Augmented Reality Navigation System for Human Traversal of Rough Terrain

by

Nicholas J. Anastas

B.S. Computer Science, University of Illinois at Urbana-Champaign
(2009)

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

September 2020

Author . . . .
Department of Aeronautics and Astronautics
August 18, 2020

Certified by. . . . .
Jeffrey A. Hoffman
Professor of the Practice of Aeronautics and Astronautics
Thesis Supervisor

Accepted by . . . .
Zoltan Spakovszky
Professor, Aeronautics and Astronautics
Chair, Graduate Program Committee

Augmented Reality Navigation System for Human Traversal of Rough Terrain

by

Nicholas J. Anastas

Submitted to the Department of Aeronautics and Astronautics
on August 18, 2020, in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Aeronautics and Astronautics

Abstract

At some point in the near future, astronauts will resume the decades dormant practice of planetary EVA. Because of the resource scarcity of extra-terrestrial environments, finding and following efficient paths during EVA will be essential to mission success. MIT's Surface Exploration Traverse Analysis and Navigational Tool (SEXTANT) was created specifically to satisfy this path efficiency need. Prior to this work, SEXTANT was used exclusively during the planning phase of a mission. Results from several field studies found that SEXTANT's path planning capabilites could be useful during the execution phase of a mission as well; information gathered during a traverse would sometimes necessitate that an existing path be recalculated. Because of its long algorithm runtime and stateless nature, however, the previous implementation of SEXTANT was unable to support this capability.

This work details the software optimizations and architectural modifications made to SEXTANT in order to accommodate real-time, mid-traverse path recalculations based on terrain information updates supplied by a user. Additionally, this work discusses the creation of *Pathfinder*: an augmented reality tool that interfaces with SEXTANT to provide path following, terrain modification, and path recalculation capabilities. *Pathfinder* implements three different path visualizations, each of which is evaluated on how it affects traverse performance. Furthermore, *Pathfinder* implements two different interfaces that allow users to modify terrain information and to recalculate existings paths based on those modifications. These interfaces are compared and used to evaluate the effect mid-traverse path recalculation has on traverse performance.

Acknowledgements

I am a man of many talents. Or at least of several talents. Certainly more than a few.

Unfortunately – for me, and also for the other souls who have witnessed or been a part of my journey – composing a Master's thesis is not one of them.

It has taken the combined effort of my advisors, my friends, my classmates, and my family to assemble this jumble of words into an order that hopefully resembles something meaningful. Regardless, after only a bit of pushing, some pulling, the occasional forceful urging, a great many judgmental stares, a healthy portion of love and kindness, and a stream of unending support, I am finally finished.

Some thanks are in order.

Tristan Endsley: Well, we did it. I don't know if you taught this old dog any new tricks, but we finally got those words on pages. Thanks so much for always being supportive and for encouraging me to find a thesis topic that was 'my own', never forcing me down some path that you didn't think I would enjoy. For the record, I would've been okay with a little more forcing. I don't know how many times I forgot lessons you taught me or advice that you gave, but I appreciate that you were always so positive when you inevitably had to teach and give them again. I'm not sure whether or not all of those whiteboarding sessions improved my thesis, but they sure did make me feel better.

Jeff Hoffman: Thank you for taking a chance on me three years ago. I had no research experience and had been out of school long enough to forget nearly all of my calculus (which only takes about 6 months anyways), but for some reason you decided to take me in and advise me. In retrospect, it certainly would have been nice to have some of that missing research experience. Thanks for all of the advice you've offered me, and for being patient as I meandered around a research topic. Particularly, thanks for telling me that I'd done a good job after you'd used *Pathfinder* for the first time. That meant a lot.

Mort: Thank you for keeping things in perspective. Sometimes snuggles really are all you need.

Nathan: Though evidence from family videos may indicate otherwise, I have always felt like you were a good friend. Even though I was the annoying little brother, you never seemed to have a problem with me hanging around. And then as we got older, you never seemed to have a problem with me stealing all of your ideas. If *Earth's Fate* taught us anything, it's that I have no problem with a little plagiarism. Almost all of my hobbies and interests can be traced back to you (except maybe biking, you really do seem to hate that) – so thanks for picking some good ones. It could be argued that your love of Star Wars has had more influence on the course of my life than any other single thing. Thanks for being a great and inspiring older brother.

Kristen: Thanks for putting up with all of the times where I lamented the work I had to do, then proceeded to not do that work. Thanks for always being around to talk me off the ledge when I was really uncertain or confused about the choices I'd made. I know I can be stubborn, and I

know I don’t always listen as well as I should, but I’m so glad that we’ve been together these past few years. All of the meals we’ve cooked together, runs we’ve gone on, games we’ve played, trips we’ve taken, shows we’ve watched – all of these things have kept me sane and even happy. I’m really looking forward to whatever adventure next awaits us.

Mom: Thanks for teaming up with dad to give Nathan and I the best childhood any two boys could ever hope for. The lowest low my kid-self experienced was probably that pile of birthday donuts – but even that is a retrospective, comedic low. I remember actually loving that doughy pyramid (and the bowling that went along with it). If that’s as bad as it ever got, you guys must’ve done something right. I don’t recall a time where you weren’t supportive of whatever I was doing with myself, and I really appreciate that. I’ve never felt confused or ashamed about any aspect of myself – you and dad really did just let me grow into whatever person I thought was best. Or maybe you did some behind-the-scenes marionetting and I just didn’t realize. If that’s the case, don’t let me know – leave me to my happy ignorance. In any case, I’m looking forward to all the future adventures that you, Nathan, Ashley, Kristen, and I are going to enjoy together.

DAD: I am thankful for everything you ever did with me and for me. The bike rides, the coaching, the wrestling “matches” where you would use just one finger, the campouts, the semi-weekly drives to Children’s Hospital, the never-ending (as is tradition) games of Axis & Allies, the Frisbee tossing with throws that you “invented”, the sessions of Goldeneye where you would camp on the body armor, the vacations, the zero times I ever beat you at racquetball, the shoulder rides… all of it. You always set an excellent example, and even now when I’m confused I’ll try to think about what action you would take and just go with that. I don’t always succeed, but I do try. Everything you did made me into the person I am today, and I hope that that person made you proud. Even through our disagreements, I always respected your mind and your ability to consider the positions of others. Your work ethic, your focus, your commitment to bettering those around you – they will all forever be an inspiration. I’ll do my best to fill those shoes.

Contents

Table of Figures

1. Introduction

1.1 Motivation

In the near future, the National Aeronautics and Space Administration (NASA) plans to send astronauts first back to the surface of the Moon and then for the first time to the surface of Mars (ISECG, 2018). One of NASA's mission priorities while exploring these foreign planetary bodies will be to perform a series of extravehicular activities (EVAs). During these EVAs, astronauts will be tasked with visiting a set of sites that have some either logistical or scientific value (Lim, Abercromby, & Kobs Nowatniak, 2019). Because it is likely that no person will have ever visited these sites, the 'ideal' human-traversable path connecting them will need to be found. Traverse routes will be planned using some combination of remote sensing data and data collected by robotic explorers (e.g. rovers, drones) during a pre-mission site survey (Kara H. Beaton et al., 2019). SEXTANT – a tool developed at MIT – was designed specifically to accomplish this task; that is, to use precursor data to find a least cost path connecting a series of geographic locations (Johnson, Hoffman, Newman, Mazarico, & Zuber, 2010). In the context of SEXTANT, the 'cost' of a path is a complex parameter that is influenced by many variables (e.g. gradient, sun angle) and that actually has several different definitions (e.g. metabolic cost, time cost) (Marquez, 2007). For the purposes of the present discussion, understanding the precise cost formula is not necessary; knowing simply that cost is a value that should be minimized is sufficent.

In past studies, SEXTANT was shown to perform its path-finding task quite well (Norheim, Hoffman, Supervisor, & Balakrishnan, 2018). However, because of long algorithm runtimes and an architecture that discarded all terrain information after a route had been calculated, SEXTANT's path-finding capabilites had only been used durring the planning phase of a mission. Since more accurate terrain data often becomes available when actually walking an EVA route, there was a desire to upgrade SEXTANT to accommodate in situ, real-time terrain updates and path re-planning. EV crewmembers equipped with a version of SEXTANT with these upgrades could avoid incurring unplanned metabolic or time costs by quickly marking and routing around a previously unknown problem area. A tool that allows the EV crew to dynamically re-route from their current location provides the additional benefit of helping the EV crew navigate back to a designated station should they be required to stray from the recommended path (for either science or safety reasons).

Prior to this work, three different display modes have been implemented and tested for use with SEXTANT-created paths: a ‘hard copy’ printed paper map, a wrist-mounted screen displaying an aerial view of the path, and an augmented reality display that presents the path as an overlay on the actual terrain (Anandapadmanaban, Tannady, Norheim, Newman, & Hoffman, 2018). All display modes had useful qualities, though no single mode was without fault. Furthermore, none of the previously-tested display modes allowed for real-time path re-calculation.

1.2 Thesis Contributions

The goal of the research presented here is two-fold:

- Firstly, we have augmented and altered SEXTANT to allow for mid-traverse obstacle demarcation and path re-planning.
- Secondly, we have developed and evaluated a new set of augmented reality user interface implementations that accommodate both path-following and real-time path re-planning.

2. Background

2.1 History of SEXTANT

Real-time route planning and re-planning is a robust and well-researched field. At the time of this writing, much of the literature is dedicated to route planning of autonomous vehicles either on city streets (e.g. for self-driving cars) or for unmanned flying vehicles (e.g. drones). In addition, much work has gone into route planning for virtual environments (e.g. video games). A review of relevant literature shows, however, that relatively little work has gone into the real time calculation of an optimal route or traversal path for human explorers in rugged terrain. As far back as 2003, researchers at MIT identified this knowledge gap – specifically in the context of future human planetary EVA – and started work on what would eventually become SEXTANT (Carr, Newman, & Hodges, 2003).

The software that is today known as SEXTANT began its life under a different name: *The Geologic Traverse Planner*. Created by Carr and Newman in 2003, this tool was designed to evaluate – exclusively – the acceptability of a planned traverse. The exclusivity distinction is made in order to emphasize that *The Geologic Traverse Planner* (oddly) performed none of the actual planning work; that work was handled entirely by a human user. The tool would take as input a user-generated traverse and – using information from digital elevation models (DEMs), landmark maps, and other data sources – calculate the metabolic cost and overall feasibility of that traverse. Though the tool was found to increase the apparent quality of a planned traverse (specifically when compared with the actual Cone Crater EVA traverse from Apollo 14), Carr and Newman noted that the user interface of the tool was cumbersome and would benefit greatly from a more user-oriented redesign.

With the partial aim of creating a more user friendly version of *The Geologic traverse Planner,* Marquez as part of her PhD work created the *Planetary Aid for Traversing Humans* (PATH) tool (Marquez, 2007). The PATH tool had two primary modes of traverse planning, which Marquez designated as 'passive automation' and 'active automation'. Pictorial representations of the outputs of the two modes is shown below in Figure 2.1.

