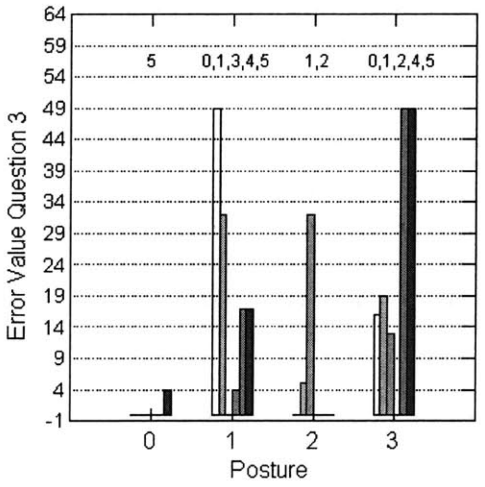
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

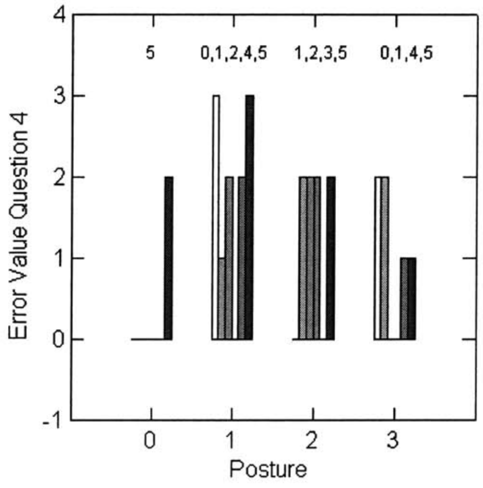
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

Subject 12 (Control Group)





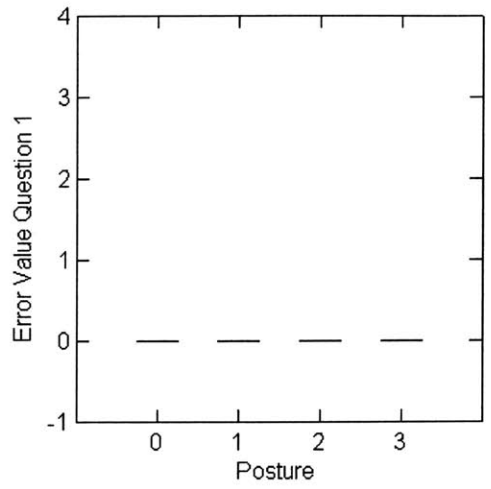


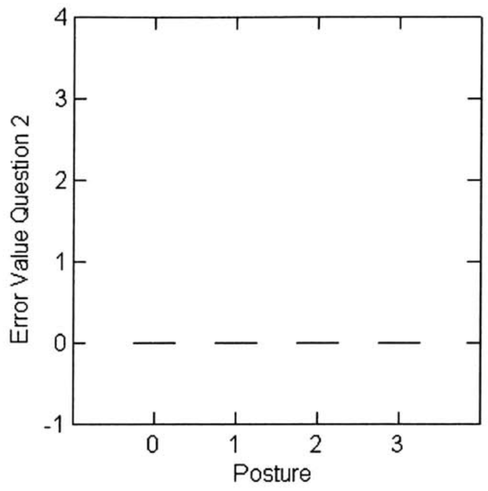


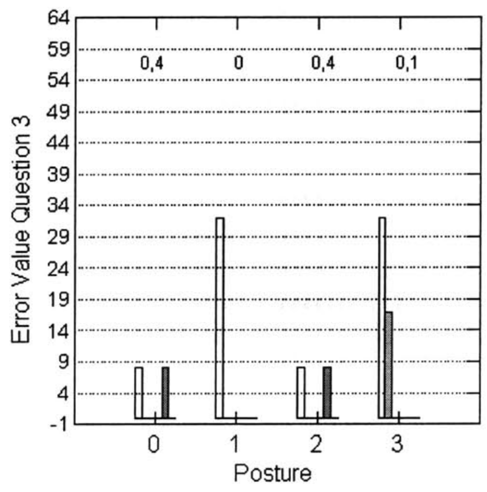
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

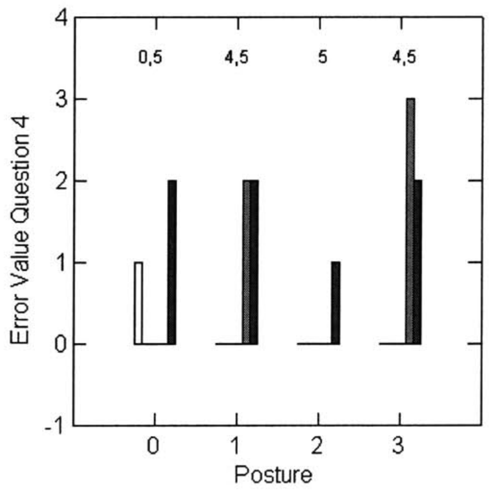
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

SUBJECT 14 (SIM GROUP)





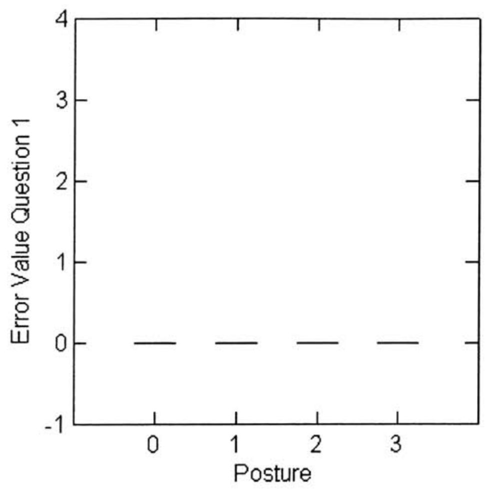


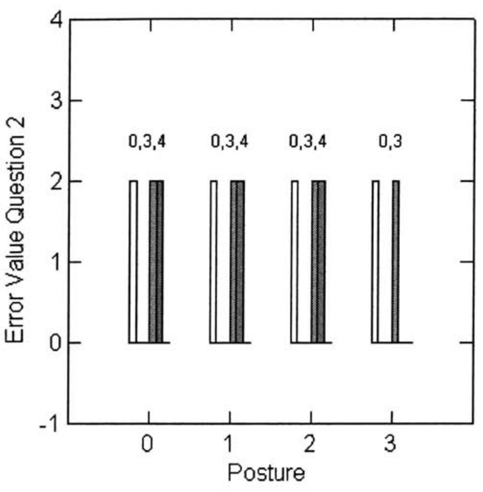


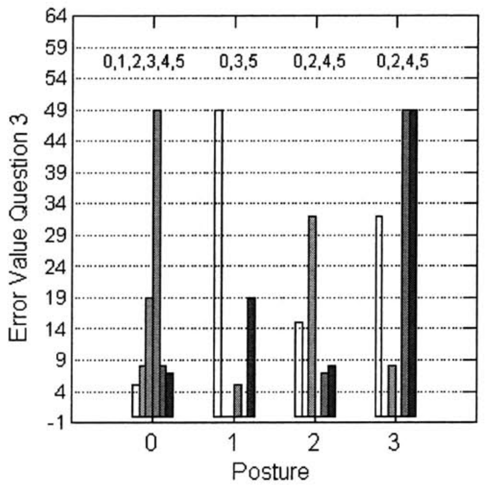
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

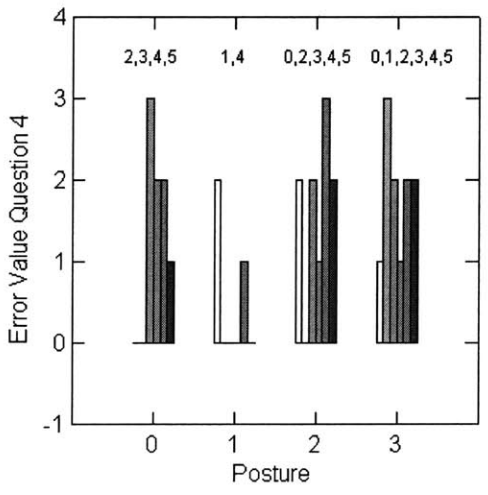
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

SUBJECT 15 (SIM GROUP)





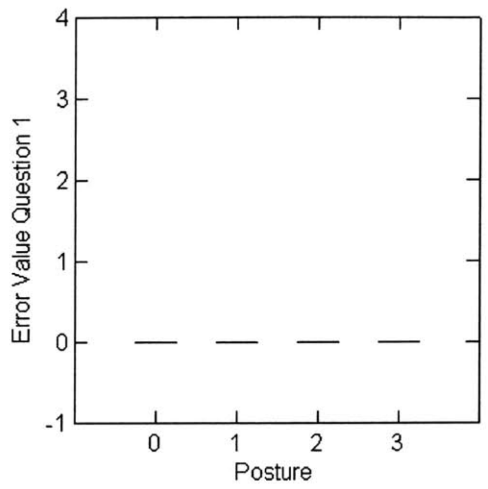


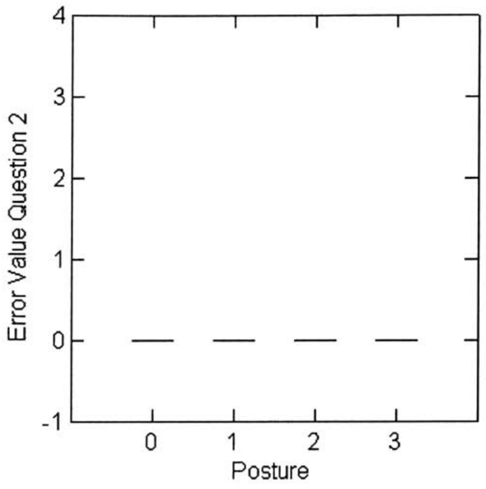


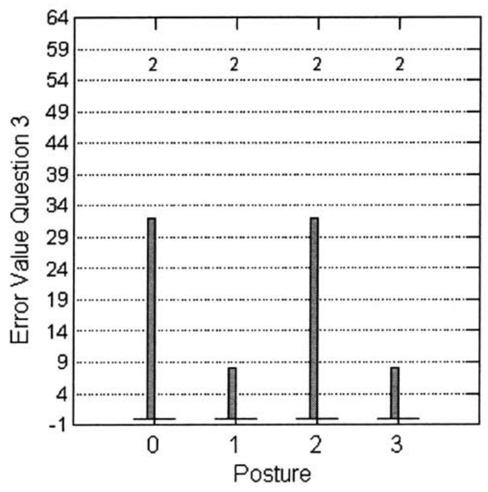
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

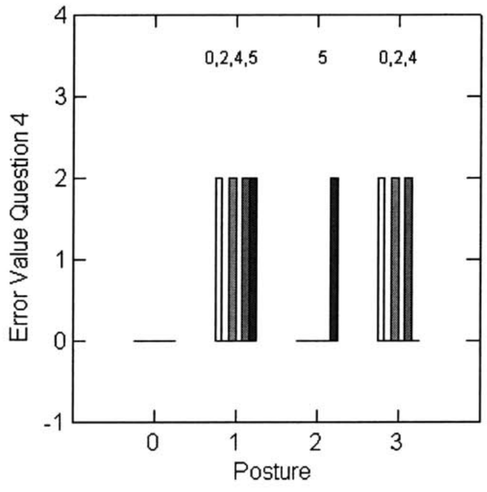
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

Subject 16 (SIM Group)





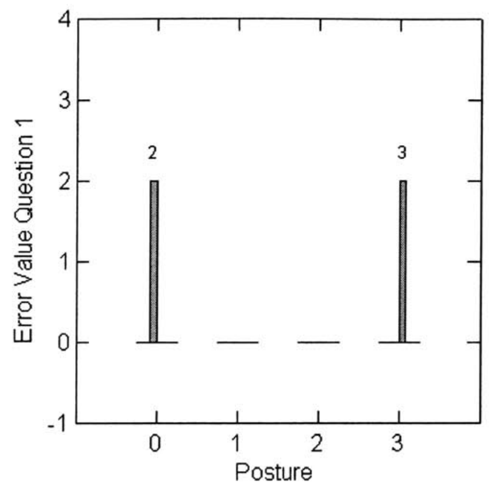


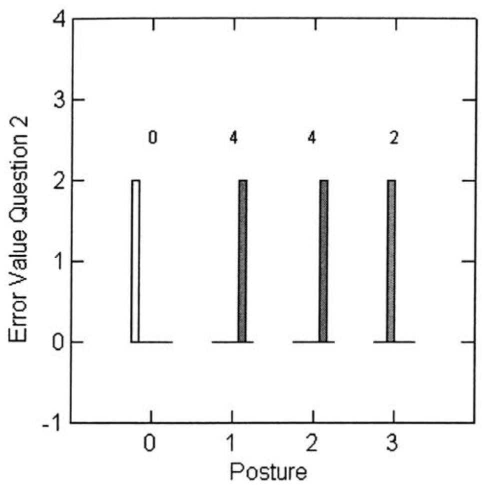


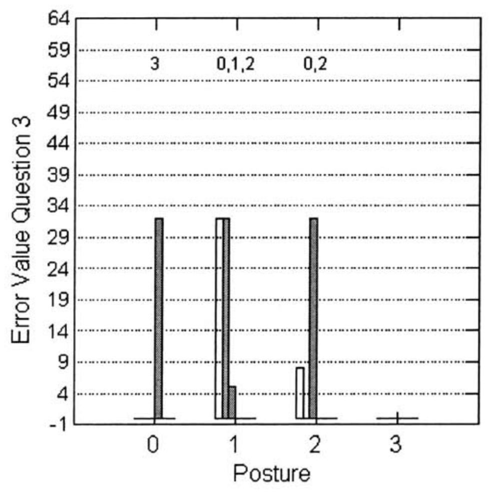
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

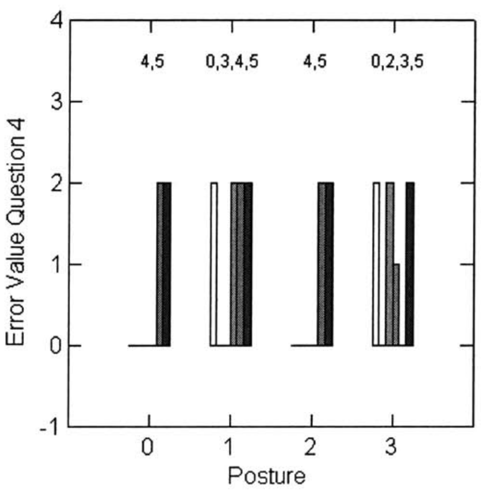
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

SUBJECT 17 (SIM GROUP)





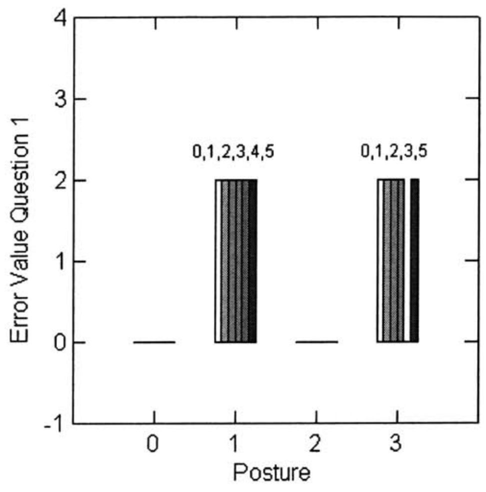


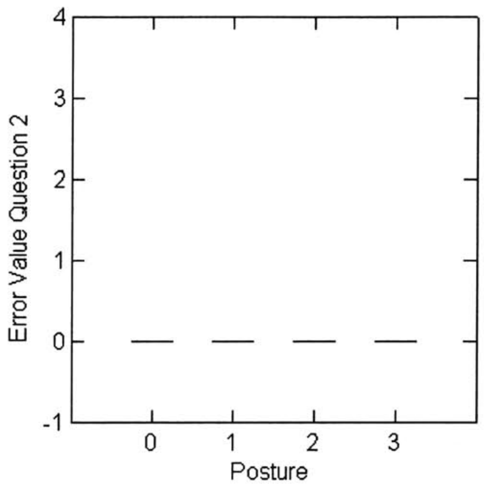


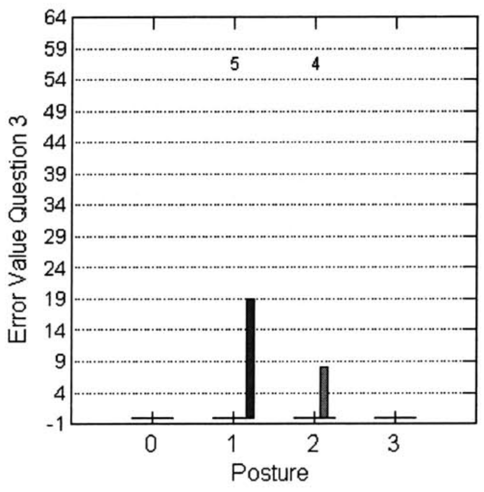
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

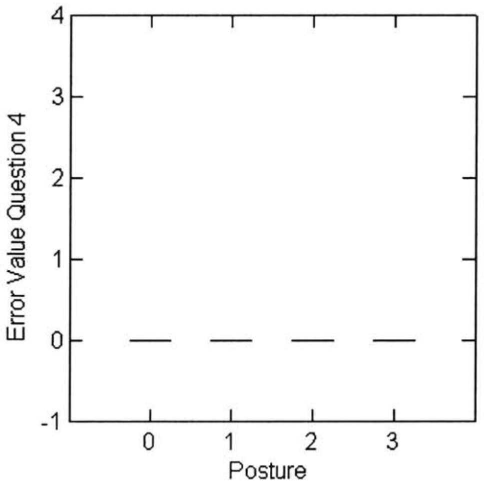
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