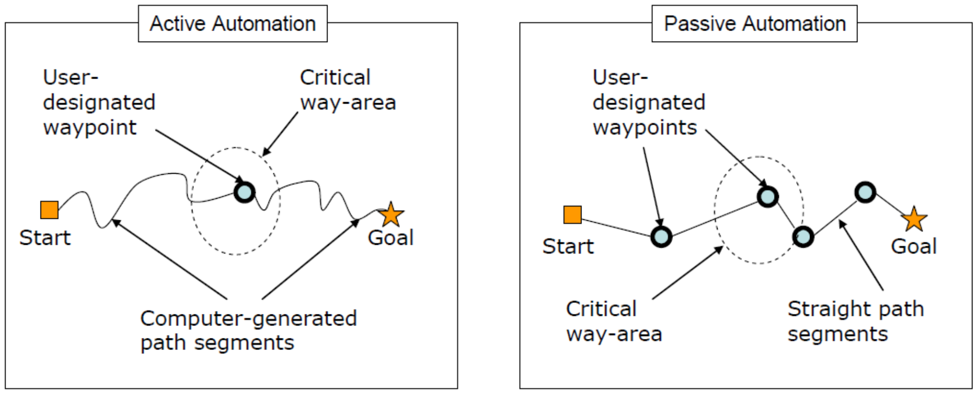
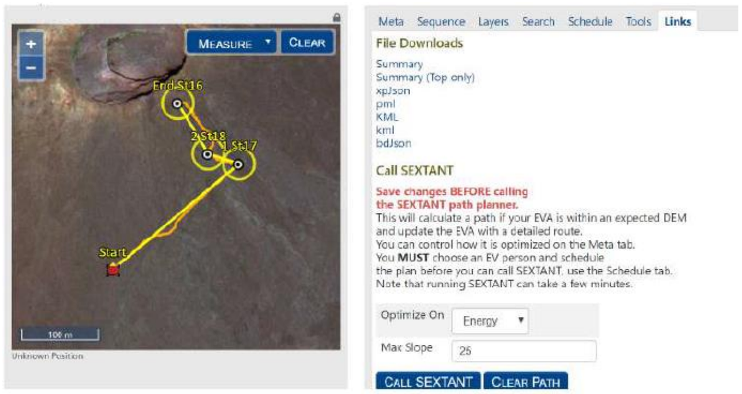


Figure 2.1: Difference between active (left) and passive (right) automation (Marquez, 2007)

In passive automation mode, PATH functioned very much like its predecessor; that is, the human user acted as the primary path planner. In order to elevate PATH's user experience beyond that of *The Geologic Traverse Planner*, Marquez integrated and allowed users to switch between various visualizations (e.g. a contour map, a metabolic cost heat map, etc.) that would help them pick the best path for a traverse. While this improved user experience was no doubt a quality contribution, PATH's active automation mode has since proven to have the larger influence on the future form of SEXTANT. In active automation mode, PATH would take a series of user-designated waypoints and – using a least-cost search algorithm similar to Dijkstra's – automatically join those waypoints in an optimal way. These algorithm-generated optimal paths were useful both as direct planning sources and also as sources of comparison against paths generated using some alternate method. One of the major findings of Marquez's work was that, while paths created using active automation mode tended to be generated more quickly and be lower cost than those created using passive automation mode, active automation had a negative effect on situational awareness. Because of this, Marquez suggested that future generations of path-planning software make sure to keep humans involved in the planning loop.

The next major improvement to what is now known as SEXTANT came from Johnson's 2010 master's thesis (Johnson et al., 2010). Johnson's work offered several important contributions, not the least of which was the first actual use of the acronym SEXTANT. In addition to giving the software its name, Johnson also spent a good deal of effort on extending the system to better handle path planning for non-human explorers, particularly rovers. Rovers consume energy and other resources much differently than their on-foot, human counterparts. Additionally, the constraints imposed on a traverse route by rovers and by humans differ greatly (i.e. managing ‘sun time’ is very important for solar panel equipped rovers). Johnson also improved SEXTANT in a broad, algorithmic manner by abandoning Dijkstra’s search algorithm in favor of the often faster A* (a-star) search algorithm. Both algorithms compute optimal paths (according to a supplied cost function), but A* is biased to investigate paths leading towards a goal point before those leading away (and thus tends to find a path more quickly). This improvement was relatively minor in the whole scheme of Johnson’s work, but has shown to be quite important in the subsequent evolutions of SEXTANT.

The last major modifications to the SEXTANT software (for the purposes of this work) came from Norheim’s 2018 work *Path Planning for Human Planetary Surface Exploration in Rough Terrain*. In its first iteration (as *The Geologic Traverse Planner*), SEXTANT was implemented in the proprietary language of the license-requiring software Matlab. In her work on the PATH tool, Marquez took the ideas of the earlier path-planning software and incorporated them into an all new piece of software written in Java. With SEXTANT, Johnson moved the path-planning software back into the Matlab environment. Hoping to build SEXTANT into more of an open-source platform, freely usable by any interested researcher, Norheim elected to once more port the software to a new language: Python (Norheim et al., 2018). To further support his goal of turning SEXTANT into a more pervasive and portable path-planning platform, Norheim spent significant effort designing and implementing a robust application programming interface (API) for SEXTANT. He tested this API with three different pieces of software: the Exploration Ground Systems (xGDS) tool (developed by NASA Ames), the Sextant WebApp, and the Holo-SEXTANT application. The results of the xGDS-SEXTANT integration can be seen below in Figure 2.2.



File Downloads

Summary
Summary (Top only)
xpJson
pml
KML
kml
bdJson

Call SEXTANT

Save changes BEFORE calling the SEXTANT path planner.
This will calculate a path if your EVA is within an expected DEM and update the EVA with a detailed route.
You can control how it is optimized on the Meta tab.
You **MUST** choose an EV person and schedule the plan before you can call SEXTANT, use the Schedule tab.
Note that running SEXTANT can take a few minutes.

Optimize On: Energy
Max Slope: 25

Figure 2.2: xGDS interface, with SEXTANT-created path in orange (Norheim et al., 2018)

SEXTANT's integration with Holo-SEXTANT proved to generate considerable interest for future work. Holo-SEXTANT is an application that runs on the HoloLens augmented reality headset (HoloLens and the concept of augmented reality are discussed in further detail in section 2.2) and displays a SEXTANT-generated path as a virtual overlay over a user's physical surroundings. Though the display itself was fairly simple, feedback from its use in an in-field study was generally positive. Subjects in particular expressed excitement at the potential utility of a hands-free path-following display that did not require any mental frame-of-reference transformations. An image of Holo-SEXTANT being used during the BASALT-2 mission is shown below in Figure 2.3.

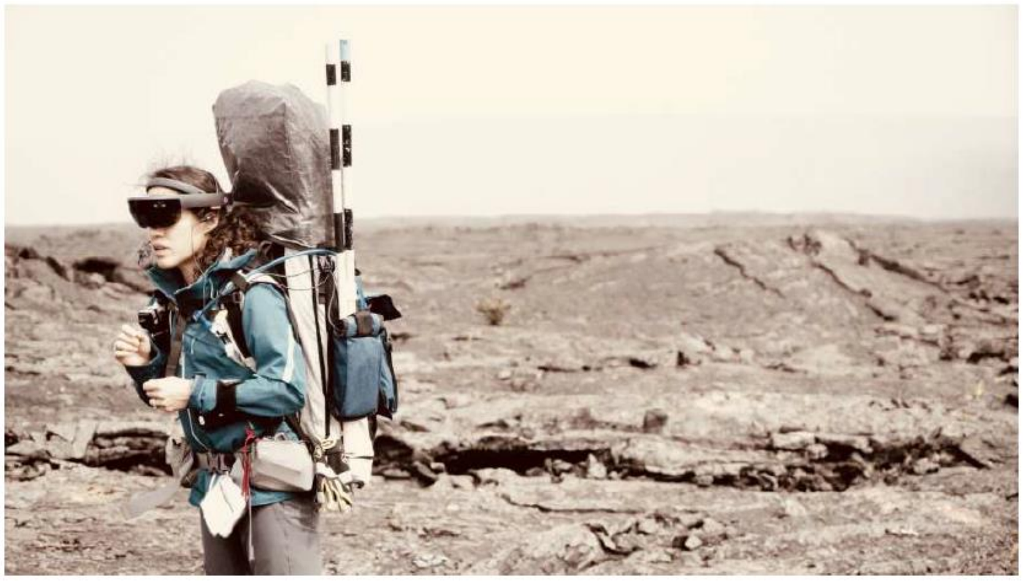


Figure 2.3: Subject navigating with Holo-SEXTANT (application on headset) during the BASALT-2 field study (Norheim et al., 2018)

2.2 Augmented Reality

2.2.1 Definition

According to Miligram and Kishino (1994), augmented reality (AR) and its technological sibling virtual reality (VR) represent two points along the single axis of the Virtuality Continuum. The Virtuality Continuum is shown below in Figure 2.4.

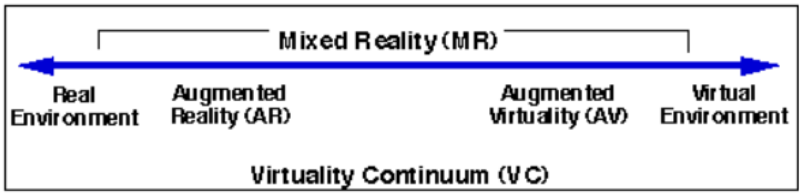


Figure 2.4: Simplified representation of the "Virtuality Continuum" (Milgram & Kishino, 1994)

The Virtuality Continuum is a one-dimensional, graphical visualization of the ratio of virtual vs real material present in a person's environment. Rightward (i.e. positive) movement along the continuum translates to an increase in the percentage of virtual content. Milgram and Kishino note that the leftmost 'real environment' end of the continuum represents a setting in which the user is surrounded by objects that are strictly 'real'; that is, objects that have "actual, objective existence" (Milgram & Kishino, 1994). A person who is walking about and experiencing the world without technological aids is interacting with a reality on this end of the continuum. The vast majority of the human experience has taken place in such 'real environments'.

At the other end of the Virtuality Continuum lies the 'virtual environment', which is today more commonly known as virtual reality. Here, a user's world is populated exclusively by "objects that exist in essence or effect, but not formally or actually" (Milgram & Kishino, 1994). In both virtual reality and its continuum-adjacent neighbor augmented virtuality, the visible portion (and often the audio portion as well) of a user's environment is entirely virtual. In the case of augmented virtuality, some gateway to the user's actual surroundings – such as a graphical window or 'cutout' that reveals a view of the real-world – may also be present.

Moving in from the extreme left end of the continuum, we find the region representing augmented reality. In contrast to both VR and augmented virtuality, AR places a user in an environment that is based solidly in a user's real-world surroundings. This reality-based environment is then 'augmented' by virtual objects such as waypoints or location-based description boxes. Krevelen (2010) refers to these virtual augmentations as "local virtuality".

Azuma (2001) presents thematically similar – though more structured – definition of augmented reality, stating that an AR system is one that:

- combines real and virtual objects in a real environment;
- registers (aligns) real and virtual objects with each other; and

- runs interactively, in three dimensions, and in real time.

While both augmented and virtual reality are often associated with specific pieces of hardware – such as the head-mounted display (a device discussed more extensively in the following sections) – it should be noted that the definitions given here come with no specific hardware requirements.

2.2.2 Historical Implementations

Though AR-like concepts appeared in earlier works of fiction, Ivan Sutherland's 'Sword of Damocles' – created in 1968 – is generally considered the first real-world implementation of the technology (Sutherland, 1968). The Sword of Damocles consisted of a head-mounted display (HMD) – a piece of hardware now almost synonymous with the technology itself – that was suspended from the ceiling. This rather precarious-looking setup – resembling the hanging sword from the eponymous ancient Greek parable – earned the device its curious nickname. A photograph of the 'Sword of Damocles' is shown below in Figure 2.5:

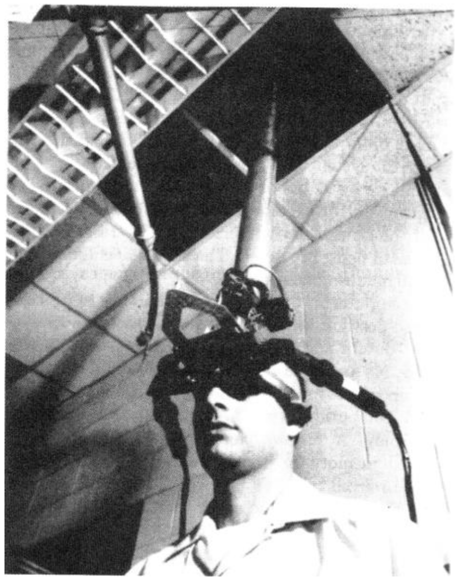


Figure 2.5: Sword of Damocles (Sutherland, 1968)

Sutherland's HMD used two miniature cathode ray tubes (CRTs) – one covering each eye – to display simple virtual objects to the user. As the wearer rotated and translated their head, the virtual objects would move and reposition themselves in such a way that they appeared to exist in three-dimensional space alongside the user. Though not entirely obvious from the image shown in Figure 2.5, the two displays within the HMD were semi-transparent (Sutherland, 1968). Because of this, the user was able to perceive both virtual and real objects – placing the headset somewhere near the 'augmented reality' region of the Virtuality Continuum.

While Sutherland developed his 'Sword of Damocles' largely to determine if such a technology *could* be created, efforts to utilize AR as a tool to actually enhance task performance began in the early 1990s. One such early effort was Louis Rosenberg's AR system 'Virtual Fixtures' (Rosenberg, 1992). In his experiment, Rosenberg tasked a group of subjects with quickly moving a wooden peg between a series of holes in a pegboard. Rosenberg chose to measure task performance through task completion time, equating a low task completion time to high performance. In his conclusion, Rosenberg found that providing subjects with virtual, task-relevant overlays (which Rosenberg likened to "a ruler guiding a pencil") increased performance (Rosenberg, 1992).

While the AR systems developed by both Sutherland and Rosenberg presented their virtual scenes via an HMD, many other developers have created augmented reality devices that display virtual content using different methods. In his work "Augmented Reality Navigation Systems", Narzt notes that the windshield of a car has, for example, been in several instances successfully conscripted for use as an AR display surface (Narzt et al., 2005). Such a car-based augmented reality system is shown below in Figure 2.6.

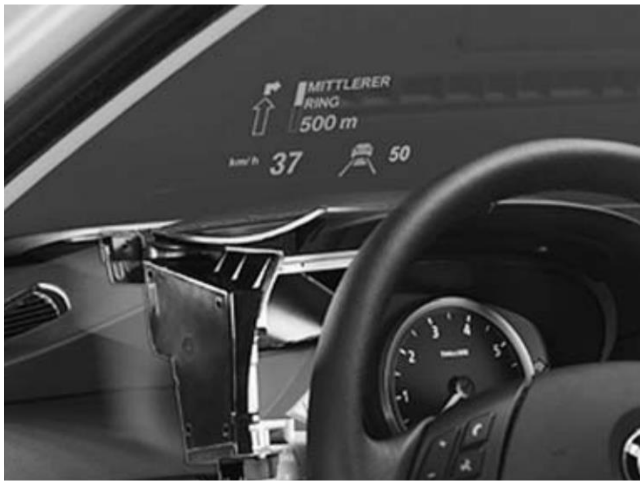


Figure 2.6: 'Optical See-Through' AR display on the windshield of a car (Narzt et al., 2005)

Because the virtual elements (e.g. current speed) of the display are drawn on a naturally transparent surface (here, the car windshield), this type of augmented reality is known as ‘optical see-through’ (OST) (Krevelen & Poelman, 2010). Believing AR could be used to enhance an automobile in a more meaningful way than a simple speedometer display, Narzt elected to develop a car-based AR navigation device of his own. This device is shown in Figure 2.7.

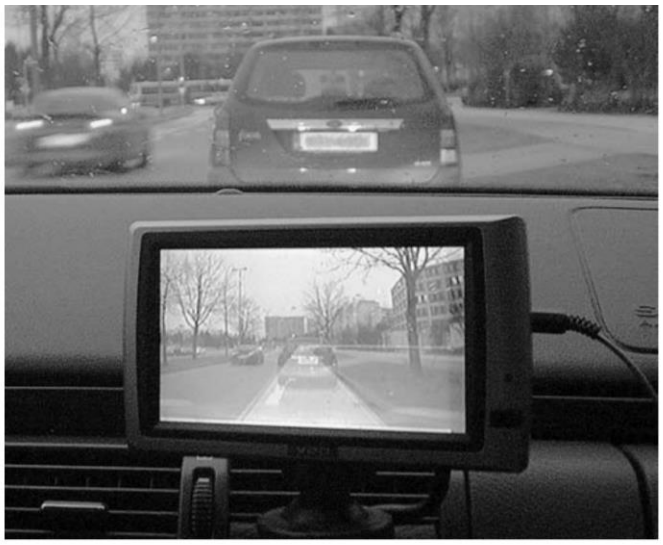


Figure 2.7: 'Video See-Through’ AR display used for navigation (Narzt et al., 2005)

Narzt’s display uses a single digital screen to render virtual objects on top of a video feed of the surrounding environment. This type of display is categorized into a family of AR systems known as ‘video see-through’ (VST). The VST and OST concepts are well-summarized by the following pair of figures:

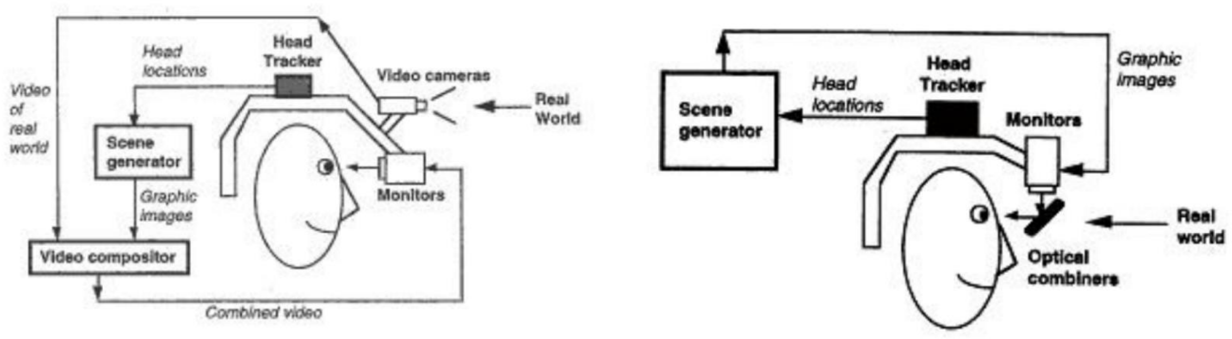


Figure 2.8: Pictorial representations of VST (Left) and OST (Right) AR (Wolfenstetter, 2007)

Because video see-through augmented reality systems are the “cheapest and easiest to implement” (Krevelen & Poelman, 2010), devices similar to the one shown in Figure 2.7 are the most prevalent implementation of AR. Indeed, VST has for the majority of AR’s history been the only means by which an average person could experience the technology. In 2014, however, Google aimed to change this historical trend by releasing their very own OST HMD: Google Glass (GG). Designed as a pair of ‘smart glasses’ on which basic virtual information could be displayed (e.g. user reviews of a nearby business), Google Glass became the first widely available optical see-through AR device. Though ultimately a commercial failure, Google Glass’s impact on the AR community was significant; the device was used by many different researchers in a host of notable AR studies. In the years since Google Glass’s release, head-mounted OST augmented reality devices have continued pushing their way into the mainstream. The highest profile OST augmented reality device available today is the Microsoft HoloLens (Figure 2.9).

Figure 2.9: Microsoft Hololens worn by a user (from Microsoft.com)

2.2.3 Current Uses

Without a doubt, augmented reality is a powerful technology. As its name suggests, it applies a filter to the environment surrounding a user, distorting and augmenting it in ways that – ideally – make a given task more enjoyable or easier to complete. This power is not lost on the members of the many human-centered industries of today’s world, and because of this AR has been incorporated (with varying degrees of success) into many different fields.

2.2.3.1 Navigation

In a 2005 study, Narzt – as mentioned above – created a VST AR navigation device for use in automobiles. By overlaying an app-suggested route on a digital display of the driver's real-world environment, Narzt's device removed the need for the driver to interpret and apply a map-based abstraction to their surroundings; rather, the driver was able to simply 'follow the yellow line' (Narzt et al., 2005). Another example – one which is foundational to this paper – where augmented reality was used as a navigation aid is Holo-SEXTANT. Trekking over the rocky terrain of Hawaii's Volcanoes National Park, members of NASA's Biologic Analog Science Associated with Lava Terrains (BASALT) team employed the Holo-SEXTANT AR system as a virtual guide – one which helped them maintain their bearings in the barren environment (Anandapadmanaban et al., 2018). Running on the Microsoft Hololens, Holo-SEXTANT presented researchers with a virtual overlay of their pre-determined traversal route. Though no formal user performance data was collected during the study, anecdotal feedback given by users suggested a general optimism about future exploration of Holo-SEXTANT-related technology. A screenshot of the Holo-SEXTANT navigation interface is shown below in Figure 2.10.

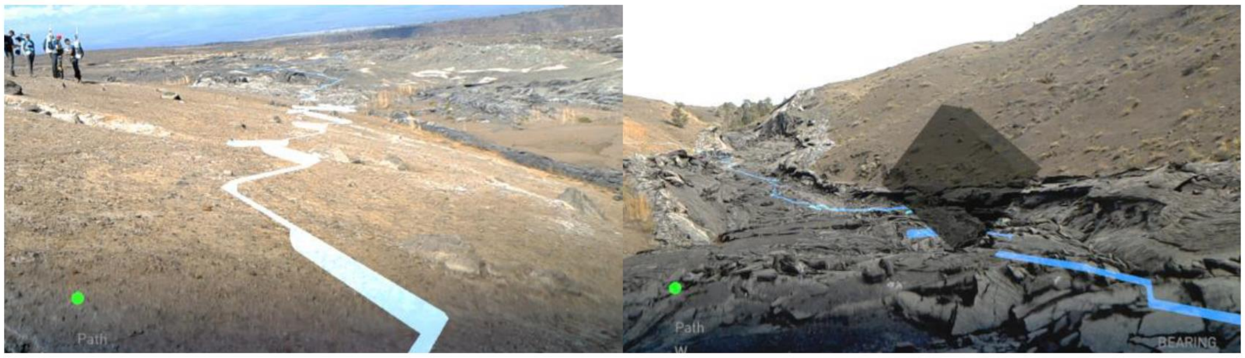


Figure 2.10: Screenshots from Holo-SEXTANT (Anandapadmanaban et al., 2018)

2.2.3.2 Maintenance & Assembly

Another field for which augmented reality is particularly well-suited is maintenance and assembly; this is in large part due to AR's ability to locate information displays in the same three-dimensional space as the objects to which they relate. A study performed by Henderson and Feiner (2011) illustrates this point well. In their study, Henderson and Feiner asked subjects to don an AR HMD and perform several maintenance tasks on an armored personnel carrier (APC) turret (Henderson & Feiner, 2011). The HMD interface provided subjects with a set of virtual instructions that contained step-by-step text, close-up views, mechanism identifiers, and virtual versions of the tools needed for a particular step. The HMD hardware and user interface can be seen in Figure 2.11:

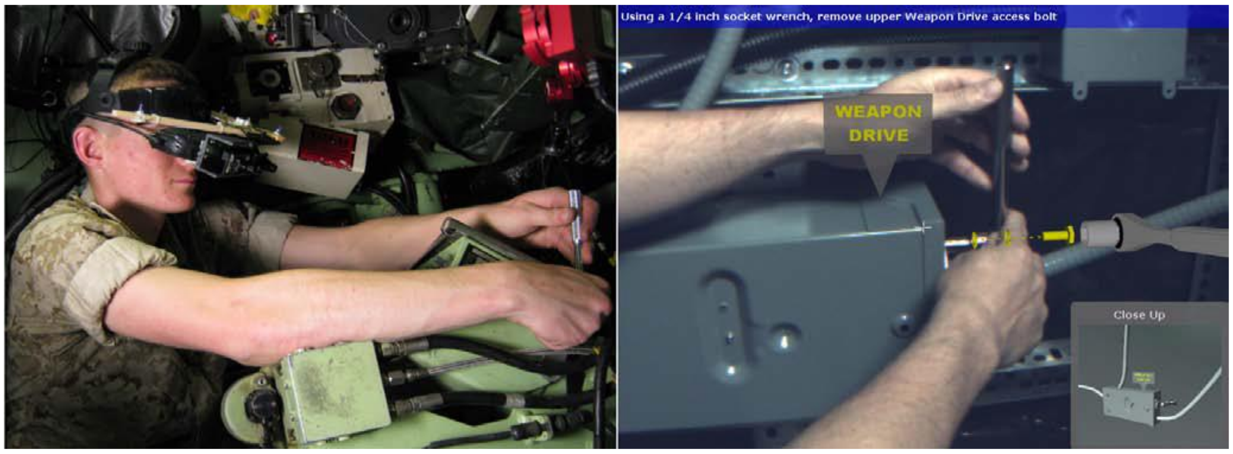

Using a 1/4 inch socket wrench, remove upper Weapon Drive access bolt
WEAPON DRIVE
Close Up

Figure 2.11: APC turret maintenance AR HMD hardware (left) and user interface (right) (Henderson & Feiner, 2011)

Though subjects generally took more time to complete a maintenance task when using the AR HMD than when using the traditional display, Henderson and Feiner did note that AR out-performed the LCD display in a criteria they called “localization time” (i.e. the time taken to visually locate the to-be-maintained mechanism).

In another assembly-related study, Syberfelt (2015) provided subjects with an AR HMD (in this case, an Oculus Rift repurposed as a VST AR headset) and tasked them with constructing basic wooden structures. Structures were assembled from simple, three-dimensional (Tetris-like) blocks in a way that Syberfeldt hoped would well-emulate work on an assembly line. In a slight deviation from the AR study norm, Syberfeldt’s goal was not to evaluate the performance (e.g. completion time) of AR against some more traditional display medium; rather, Syberfeldt wanted to evaluate AR’s usability and enjoyment-of-use in an assembly line setting. The largest barrier to the adoption of AR technology, Syberfeldt argues, will be acceptance of the product by actual workers. The study found that the AR interface was reasonably usable and understandable – even by first-time users – though traditional paper-based instructions received an equal or better rating in all collected qualitative measures.

2.2.3.3 *Surgery*

In their work "Emerging Technology in Surgical Education: Combining Real-Time Augmented Reality and Wearable Computing Devices" (2014), Ponce and Menendez recruited surgeons to help evaluate the efficacy of using AR in a surgery setting. One of these surgeons (surgeon A) remained on-site and actually performed the surgery, while the other (surgeon B) oversaw the surgery and provided guidance from a remote location. Surgeon A wore a Google Glass (GG) headset, the camera of which was used to stream a 'surgeon's eye view' of the operation to surgeon B. While viewing this surgical live-stream, surgeon B positioned and manipulated his hands as if he were performing the operation himself. A second camera captured surgeon B's hand motions and relayed them back to surgeon A, allowing him to "see a hybrid image combining the image of the surgical field with the image of his colleague's hands or tools through the GG head-mounted display" (Ponce et al., 2014). This setup is shown below in Figure 2.12.

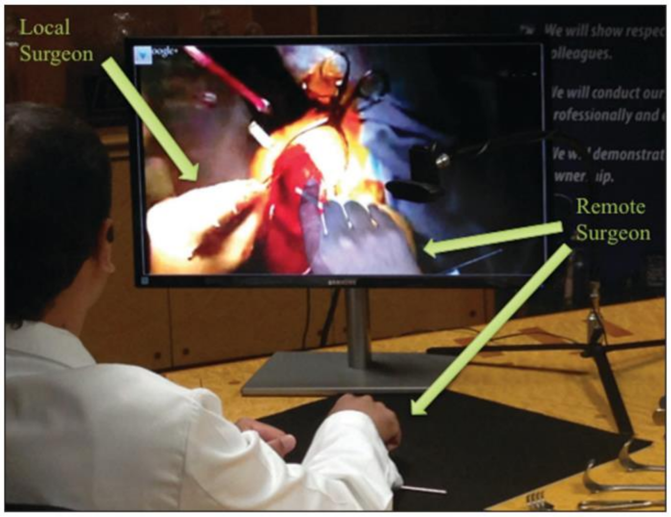


Figure 2.12: Ponce and Menendez experimental setup

Though the surgery took longer than it likely otherwise would have, surgeon B noted that communicating through augmented reality felt natural and that it allowed "more precise instruction than through verbal interaction" (Ponce et al., 2014). In a separate study, Dickey and Hakky (2016) conscripted several urology students and faculty to help them evaluate the effectiveness of using AR to assist with the placement of an inflatable penile prosthesis (IPP). Using a GG headset, students performing the IPP placement were able to view video-based descriptions of each step of the operation. In addition, instructors were given the ability to issue guidance to an operating student by moving a cursor or drawing virtual images on the GG display. Though the study found no quantitative indication of performance increase (since no quantitative measures were actually taken), a post-procedure qualitative assessment did reveal that “both faculty and trainees view this new technology [AR] in a positive light” (Dickey et al., 2016).

2.2.3.4 Geology

When surveying a particular region, geologists will employ maps and other geological implements that contain information on “details of rock structures, pockets and seams of minerals, mine topography, and soil data” (Mathiesen, Myers, Atkinson, & Trevathan, 2012). As with any data set, these charts take on many different forms and can be displayed in many different ways. While the traditional paper map remains a useful aid, Mathiesen notes that geological data presented in a three-dimensional format (which allows for color-coding, panning, and zooming) offers significant advantages. Knowing this – and also recognizing the potential benefits of applying AR to this scenario – Mathiesen conducted a study where he asked subjects to view and analyze geological data using a VST AR device. Understanding that geological studies often involve physically demanding traverses, Mathiesen elected to implement his augmented reality display on an Android mobile device. The display itself used GPS (along with location data present in the 3D geological map) to determine the device-relative positions of various geological formations. Equipped with this information, the device displayed the geological maps as three-dimensional overlays on the video from the forward-facing camera. Depth information was relayed to the user through a transparency gradient; the closer a geological element was to a user, the opaquer it appeared. As is the case for many AR studies (often due to low sample size), Mathiesen collected no statistically significant quantitative performance data. However, qualitative feedback provided by subjects was “overwhelmingly positive”, many citing as the source of their positivity the device’s elimination of the need to perform mental map-to-world transformations.

2.2.3.5 Education

Augmented reality has also found productive use in the world of education. Educators have long understood that different learners have different needs when it comes to absorbing and retaining knowledge; some learners prefer text-based material, while others prefer to have material presented in a visual manner (Felder & Silverman, 1988). It has also been shown that students tend to learn material better (or are at least able to recall it more easily) when the learning takes place in a relevant context (Godden & Baddeley, 1975). Augmented reality has the unique ability to present visual information in the exact context to which it is most relevant, and because of this has found success in settings of learning such as classrooms and museums. In his work "Applications of Augmented Reality for Archaeological Purposes", Wolfenstetter discusses how ARCHEOGUIDE and LIFEPLUS – two augmented reality applications – were used to enhance a user's visit to a cultural heritage site (Wolfenstetter, 2007). Both of these projects used AR HMDs, though the former used a VST setup while the latter used an OST setup. As described by Wolfenstetter, ARCHEOGUIDE and LIFEPLUS present users with virtual re-creations of ancient structures and works of art, cleverly placing the re-creations in the exact locations that their historical counterparts once occupied. Wolfenstetter cites accessibility as one of the key advantages that ARCHEOGUIDE and LIFEPLUS have over their traditional alternatives. The applications allow users to observe and learn from the virtual architectural models simply by walking around and looking at them. Since humans from childhood use this 'learn by observation' technique, the two applications require little training and are quite intuitive (i.e. are accessible).

2.2.3.6 Gaming

Whereas an enthusiastic gaming community is arguably responsible for the recent resurgence of virtual reality as a consumer product (e.g. Oculus Rift, PSVR), the same cannot (as of yet) be said for augmented reality. This lack of mass adoption by gamers is likely due – at least in part – to the significant cost difference between current AR and VR headsets. As of this writing, first-party VR HMDs cost about $200; in contrast, the cost of an AR HMD is about $2000 – an order of magnitude greater. Even so, augmented reality games do exist, and are actually quite often an excellent showcase of the technology. One of the most popular augmented reality games is Microsoft's 'RoboRaid'. Playable only on the HoloLens, RoboRaid places a player in a scenario where their home (or whatever building they currently occupy) becomes the target of a robot assault. Flying metallic adversaries burst through walls and launch projectiles at the player, forcing them to duck and dodge to avoid in-game injury. The HoloLens, which constantly generates and re-generates a three-dimensional map of a player's surroundings, creates this 'wall bursting' effect by drawing a 'hole' texture on top of one of its 3D-mapped surfaces. Simultaneously, the 'ducking and dodging' actions performed by the player are interpreted by the headset's built-in pose tracking system; this system accomplishes its position and orientation tracking through a combination of image processing and inertial measurement unit (IMU) analysis (Aaron, Zeller, & Wojciakowski, 2017).

2.2.4 Modes of Interaction

Many of the digital devices that we interact with today display information – much of which is inherently two-dimensional (spreadsheets, documents, etc.) – on a flat LCD screen. Input tools historically associated with these devices (such as the cursor-controlling mouse and touchpad and the text-generating keyboard) lend themselves particularly well to these display formats. Even so, the most pervasive piece of modern digital hardware – the ever-present smartphone – has abandoned these traditional modes of input. This abandonment is not a byproduct of some substantive difference in the content viewed on the two device types. Rather, the change is a result of the *way* in which mobile users consume information. No longer constrained to seated positions in an office or coffee shop, smartphone users interact with content 'on the go' in a wide array of dynamic situations. To accommodate this, smartphones chose to discard the bulky and cumbersome external input devices favored by their desk-based relatives, opting instead to receive input by way of an integrated touch-screen. To reach its current usable and seemingly stable configuration, the smartphone underwent many design changes. Though officially an older technology, augmented reality devices have not yet experienced the same level of refinement as the modern smartphone; even so, they too have seen a good number of input mode iterations. A discussion of the variety of input methods used for augmented reality, both successful and unsuccessful, follows in the sections below.