SUBJECT 18 (SIM GROUP)





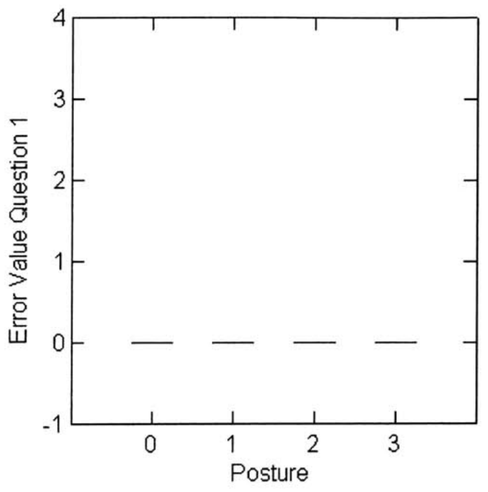


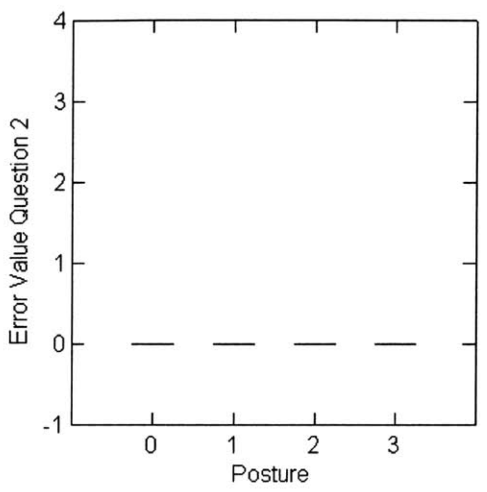


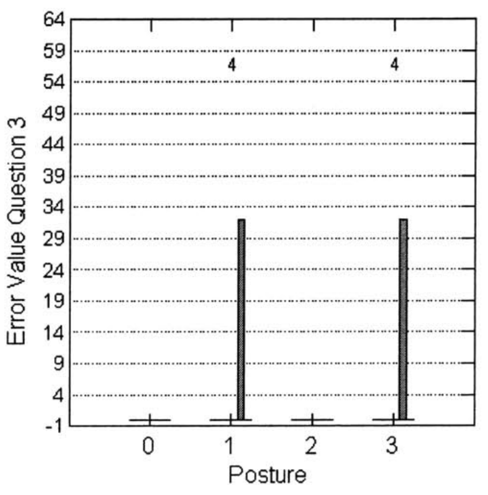
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

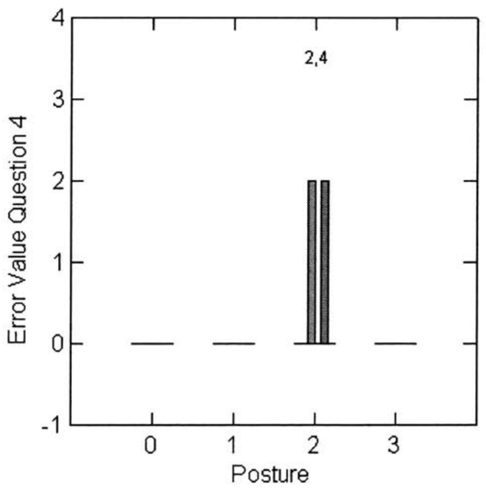
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

SUBJECT 19 (SIM GROUP)





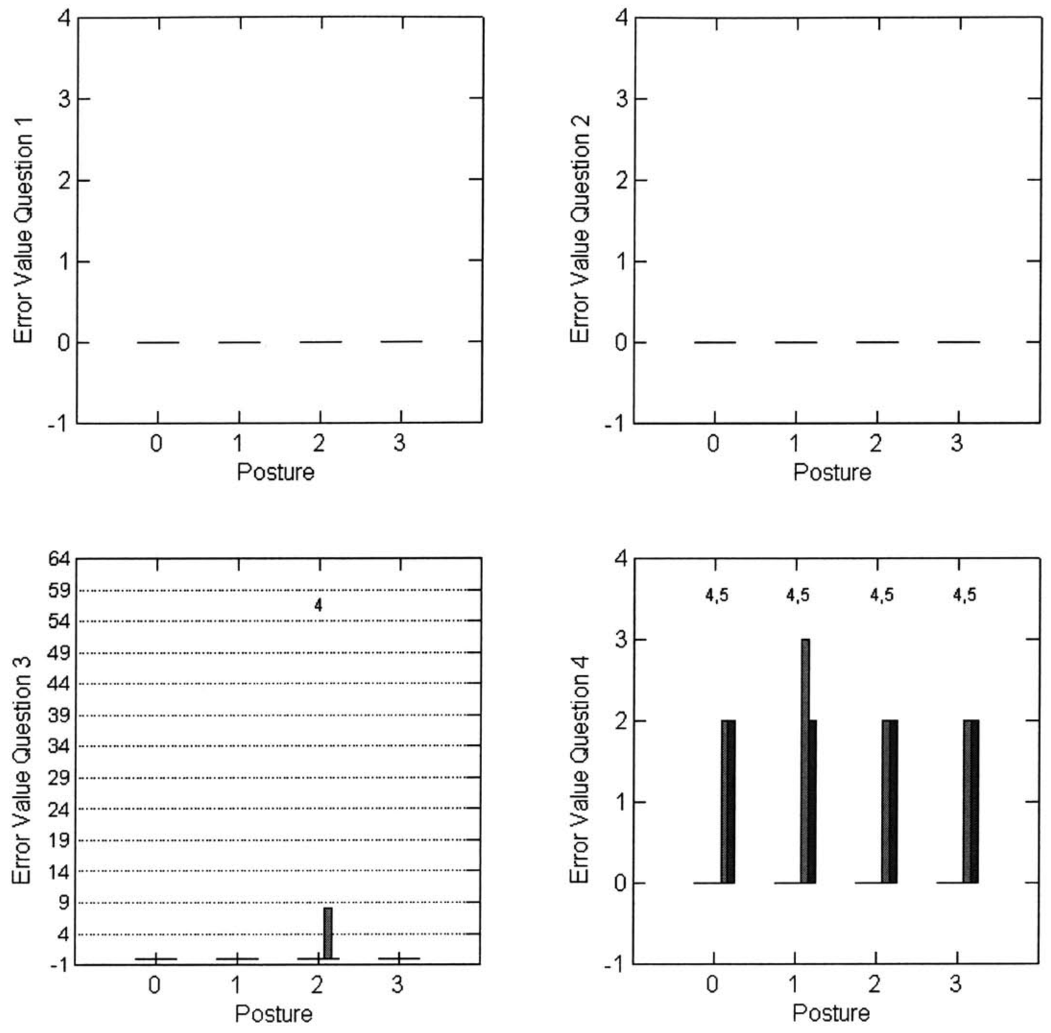




Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

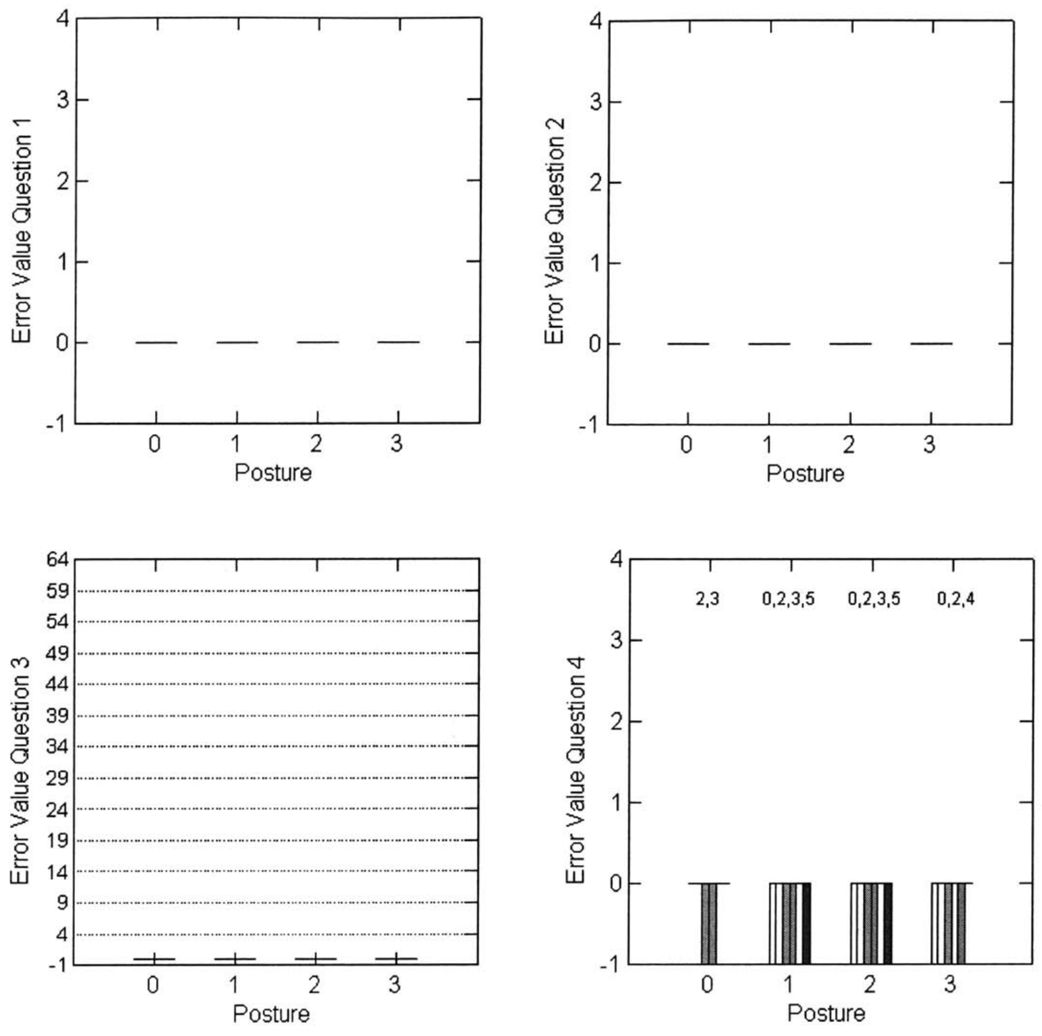
SUBJECT 20 (SIM GROUP)



Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

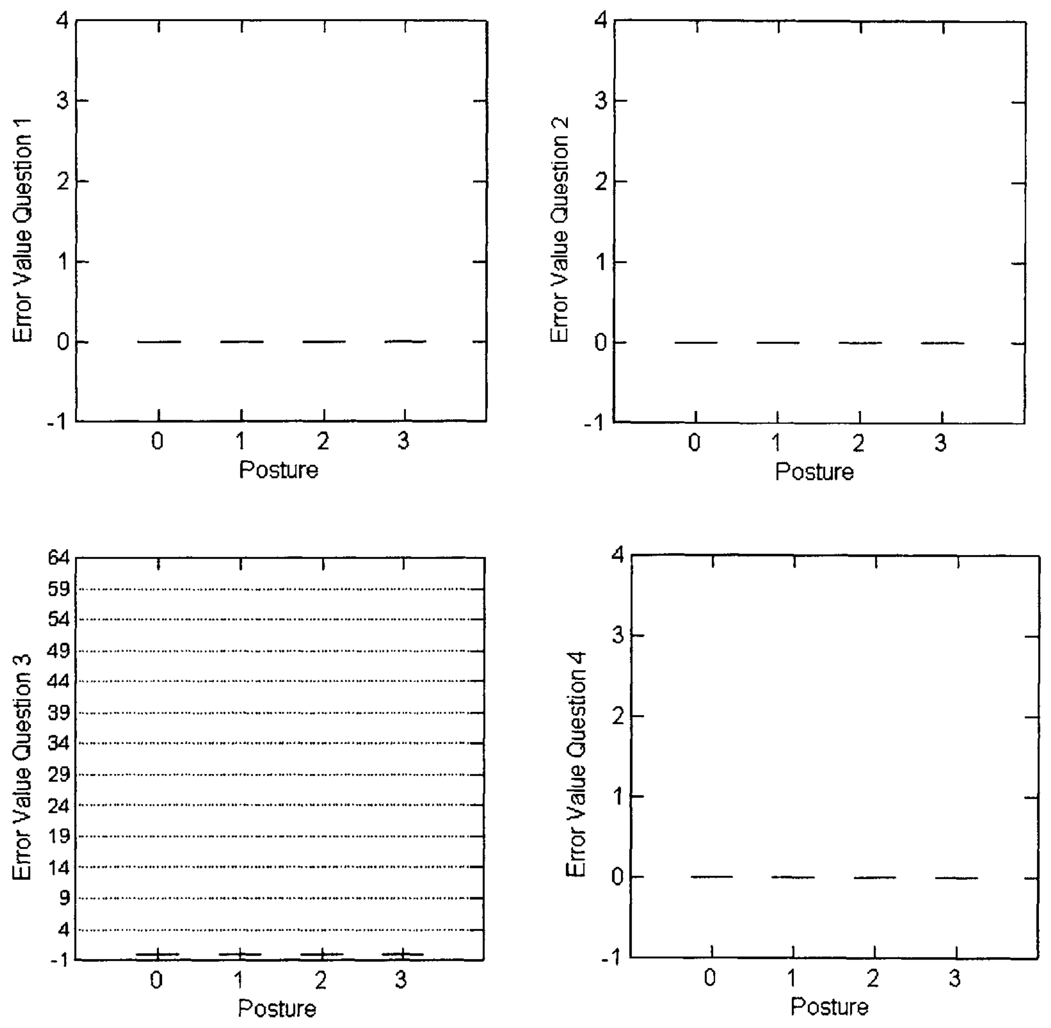
SUBJECT 21 (SIM GROUP)



Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

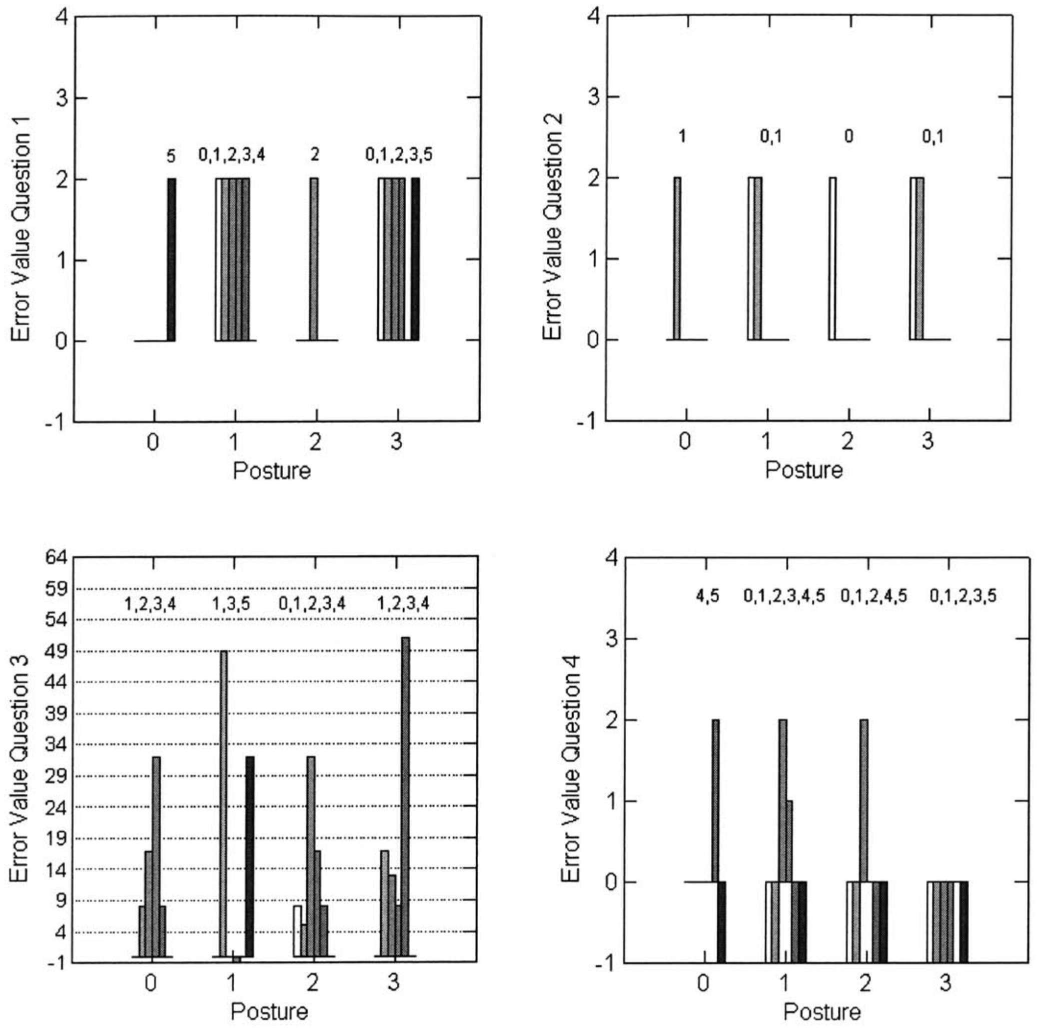
SUBJECT 22 (SIM GROUP)



Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

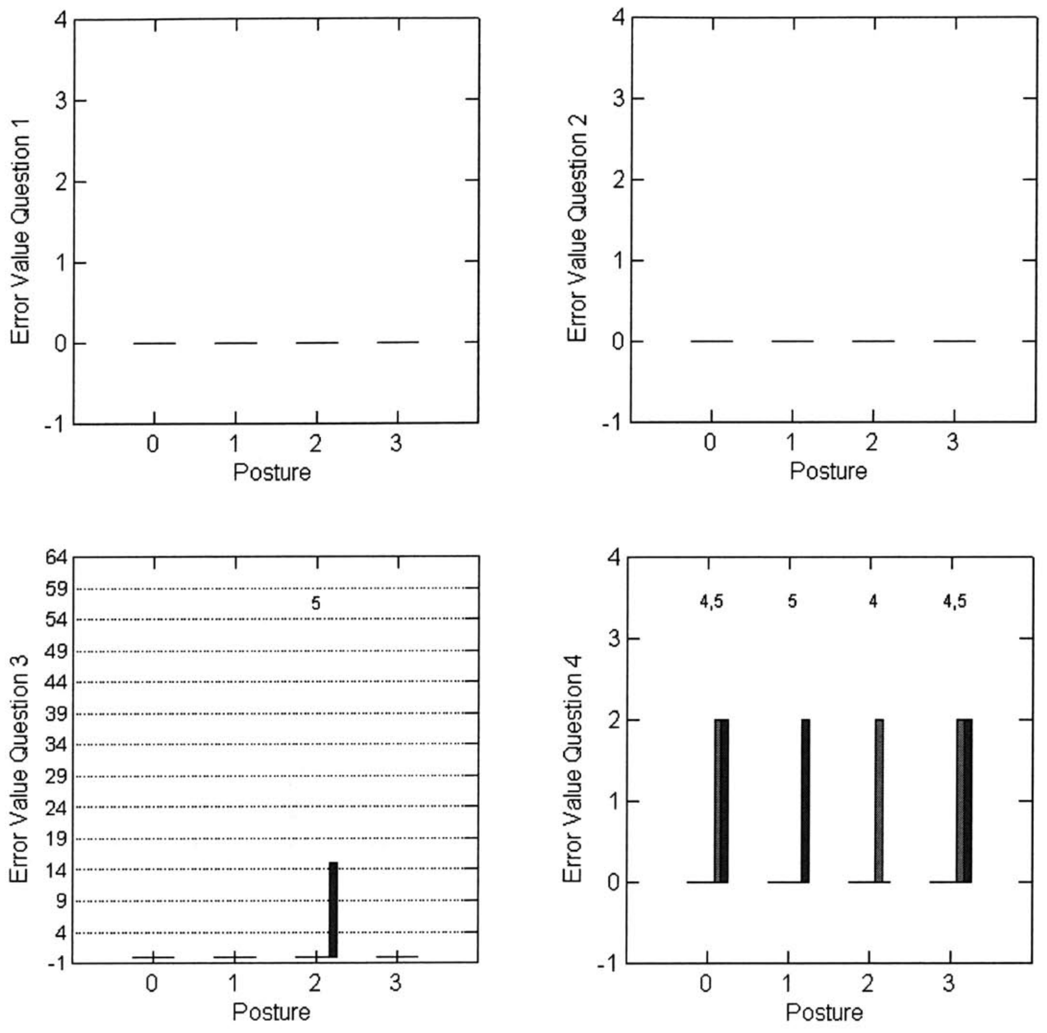
SUBJECT 23 (SIM GROUP)



Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

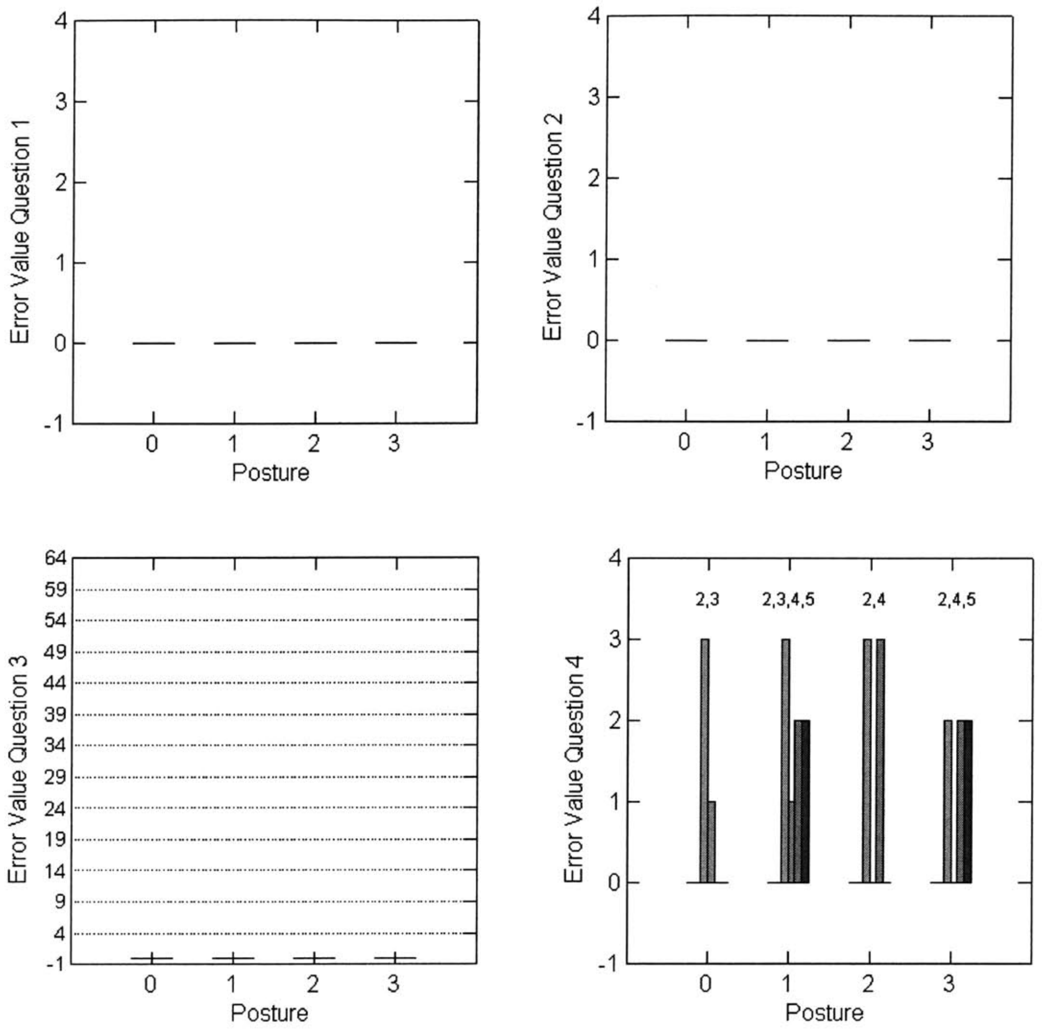
SUBJECT 24 (SIM GROUP)



Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

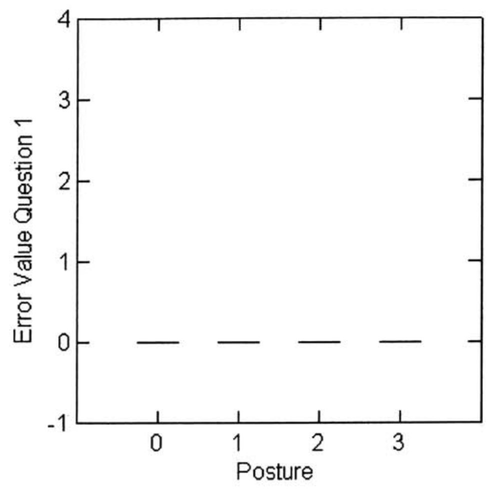
SUBJECT 25 (SIM GROUP)

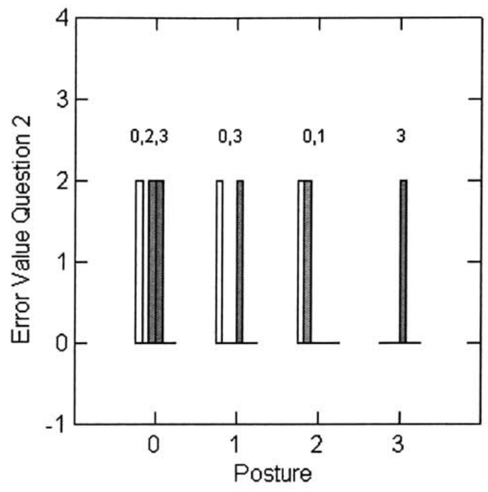


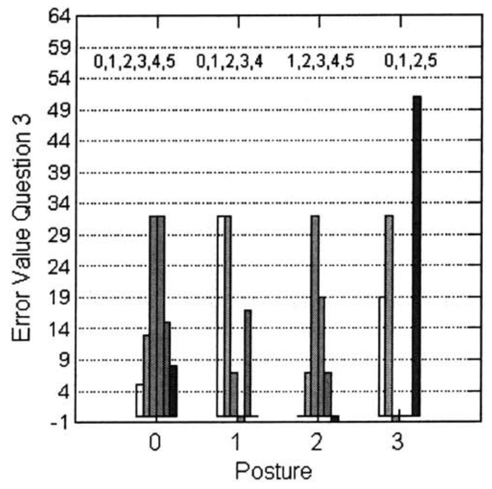
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