2.2.4.1 Pose Tracking

The most important, essentially medium-defining input that a user provides to an AR system is their own position and orientation. A person or object's position and orientation – when taken together – are often referred to as that person or object's "pose"; future mentions of pose in this document are intended to be interpreted in this manner. Pose tracking is so important because it is required to realize the first point (i.e. "combines real and virtual objects in a real environment") of Azuma's three-point definition of AR. Sutherland's Sword of Damocles implemented pose tracking via two different methods: forward kinematics of the arm connecting the headset to the ceiling (i.e. the “dangling sword” portion of the device) and a 12-point, SONAR-like ultrasonic locating system. Though neither implementation is still used by current AR devices, pose tracking continues to be an important feature of modern AR systems. AR headsets such as the Microsoft HoloLens and the Magic Leap use an onboard inertial measurement unit (IMU) and a variety of depth and “environment understanding” cameras to implement a form of pose tracking called SLAM (simultaneous localization and mapping) (Decker, 2019) . SLAM is a particularly useful form of pose tracking for AR because – as its name implies – it maps the user’s nearby environment at the same time as it tracks their pose. This mapping can be transformed into a three-dimensional mesh, which in turn can be used to facilitate interactions between real and virtual objects (knowing the basic geometry of one’s surroundings is *incredibly* useful for AR applications). Some studies – such as MIT’s own Holo-SEXTANT project (Anandapadmanaban et al., 2018) – have also used the global positioning system (GPS) to track a user’s position. This type of position tracking (orientation tracking *can* be done with GPS, though special handling is required) works well for situations where highly accurate position data is not required. Example use cases are navigating a kilometers-long path (e.g. Holo-SEXTANT) or displaying information about nearby businesses or addresses in a hypothetical urban-based navigation application.

2.2.4.2 Mouse & Keyboard

The mouse and keyboard have become user input staples of the desktop computing world (though this is changing somewhat with the rise of tablets and high-quality touch interfaces). The mouse works so well in the desktop environment because its two degree-of-freedom (left-right, forward-back movement on a tabletop) and 0th-order nature (position changes of the mouse correspond to position changes of the cursor) maps perfectly to the two-dimensional workspace of a computer monitor. Similarly, the keyboard thrives in the desktop environment because its many tactile keys are as-of-yet unrivaled in their ability to accommodate text input. Hoping certain traditional design techniques would “make the jump” to AR, early AR applications tended to mimic their desktop-based ancestors by implementing the WIMP (windows, icons, menus, and pointing) design pattern. Developers quickly discovered, however, that WIMP interfaces performed poorly in AR applications – in large part due to their inability to handle interactions with a third spatial dimension (Krevelen & Poelman, 2010). Even so, the types of interactions afforded by the mouse and keyboard (object selection and text input, respectively) are still very much useful in AR applications. A common method of attaining mouse-like, point-and-click functionality in AR is to place a cursor at the center of the viewport, which in effect aligns the cursor to the AR device's forward vector. The actual "click" of the cursor can be accomplished through gesture recognition (as is done in the HoloLens), by a button press on a handheld controller, or through some other means. A common – though somewhat cumbersome – technique for handling text input is via a cursor-click-to-type virtual keyboard. In a more inventive effort to solve the keyboard conundrum, Antoniac and Pulli (2001) developed the Marisil Mobile User Interface Framework. Marasil (Figure 2.13) uses gesture recognition to project a contextually relevant set of virtual keys on to a user's palm, the keys of which can by "typed" by touching the appropriate location on the palm.

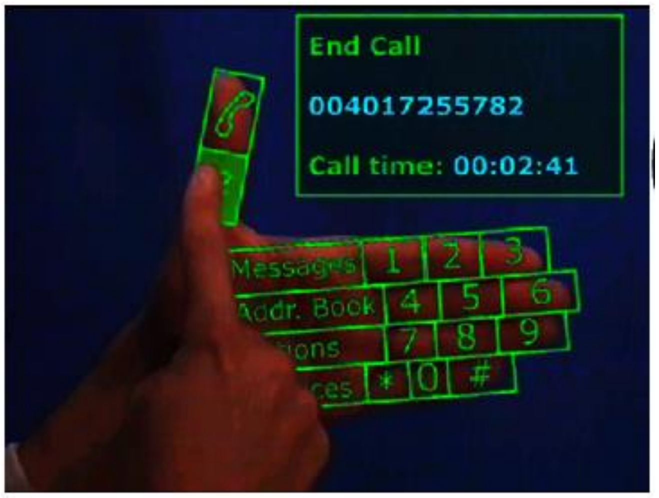


Figure 2.13: Marisil Mobile User Interface Framework (Krevelen & Poelman, 2010)

Other techniques for handling text input in AR involve haptic gloves, tangible user interfaces, and the precise pose tracking of a user's fingertips (Krevelen & Poelman, 2010). As of yet, no solution has proven to be the AR-equivalent of the now-classic "mouse and keyboard" combination; the area very much remains open for further research.

2.2.4.3 Touchscreen

Though arguably the most immersive form of augmented reality, HMD-based AR systems tend to suffer when they attempt to allow complex interactions with virtual objects. Many of these interaction difficulties arise because the most obvious or intuitive mode of interacting with an

HMD-projected virtual object – reaching out and physically touching it – is very difficult to simulate convincingly. NASA managed to do so at the VR lab at Johnson Space Center using a “tendon-driven robotic device”, though their system is non-mobile and can only be used within the room in which it was built (Dunbar, 2017). Video-see-through AR devices circumvent the need to simulate object weight and touch by presenting virtual objects (and the rest of the AR environment) in a more “traditional” way: via a digital screen. Indeed, if the screen of the VST AR device also happens to be a touchscreen, a user’s familiarity with smartphones and tablets lends an intuition altogether unavailable to HMD-based systems. Common touch interface features such as “touch to select”, virtual buttons, and a virtual keyboard can – if implemented well – be used with relatively little training. Even so, the ever-difficult task of creating an intuitive system for affecting the pose of virtual objects remains unsolved for AR touch interfaces (Marzo, Bossavit, & Hachet, 2014). Three-dimensional manipulation is difficult in all settings, however (3D modeling software has been around for years, and still no agreed-upon “best” system exists), so this fact speaks less to the deficiencies of the touch interface and more to the complexity of the task itself.

2.2.4.4 Speech Recognition

One of the key benefits of AR is that it allows users to simultaneously interact with both real and virtual objects. The most intuitive (and even instinctual) way to interact with a real object is through direct manipulation with the hands – and ideally a set of hands not preoccupied with holding something else. Because of this, many AR systems strive to create a “hands free” (or at least “hands empty”) form of input handling. One of the more obvious ways of achieving this is through automatic speech recognition (ASR). ASR is a good tool for performing actions such as dictation, showing or hiding user interface elements, or proceeding to the next instruction of a multi-step task. Actions that are hard to describe in few words (such as object rotation, translation, or scaling) are much more difficult to implement with ASR. One of the major complaints with AR systems (and really any system) that use ASR is recognition accuracy, a metric often measured using word error rate (WER). As of 2017, Google’s cloud-based speech recognition technology (one of the best in the industry) achieved a WER of 4.9% (Protalinksi, 2017). Though good, a 4.9% WER still means that every 20th word is misinterpreted in some way; this error rate only increases when using ASR in on-device mode (i.e. without access to the cloud) or in a noisy environment. In situations where mistakes must be kept to an absolute minimum (e.g. a hospital surgical setting), ASR may prove an unsatisfactory form of input handling.

2.2.4.5 Gesture Recognition

Gesture recognition – much like speech recognition – has become a popular form of AR input handling, in large part due to its non-reliance on additional input peripherals. Gesture-based systems have the added benefit of often being very intuitive; real world, hand-based object manipulations (e.g. grasping, pinching, dropping) can often be translated to recognizable input gestures with relative ease. In his work "FingARTips", Buchmann (2004) tasked subjects with laying out an urban environment using press, point, and grab gestures. Most subjects reported that using these gestures to move and manipulate urban structures was "easy and intuitive". Subjects who had a less pleasant experience attributed their difficulties to poor hand and finger tracking.

2.2.4.6 Hand Controllers

Another common tool for supplying input to an AR system – particularly in video gaming system – is through hand controllers. Microsoft's HoloLens is shipped with a very simple hand controller, a one button 'clicker' that provides what is essentially the left-click functionality of a desktop mouse (Decker, 2019). Though simple, some users prefer the tactile nature of using the clicker over hand gestures. A more robust example of an augmented reality hand controller is the one that comes packaged with the Magic Leap AR headset, shown below.

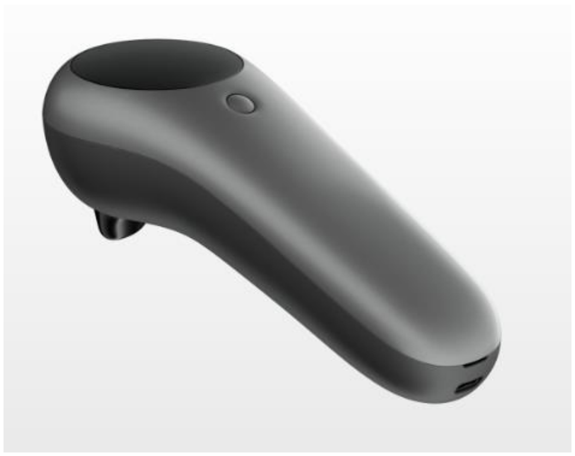


Figure 2.14: Magic Leap Hand Controller (magicleap.com)

Magic Leap's hand controller has a single analog trigger, two digital buttons, and a touch-pad and supports six degree-of-freedom motion tracking (MagicLeap, 2018).

2.2.5 Advantages & Drawbacks of Augmented Reality

In the previous sections discussing the history, uses, and input types of AR, certain recurring themes have emerged. Some of those themes reveal generally positive aspects of the technology, while others reveal generally negative ones.

2.2.5.1 Advantages

- Reduces task time when dealing with complex structures
    - Hou et al. developed a prototype AR system for Lego assembly and determined that the system yielded shorter task completion times, fewer assembly errors, and lower total task time when compared to paper-based manuals (Hou, Wang, Bernold, Asce, & Love, 2013).
- Reduces time to locate/identify particular objects
    - In their APC maintenance study, Henderson and Feiner found that localization time (time to 'look at' or 'visually acquire' the appropriate object for the current task step) was decreased when using AR (S. J. Henderson & Feiner, 2009).
- Increases ability to understand abstract or complex spatial relationships
    - In Mathiesen's study, where AR was used to visualize sub-surface geological features, researchers noted how AR increased comprehension of the geological structures (Mathiesen et al., 2012).
    - The coexistence of virtual objects and real environments allows learners to visualize complex spatial relationships and abstract concepts.
    - In their study, Wu et al. note that generally 'unobservable phenomenon' – such as the rotation of the Earth – can be more easily learned if students can visualize and interact with them; AR provides such a means (Wu, Lee, Chang, & Liang, 2013).
- Wearable/mobile
    - When implemented as an HMD, AR can be a "hands free" technology; this is useful for performing many manual tasks.

    - In the Holo-SEXTANT study, subjects were required to traverse difficult terrain while wearing large, heavy backpacks. Having unoccupied hands is helpful in such physically strenuous scenarios (Anandapadmanaban et al., 2018).

2.2.5.2 Drawbacks

- Ease-of-use issues
    - When compared with traditional assembly techniques, Syberfelt found that subjects found traditional paper-based instructions easier to understand. (Syberfeldt et al., 2015). While this is reasonable given that most users possess vast experience dealing with paper instructions and relatively little dealing with AR, it remains a drawback that must be accounted for.
- Potential for information overload
    - In developing an AR display for locating a room in an office building, Julier notes that displaying all of the information relevant to a task can overburden a user (Julier, Baillot, Brown, & Lanzagorta, 2002). A good mitigation technique is applying an information filter, displaying only the items that are most relevant to a user's current task.
- Indoor/outdoor use, lighting concerns
    - Anandapadmanaban needed to modify his HoloLens in order for the display to be visible while being operated in an outdoor environment (Anandapadmanaban et al., 2018).
- Inadequate interaction modes
    - Gestures and speech recognition remain too unreliable to be used in all situations, and using hand controllers negates AR's "hands free" advantage. More work still needs to be done to find a robust and usable input system for AR.

2.3 Extravehicular Activity (EVA)

2.3.1 Definition

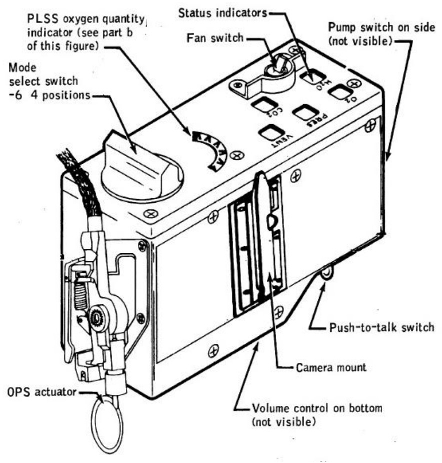
Extravehicular activity is – according to NASA – "an icon of human space exploration" (Roberts, 2015). Indeed, from those fateful first steps on the lunar surface to the construction of the International Space Station, the accomplishments of astronauts on EVA account for some of the most memorable in all of human history. Even so, the ‘formal’ definition of EVA is quite simple (and is essentially a deconstruction of the term itself). The prefix ‘extra’ means ‘outside of’ or ‘beyond’, while the word ‘vehicular’ refers to the spacecraft or habitat in which an unsuited astronaut generally resides. Merge the two, and you arrive at a suitable definition: any task (or, rather circularly, *activity*) that is conducted outside of a spacecraft. The types of tasks performed during EVA are many and varied, ranging from repair to exploration to simply determining if a human can survive in such a hostile environment.

2.3.2 EVA Roles

The work and effort of many individuals is required in order to execute a successful EVA. The role of each involved individual falls into one of three primary groups: extravehicular (EV) crew, intravehicular (IV) crew, and Mission Control Center (MCC) personnel (sometimes referred to as Mission Support Crew – MSC – depending on the mission). EV crewmembers are those individuals who actually don a space suit and depart the safety of their habitat, entering a hostile environment as the physical instrument that accomplishes mission objectives. Historically – with the exception of the Gemini missions – EV crews have consisted of two people. The IV crew are those individuals who remain within the shirt-sleeve environment just departed by the EV crew. IV crew are always close enough to their EV counterparts to communicate in real time and – according to the Pavilion Lake Research Project (PLRP) study – should consist of two crewmembers: one to focus on operations and one to focus on science. MCC personnel are generally all located in an Earth-based facility and consist of operations, science, and communications experts (K. H. Beaton et al., 2017).

2.3.3 Communication during EVA

Historically, EVA has been an information poor environment, particularly for the EV crew. Other than their own human senses, suited EV crewmembers have only two ways of receiving information about their surroundings and current status: a chest-mounted diagnostics display and a radio communications system. The diagnostics display used during the Apollo missions was worn on the chest and was part of a piece of hardware called the Remote Control Unit (RCU). A diagram of the RCU is shown in Figure 2.15.


PLSS oxygen quantity indicator (see part b of this figure)
Status indicators
Fan switch
Pump switch on side (not visible)
Mode select switch -6 4 positions
Push-to-talk switch
Camera mount
OPS actuator
Volume control on bottom (not visible)

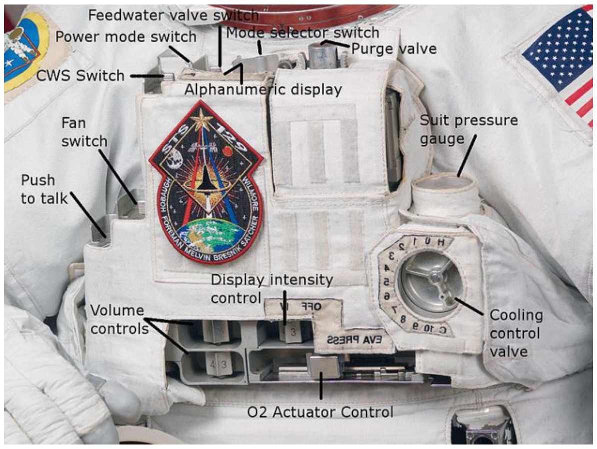


Figure 2.15: (Left) Diagram of the Apollo Remote Control Unit (Jones, 1995)

Figure 2.16: (Right) EMU Display and Control Module (NASA)

The RCU had a gauge indicating how much oxygen remained in the Portable Life Support System (PLSS) and several status and warning indicators for other suit functions. The descendant of the Apollo-era A7L space suit – known as the Extravehicular Mobility Unit (EMU) – was worn by NASA astronauts during shuttle and International Space Station (ISS) missions. The Display and Control Module (DCM) on the EMU, though upgraded with a digital display, served much the same function as the RCU before it. The vast majority of information flowing to EV crewmembers comes through the radio communication system. Select IV crew and individuals from mission control can access this radio system to directly exchange vocal requests and instructions with EV crewmembers.

2.3.4 Planetary EVA

2.3.4.1 Geological Objectives/Operations

Scientific objectives of planetary EVAs are typically geological in nature. However, relatively few human-based, extra-terrestrial missions with geological objectives have been performed; the number of such missions performed on Earth is far greater. Because of this, looking at Earth-based studies can offer valuable insight into how operations relating to geological objectives might be carried out during future planetary EVAs. Hodges describes field geology as a “discovery-based science”, noting that studies commonly begin with an ill-posed or overly general problem statement. Hoping to either refine or find outright answers to their research questions, field geologists begin a study by creating an initial traverse plan. Having not yet set foot in the region of study, terrestrial geologists create these initial traverse plans using data from remote sensing equipment. Upon arrival at a study site, geologists will commonly ascend a nearby hill to make an in-person re-assessment of the planned traverse. As the geologist moves from station to station, they will take notes and collect data, altering and updating the traverse plan as needed. Additionally, the geologist will during their traverse be constantly on the look-out for targets of opportunity (TOPs) – stations of geological interest that had not been obvious in the remote sensing data (Hodges & Schmitt, 2011).

2.3.4.2 Apollo

Of all the EVAs that have been conducted – from Alexei Leonov's initial 1965 spacewalk until the present day – only a small minority have taken place on the surface of planetary bodies. Indeed, the Apollo missions from the late 1960s and early 1970s remain the only events during which humans have set foot on extra-terrestrial soil. The large disparity between the number of planetary vs. non-planetary EVAs can be seen in Figure 2.17.

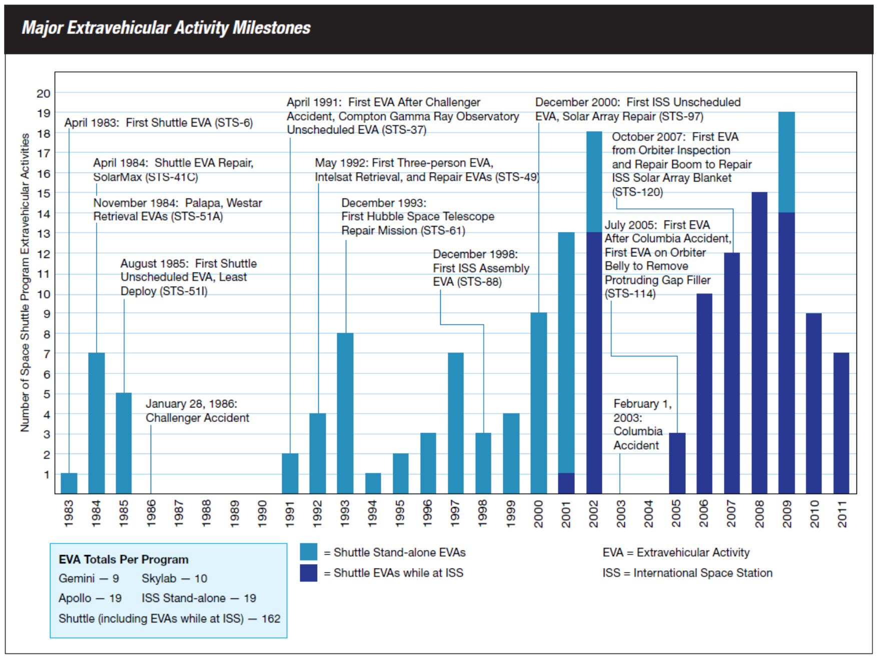


Figure 2.17: Major EVA Milestones and Counts, by Year (Patrick, Kosmo, Locke, Trevino, & Trevino, 2010)

Even so, with NASA’s current plan to land astronauts back on the Moon by 2024, and to have “sustainable (Lunar) exploration” by the late 2020s, planetary EVA is likely to become much more common in the near future (NASA, 2020).

On every Apollo mission that successfully landed astronauts on the moon, at least one EVA was performed. The EVAs of the early Apollo missions (11 and 12) were dedicated primarily to engineering and testing-related tasks, while those of the later Apollo missions (14-17) focused much more heavily on scientific objectives. EVA traverse routes were planned long in advance of the actual mission, using information derived from photographic imagery and topographical maps (Muehlberger, 1981).

Of particular interest in understanding the difficulties of planning and conducting a walking EVA is the second EVA of Apollo 14. The intended goal of this EVA was to explore and perform geological studies at the edge of Cone Crater. Cone Crater resided approximately 1.3 km away from the Lunar Module (LM), making Apollo 14’s EVA-2 the longest walking EVA ever performed. The planned route of the EVA is shown in Figure 2.18.

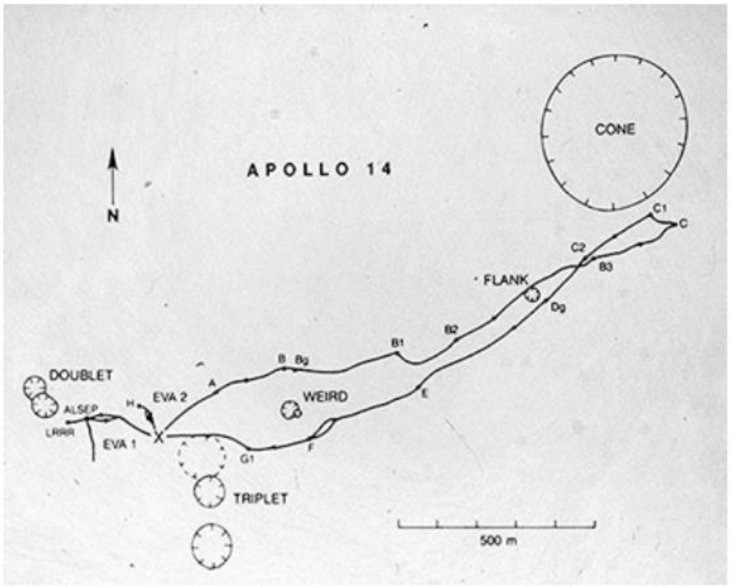


Figure 2.18: Planned Route for EVA-2 in Apollo 14 (Johnson et al., 2010)

As they traversed the lunar surface on foot, astronauts Alan Shepard and Edgar Mitchell became increasingly uncertain of their exact location. The lack of atmosphere on the Moon combined with the monochromatic nature of the terrain made depth perception and landmark identification very difficult. Fatigue also became a problem for the astronauts as they ascended steep slopes that were never intended to be part of the traverse. Because of schedule and safety concerns, the EVA was ultimately aborted before the astronauts were able to reach Cone Crater; they were only about 100 meters from their destination. A quote from Mitchell illustrates well the difficulties and confusion he and Shepard experienced during the EVA.