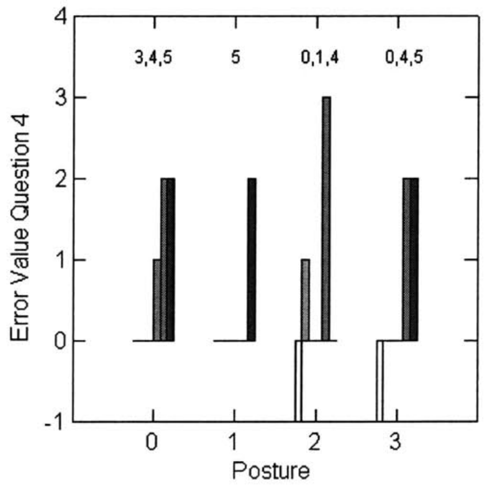
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

SUBJECT 26 (SIM GROUP)





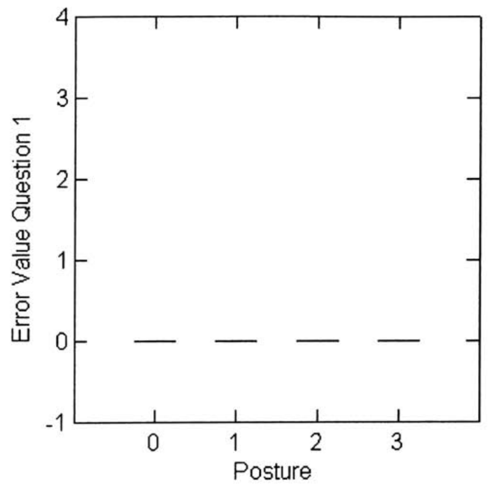


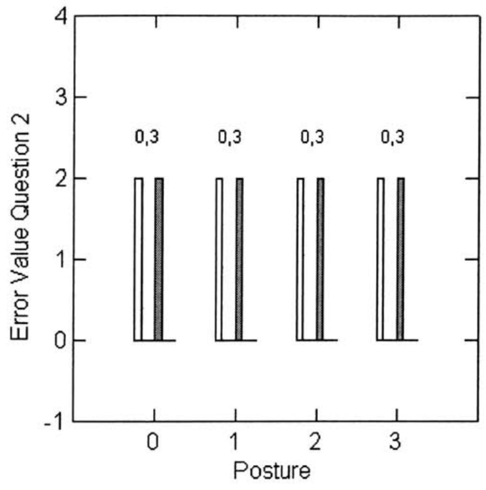


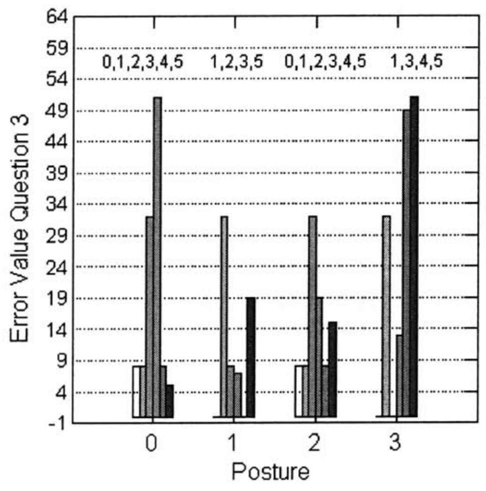
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

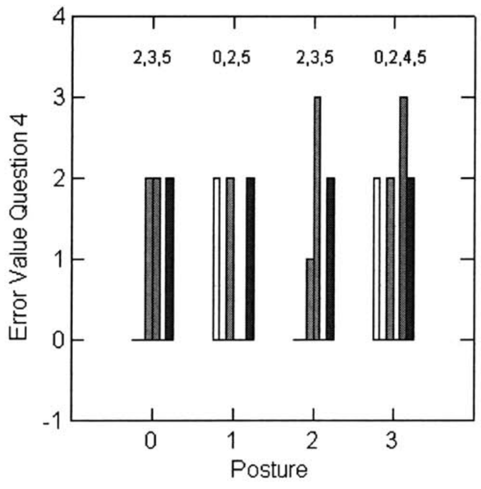
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

SUBJECT 27 (SIM GROUP)









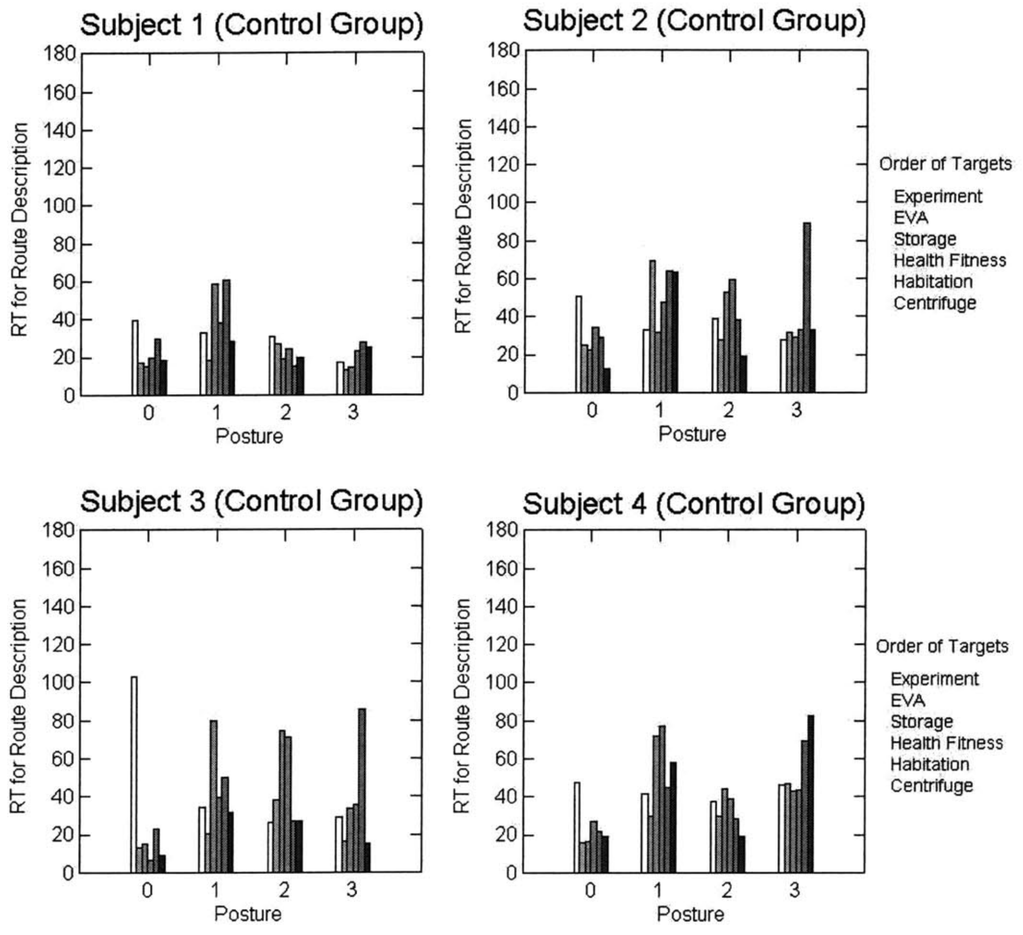
Postures: 0, upright; 1, left shoulder down; 2, upside down; and 3, right shoulder down.

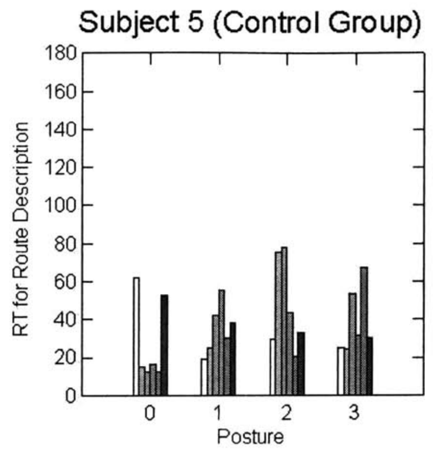
Order of Targets: 0, Experiment; 1, EVA; 2, Storage; 3, Health Fitness; 4, Habitation; 5, Centrifuge.