> *You'd say, 'Well, this next big crater ought to be a couple of hundred meters away, or 100, or 150.' It just wasn't anywhere in sight… You could not get enough perspective from any one spot to pin down precisely where you were. The undulations over the neighborhood were probably 10 to 15 feet (high)… It looked like we were in a large group of sand dunes (Jones, 1995).*

Apollo 14 was the last Apollo mission to feature an EVA that required astronauts to travel large distance on foot; all subsequent Apollo missions (Apollo 15-17) utilized a Lunar Roving Vehicle (LRV). Though the LRV allowed astronauts to travel more quickly than they otherwise could have on foot, the maximum range of an EVA was still very much limited by the walking capabilities of a suited astronaut. Each LRV-equipped EVA accounted for the possibility that the LRV might malfunction or become inoperable during a traverse by enforcing a "walk-back" constraint. This constraint required that – at any point during a traverse – a suited astronaut would be capable of and have sufficient resources to return safely on-foot to the LM. An unplanned event during the Second EVA of Apollo 17 resulted in a situation where astronauts Eugene Cernan and Harrison Schmitt nearly violated their walk-back constraint. While surveying a mission site, Cernan and Schmitt noticed a patch of "orange soil". The astronauts and mission crew deliberated and ultimately elected to collect an unplanned sample of the soil. Because of low consumable levels and the walk-back constraint, the original objectives at the site had to be abandoned (Jones, 1995).

2.3.4.3 Analogues

The most recent execution of a planetary EVA was during the Apollo 17 mission in 1972; that was nearly half a century ago. Since that time, many new technologies and processes have been developed that have the potential to alter the way an EVA is planned and executed. Because foreign planetary bodies remain inaccessible, researchers must rely on analogue studies to determine which of these recent developments ought to be incorporated into a modernized EVA plan. Abercromby’s Desert Research and Technology Studies (DRATS) was one such analogue study. The primary objective of DRATS was to evaluate the capabilities and habitability of the Space Exploration Vehicle (SEV): a pressurized, wheeled vehicle designed for extended use during planetary EVA (Abercromby, Gerhnhardt, & Litaker, 2012). NASA Extreme Environment Mission Operations (NEEMO), an analogue based in an underwater research facility, allows researchers to simulate different gravity levels through the use of various weights and buoyancy effects. At NEEMO, researchers can evaluate EVA procedures under the gravity conditions in which they will actually be performed.

Of particular import to this paper is the work performed as part of the Biologic Analog Science Associated with Lava Terrains (BASALT) study. One of the primary goals of BASALT was to evaluate concepts of operations (ConOps) for future planetary EVAs. In particular, BASALT sought to develop these ConOps under Mars-like conditions for EVAs with science-based objectives. Harmoniously, another of BASALT’s primary objectives was to perform legitimate research relating to the ability of volcanic terrain to host life. BASALT’s dual-pronged approach (i.e. having scientific and operational objectives) placed properly motivated subject matter experts in conditions very close to those that will likely be experienced by future astronauts during planetary EVA. Operational results from the BASALT field studies indicate that grouping tasks according to whether or not they require feedback from experts at MSC can increase the efficiency (i.e. reduce the down-time of EV crewmembers) of an EVA. The developed ConOps recommend that the EV crew first perform tasks that require analysis and input from ground, and afterwards while waiting for a response (sometimes over an extended time delay) perform those tasks that they can accomplish independently.

Each test EVA performed during the BASALT studies had subjects traverse to a region containing several close (<100m apart) sites of geologic interest. An example of a planned traverse from the BASALT-1 study is shown in Figure 2.19.

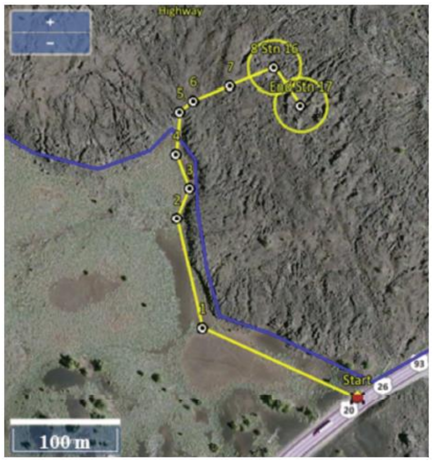


Figure 2.19: Example (in yellow) of BASALT-planned traverse (K. H. Beaton et al., 2017)

The BASALT-proposed structure of an EVA traverse is outlined in Table 2.1.

Table 2.1: Summary of the structure of an EVA traverse as proposed by the BASALT project.

	Time	*Summary*	*Tasks*
Approach	Varied	Travel to site of scientific interest	Traverse, record still imagery and video, look for TOPs
Contextual Survey	~5 min	Collect and record basic information about site	Position cameras, provide basic (verbal) description of site – color, presence of water, etc.
Candidate Sample Location Search	~30 min	Quickly locate samples that may require further investigation. Mark these samples and send information back to ground.	Find samples that meet science objective, place markers, take photographs
Pre-Sample	~60 min	MSC instructs crew to do follow up on several of the candidate samples	Take more pictures, use science instruments to gather more information
Sample	~30 min	MSC instructs crew on which of the pre-sampled locations to actually take samples from	EV crew collect suite of 7 replicates (rocks) of various sizes, places in storage

Steps 1-3 were performed for each site in succession and could be performed without support from MSC. As they gathered contextual and candidate sample information on a specific site, EV crew would send that information (over a time delay) to MSC. MSC would then process, discuss, and make decisions using that data while the EV crew continued on to the next site. By the time the EV crew finished their contextual surveys and candidate sample searches for all sites, MSC would have prepared a pre-sample plan for the EV crew. The EV crew then carried out the MSC-proposed pre-sample plan, sending additional information back to MSC as they did so. Once the EV had completed pre-sampling, MSC would have prepared a list of sites from which to take actual samples. The EV crew collected samples from the designated sites, then traversed back to base (K. H. Beaton et al., 2017).

3. Technical Design

3.1 Overview

The software used and created for this work consists of two primary components: a server and a client. The server is the current iteration of the SEXTANT tool, and the client is a newly designed application called *Pathfinder*. SEXTANT and *Pathfinder* communicate via a TCP socket connection; while this means the two applications could theoretically interact while being hosted by two different devices, for this project they ran alongside one another on a single Microsoft Surface Go tablet.

The SEXTANT server handles the loading, storing, and management of all terrain and terrain-related data (e.g. impassible terrain, geographic coordinate systems, etc.). Additionally, once SEXTANT is provided with a start and an end location, it is responsible for performing all optimal path calculations.

Pathfinder is designed to be a frontend application for SEXTANT, displaying the terrain and path information contained within SEXTANT and providing a means for the user to supply inputs to SEXTANT. The original intention was to implement *Pathfinder* as an AR application, but due to the COVID-19 pandemic, testing such a fully-functional AR device – which would require an outdoor environment – was deemed infeasible. Instead, *Pathfinder* was created as a tablet-based 'simulated AR' application that can be used while standing in-place in an indoor location. Testing in an indoor environment allowed the researcher to conduct the experiment remotely, decreasing the risk of COVID-19 transmission. As a simulated AR application, *Pathfinder* accepts inputs common to tablet-based AR systems – such as device orientation tracking – but the environment it presents to the user is entirely virtual. In addition to displaying and interfacing with SEXTANT-stored data, *Pathfinder* also tracks and stores the current location and orientation of the user with respect to the terrain (SEXTANT is generally unaware of the user and their whereabouts).

3.2 SEXTANT

3.2.1 Existing Codebase

Though SEXTANT has at various moments during its long history existed as an application written in Java, Matlab, and other languages, the most recent version of SEXTANT – and the one inherited by this project – was written in Python 2. This inherited version of SEXTANT will, when mentioned in the future, be referred to as SEXTANT-2018. The Python programming language is user friendly, has an incredible amount of community support, and is widely used by academics and developers. For these reasons, we elected to build upon – rather than wholly replace – the SEXTANT-2018 code base. The three primary features of SEXTANT-2018, all of which have been preserved through the modifications and updates made for this project, are detailed below.

The first of SEXTANT-2018's primary features is the implementation of the *GeoPoint* data structure. This data structure manages location information for point-like objects; that is, for objects whose physical dimensions are insignificant on the kilometers-large scale of most terrain models. Of considerable frustration to two dimensional map-makers is that the Earth (and any other large, physical body worth exploring) is not two dimensional. There are many ways to unwrap and flatten the explorable surface of the round Earth, and each one comes with a its own set of distortions and inaccuracies. For example, the Mercator projection – shown in Figure 3.1 – famously enlarges the land masses closer to the North and South Poles (note how Greenland as drawn is larger than South America).

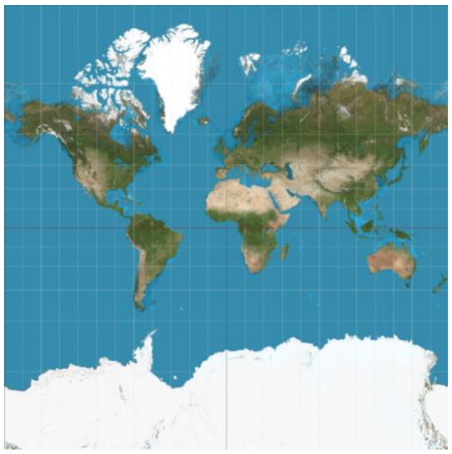


Figure 3.1: Mercator projection of Earth between 85°S and 85°N (D. R. Strebe, 2011)

Spherical coordinate systems such as latitude/longitude somewhat mitigate the distortion problem, though they too are flawed since the Earth is not a perfect sphere. Because finding a single projection or coordinate system that accurately maps the entire Earth is essentially impossible, a widely adopted practice is to use a set of locally accurate coordinate systems called the Universal Transverse Mercator coordinate system (UTM) (*The Universal Transverse Mercator (UTM) grid*, 1999). UTM divides Earth along its latitude lines into 60 different regions, each with a longitudinal width of 6°. A position on Earth is designated in UTM with three values: the UTM zone itself and the planar, two-dimensional 'easting and northing' coordinates of that position. Easting and northing represent – respectively – how many eastward and northward meters one must travel from the UTM zone's origin (where the meridian of the zone meets the equator) to reach the position in question. Though UTM is very commonly used to describe geographic locations, other systems such as latitude and longitude also appear regularly. In addition, terrain-related data structures (e.g. DEMs) often have their own internal coordinate systems. SEXTANT-2018's *GeoPoint* data structure abstracts the concept of geographic location and coordinates, making calculations and conversions from one system to another a simple task. The related *GeoPolygon* data structure provides similar functionality, though for an entire collection of points (as opposed to just one).

SEXTANT-2018's second primary feature is another pair of data structures: the *GridMesh* and the *GridMeshModel*. A *GridMesh* is an abstract object that represents any type of geographical data that can be modeled as a grid. One example of this – and the most important for work related to SEXTANT – is the elevation data stored in a DEM; another example would be a record of the average rainfall experienced by a region of land separated into a set of equally sized tiles. A *GridMesh* contains a the following data members:

- A *GeoPoint* that designates the northwest corner of the represented region
- A resolution parameter that specifies the distance (in meters) between neighboring grid cells
- A matrix for holding the actual data

As evidenced by this short list, a sole purpose of the *GridMesh* is to hold data. Functionality specific to elevation data (e.g. gradient calculations) is handled by the *GridMeshModel*. *GridMeshModels* are loaded by request from user-specified sub-sections of a *GridMesh*. During initialization, a *GridMeshModel* performs and caches the results from several calculations that make future computations – such as calculating a least cost path – much faster. This data caching, while useful, comes with a cost: a larger memory footprint. Because of this, generally only small sub-sections of a *GridMesh* will ever be loaded into *GridMeshModels* at any given moment. In addition to offering a suite of useful elevation-related functions and methods, the *GridMesh* and *GridMeshModel* also provide a convenient abstraction for the many different elevation data formats (e.g. geotiff). Regardless of the format of the input data, all data processing and manipulation occurs through the *GridMesh* and *GridMeshModel* data structures.