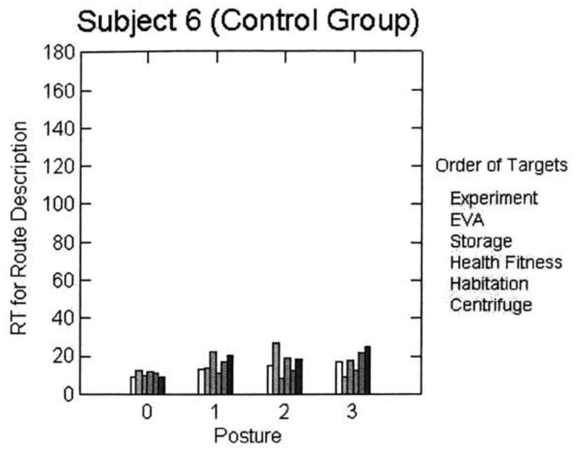
RESPONSE TIMES FOR ROUTE DESCRIPTION QUESTIONS

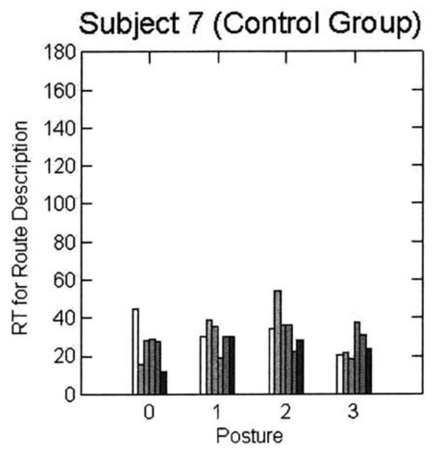
Postures

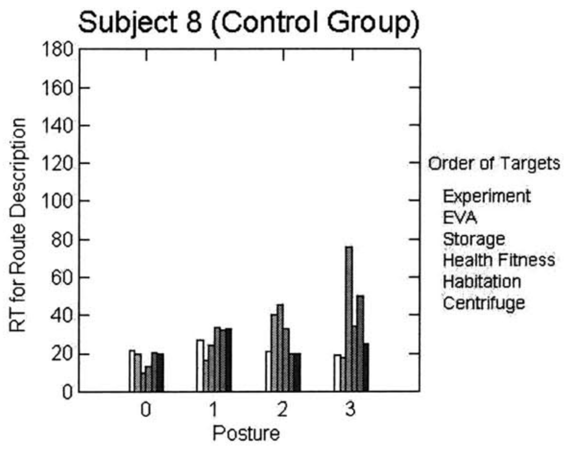
1: Upright; 2: Right Shoulder Down; 3: Upside Down; 4: Left Shoulder Down.

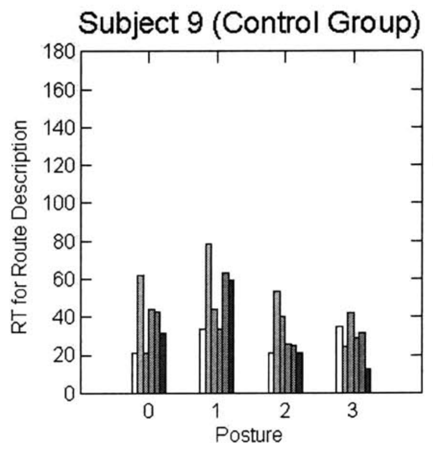


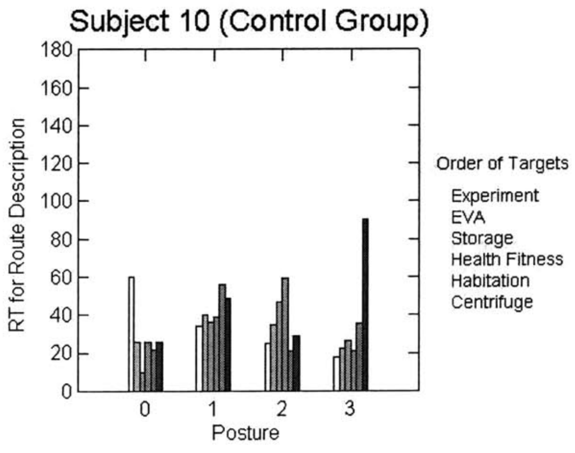

Subject 5 (Control Group)
RT for Route Description
0
20
40
60
80
100
120
140
160
180
0
1
2
3
Posture

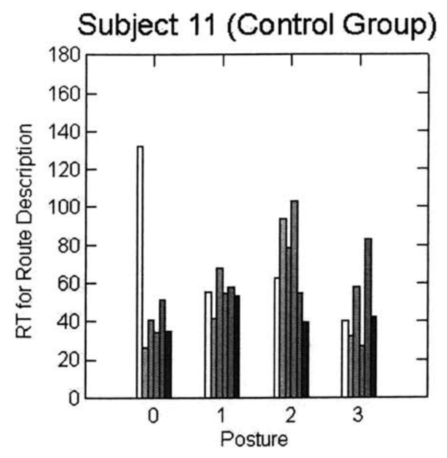

Subject 6 (Control Group)
RT for Route Description
0
20
40
60
80
100
120
140
160
180
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

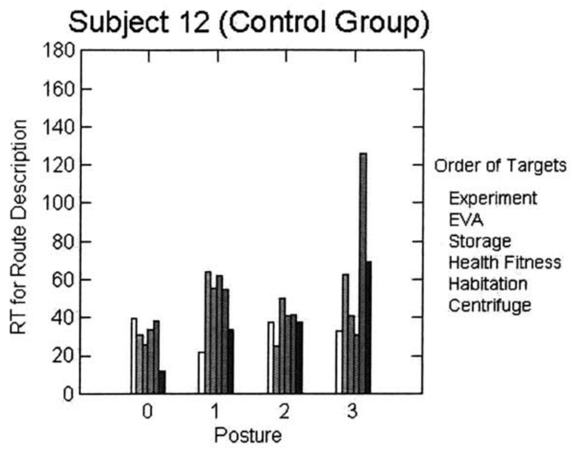

Subject 7 (Control Group)
RT for Route Description
0
20
40
60
80
100
120
140
160
180
0
1
2
3
Posture

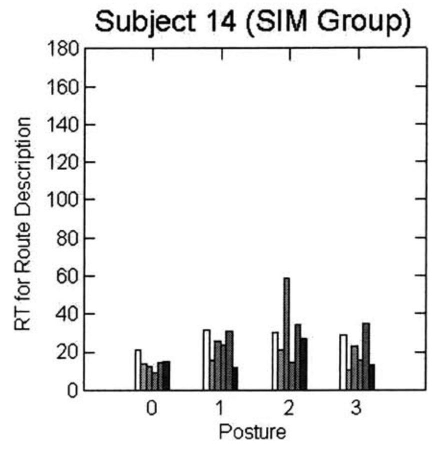

Subject 8 (Control Group)
RT for Route Description
0
20
40
60
80
100
120
140
160
180
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

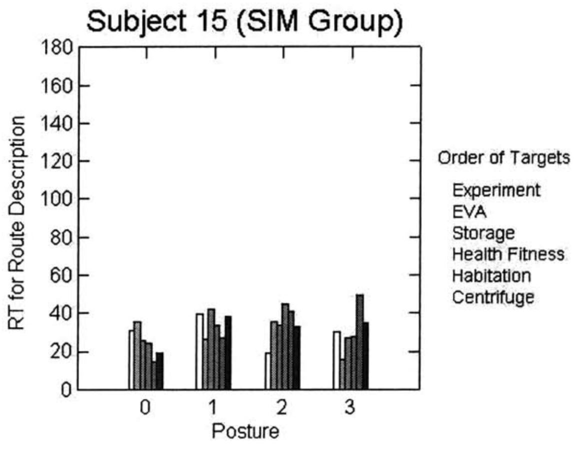

Subject 9 (Control Group)
RT for Route Description
0
20
40
60
80
100
120
140
160
180
0
1
2
3
Posture

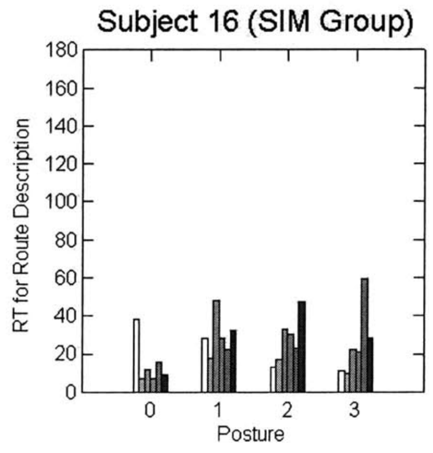

Subject 10 (Control Group)
RT for Route Description
0
20
40
60
80
100
120
140
160
180
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

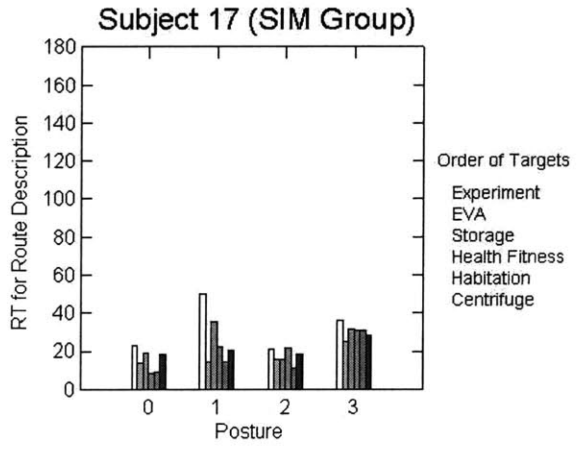

Subject 11 (Control Group)
RT for Route Description
Posture
0
1
2
3
0
20
40
60
80
100
120
140
160
180

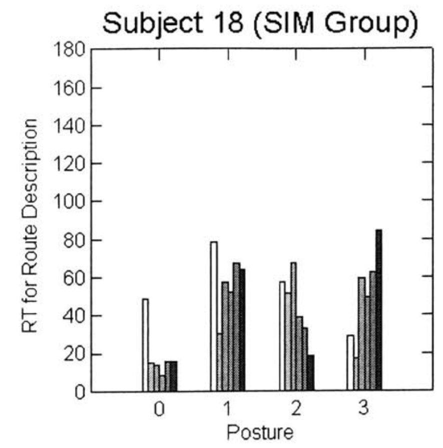

Subject 12 (Control Group)
RT for Route Description
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

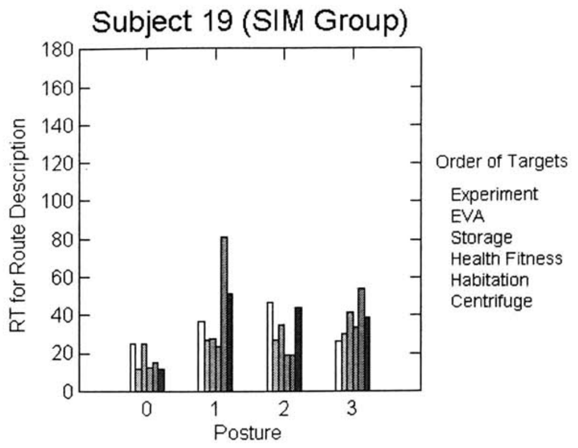

Subject 14 (SIM Group)
RT for Route Description
Posture

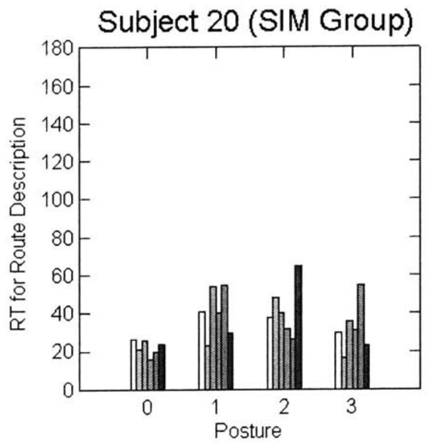

Subject 15 (SIM Group)
RT for Route Description
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

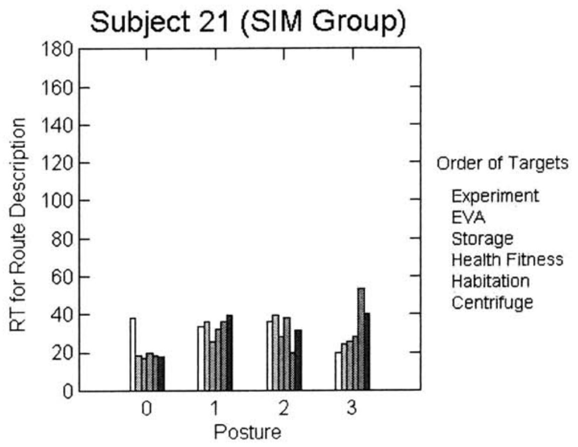

Subject 16 (SIM Group)
RT for Route Description
Posture

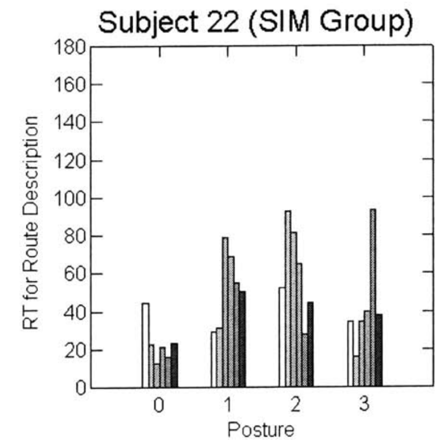

Subject 17 (SIM Group)
RT for Route Description
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

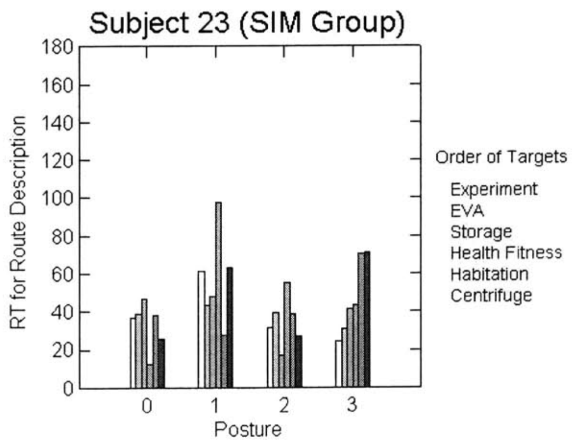

Subject 18 (SIM Group)
RT for Route Description
Posture
180
160
140
120
100
80
60
40
20
0
0
1
2
3

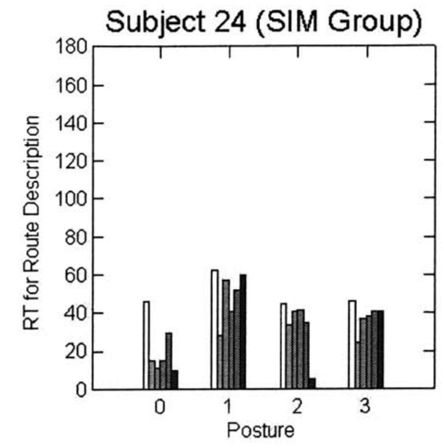

Subject 19 (SIM Group)
RT for Route Description
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

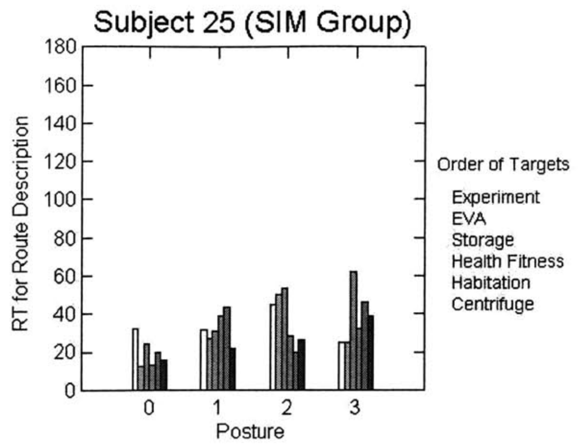

Subject 20 (SIM Group)
RT for Route Description
Posture

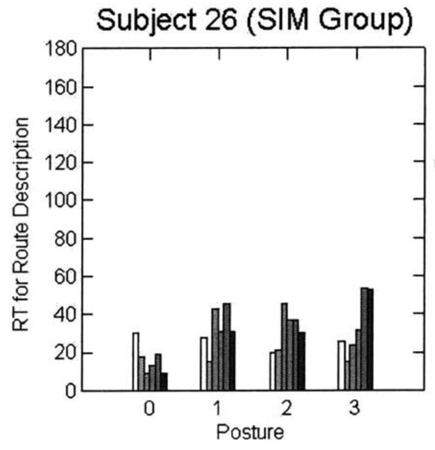

Subject 21 (SIM Group)
RT for Route Description
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge

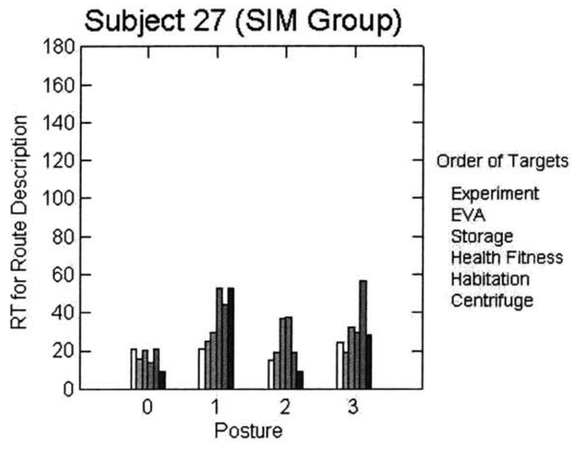

Subject 22 (SIM Group)
RT for Route Description
Posture


Subject 23 (SIM Group)
RT for Route Description
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge


Subject 24 (SIM Group)
RT for Route Description
180
160
140
120
100
80
60
40
20
0
0
1
2
3
Posture


Subject 25 (SIM Group)
RT for Route Description
180
160
140
120
100
80
60
40
20
0
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge


Subject 26 (SIM Group)
RT for Route Description
180
160
140
120
100
80
60
40
20
0
0
1
2
3
Posture


Subject 27 (SIM Group)
RT for Route Description
180
160
140
120
100
80
60
40
20
0
0
1
2
3
Posture
Order of Targets
Experiment
EVA
Storage
Health Fitness
Habitation
Centrifuge