SEXTANT-2018's third and final primary feature is the one it was built for: the ability to compute least cost paths. To compute a least cost path, a cost function is required (i.e. a function that computes the cost of moving from one grid position to a neighboring position). SEXTANT-2018 has cost functions for computing the distance, time, and metabolic costs for human explorers. Though past versions of SEXTANT supported cost functions that accounted for complex inputs such as sun angle, SEXTANT-2018's cost calculations are dependent only on the terrain gradient and suited weight of the explorer. Even so, SEXTANT-2018 allows users to easily modify existing or create entirely new cost functions. Using whatever cost function has been specified, SEXTANT-2018 calculates optimal (i.e. minimum cost) paths using the A* search algorithm (P. E. Hart, Nilsson, & Raphael, 1968). After calculating an optimal path, SEXTANT-2018 analyzes and provides information on that path such as distance, metabolic cost, and total expected traverse time.

3.2.2 Upgrade to Python 3

As previously stated, SEXTANT-2018 was written in Python 2. Python 3 – which as of January 1, 2020 is the only officially supported version of Python – would have been used to create the software, but unfortunately a Python 3 compatible version the important Geospatial Data Abstraction Library (GDAL) package was not available at the time. When work began on this project, however, the GDAL package had been appropriately upgraded. With no additional Python 2 dependencies, SEXTANT was upgraded to Python 3 using the official 2to3 application (along with a few spot fixes as needed).

3.2.3 Reducing Path-Finding Calculation Time

One of the primary deficiencies of SEXTANT-2018 – particularly with regard to its ability to allow for mid-traverse path recalculations – was how long path planning took. On our test machine (Dell Latitude E7270, Intel Core i5-6300U CPU, 16 GB RAM, Windows 10), optimal path calculations could sometimes take over one minute. Long path calculations and re-calculations would lead to longer traverse times, which in turn would leave less time to perform science and the designated geological stations. In addition, in-field users would likely become frustrated with an application that took minutes to perform its primary task. For these reasons, one of the primary goals of this work was to decrease the runtime of the path-finding algorithm.

3.2.3.1 Trade Study

Norheim (2018) acknowledged that the runtime performance of SEXTANT needed to be improved before it could reasonably be used in a real-time scenario. He also suggested that such a performance increase might be achieved through use of a better (or at least better suited to the needs of SEXTANT) path-planning algorithm. As a preliminary step to determine which algorithms might be good potential candidates to incorporate into SEXTANT, we elected to perform a trade study of existing optimal search algorithms. The results of that trade study are shown in Table 3.1 (Abd Algfoor, Sunar, & Kolivand, 2015; Connell & La, 2017; Ferguson & Stentz, 2005; Koenig & Likhachev, 2002; Nash, Daniel, Koenig, & Felner, 2007).

Table 3.1: Results from Path Finding Algorithm Trade Study

Name	*Graph Type*	*Key Points*	*Runtime*	*Memory Footprint*	*Optimal Path?*
D / D* Lite*	2D Square Grid	Designed to better handle mid-traverse path recalculations (stores and uses optimality information from initial finds)	Same initial find as A*, orders of magnitude faster for re-calculations	large	Yes
*Field D**	2D Square Grid	Movement is no longer restricted to eight directions. paths look more "natural"	1.7x that of D*	large	Not guaranteed, but on average path cost is less than D*
*Theta**	2D Square Grid	Essentially a better version of Field D*	slower than D*, better than Field D*	large	Not guaranteed, but on average path cost is less than D* or Field D*
TA, TRA**	Tri Grid	By varying the sizes of objects, the authors examined the effect of the environment on movement during a navigation task	-	-	-
*RRT**	2D Square Grid	Excellent for handling dynamic scenarios, such as obstacle avoidance in drones	Very fast	-	No, may be highly non-optimal

In addition to the algorithms shown in this table, several other families of algorithms involving different graph types were explored (e.g. hexagonal). Because terrain elevation information is so often stored in a DEM, and because DEMs are organized as two-dimensional square grids, we ceased considering such algorithms after only a short time. Rapidly-exploring random tree (RRT) algorithms are seeing broad use and are being discussed heavily in current optimal search and path-planning research. Because of this, we were initially excited at the prospect of incorporating an algorithm from the RRT family into SEXTANT. After reviewing relevant literature, however, it became clear that RRT is most useful in truly dynamic scenarios where the positions of environmental obstacles are changing rapidly (e.g. obstacle avoidance for autonomous drones). RRT is designed to find as quickly as possible any clear path to a goal location; the optimality of the found path is of secondary importance. Because obstacles in environments where SEXTANT will typically be used are of the static, geologic variety (though possibly unnoticed during pre-traverse surveys), we believe that RRT and its prioritization of an immediate, non-optimal result was not a good fit. Ultimately, we came to the same conclusion as Norheim: some algorithm from the D* (dynamic A*) family would be most appropriate for use in SEXTANT. The actual paths computed by D* and its ancestor A* are identical (assuming identical cost functions and inputs); the runtimes of their initial finds are also essentially the same. Where D* really outpaces A* is during path re-planning; D* was designed specifically to handle situations where a previously unknown obstacle is encountered in the middle of a traverse. D* intelligently saves any calculated information that remains valid even with the presence of the new obstacle, and by doing so reduces tremendously the time it takes to re-calculate a new path. The situation for which D* is designed to work best is the exact situation we were hoping to modify SEXTANT to accommodate, and because of this it seems to be a good choice. Field D* and Theta* - two derivations of basic D* - offer appealing alternatives to their common ancestor. Both algorithms still offer the same path-recalculation speed so relevant to SEXTANT's needs, and both also offer slight modifications to the geometry of the calculated path. While movement from one grid location to another in D* is restricted to eight directions (N, NW, W, SW, S, SE, E, NE), neither Field D* nor Theta* possesses such a constraint. Because of this, the paths produced by Field D* and Theta* are on average slightly less costly and appear more "natural" than those produced by basic D*. This "natural" look of paths computed by Field D* is well-represented by the image in Figure 3.2.

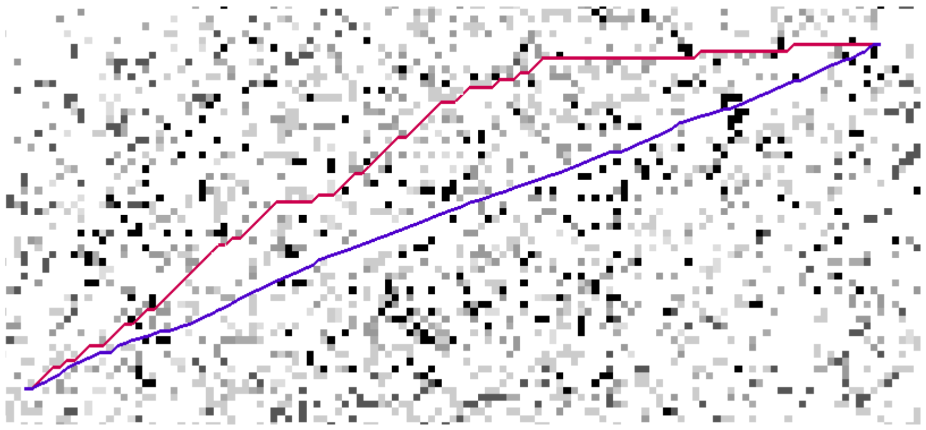


Figure 3.2: Paths produced by D* (red) and Field D* (purple)

Ultimately, even though D* seems as though it would work well within SEXTANT, we decided to leave A* as the primary path-finding algorithm. The reasons for this choice are discussed in sections 3.2.3.2 and 3.2.3.3.

3.2.3.2 Moving Algorithm Runtime to C++ Module

Although it was determined that SEXTANT's path re-planning runtime performance would likely be improved by switching to D*, we elected to first profile the existing codebase to determine if implementing a few basic software-level optimizations would sufficiently increase runtime performance. The profiler revealed – not unexpectedly – that the vast majority of the time spent generating a path took place inside the A* loop. Though Python is an easy to understand and use programming language, it is not known for its speed. The implementation of Python used for this project – often referred to as CPython – is written in both C and Python. CPython (and Python implementations in general) has several features, such as automatic memory management via reference counting, that ease the task of programming and reduce the chance of creating bugs. Though useful, many of these "convenience" features make any given operation in Python slower than that same operation written in C or C++. In an attempt to reduce the overall runtime of the A* loop, we wrote an external module using C++ that wholly contained the processing of the A* algorithm. This external module was created using the *pybind11* library. Table 3.2 presents results of a performance evaluation, which indicates the level of performance increase gained through the creation of the external module.

Table 3.2: Runtime test results for Python and C++ implementations of A*

Path Information	*Python A**	*C++ A**
283m Path, 1000x1050 grid	437 ms	307 ms
930m Path, 1000x1050 grid	7.88 s	886 ms
1730m Path, 1000x1050 grid	48 s	1.62 s
1938m Path, 1000x1050 grid	42.7 s	1.78 s

The length of a path and the size of the grid are not the only factors that contributes to path-generation runtime; obstacle complexity and large variations in the cost output of the cost function can also play a large role. In addition, the scheduling choices made by the CPU of the machine running the tests will also affect runtime. Even so, these preliminary test numbers indicate at least an order of magnitude improvement in runtime. Ultimately, it was decided that this performance increase was sufficient for accommodating real-time, mid-traverse path recalculations; adding an implementation of any version of D* to SEXTANT was deemed unnecessary. However, future studies should consider adding to SEXTANT either field D* or Theta*, as both are likely to produce more natural-looking (i.e. less jagged) paths.

3.2.3.3 Data Caching

One of the most time-expensive operations of the A* algorithm is the computing of costs from a given grid location to all (usually) eight of its neighbors. Interestingly, all of the cost values of a given DEM will remain the same, even if new obstacles are added. Because of this, it is possible to precalculate and store all cost values when a piece of terrain is first loaded. This changed turns what was formerly a calculation into a simple array lookup. By precalculating and caching cost data, runtime performance was improved, but at the price of increased initial load times and memory usage. We were able to achieve further runtime performance increases by caching a per-grid-position value called the 'heuristic cost'. The heuristic cost of a grid cell is an educated guess at the cost of traversing from that cell to the goal location. A* uses these heuristic cost values to bias searches in the direction of a goal node. A cell's heuristic cost remains valid so long as the goal position does not change. Since changes in the position of the goal are rare, the heuristic costs cache seldom needs to be recalculated.

3.2.4 Manual Adding and Removing of Obstacles

In addition to the cost functions, the shape of a SEXTANT path is also controlled by a set of impassable grid locations known as obstacles. Though some A* implementations represent obstacles as normal grid cells that can only be traversed to by incurring extraordinarily large costs, SEXTANT-2018 implemented obstacles using a data structure (a matrix of ‘can enter’ or ‘cannot enter’ Boolean values) wholly independent of cost. Obstacle designations in SEXTANT-2018 happened only once – during the initial load of a DEM – and these designations depended only on the gradient of the grid cell in question. A cell’s gradient is computed using the gradient method from the *numpy* Python library. *Numpy* computes gradients “using second order accurate central differences in the interior points and either first or second order accurate one-sides (forward or backwards) differences at the boundaries” (Oliphant, 2020). If the gradient of a cell exceeds some pre-defined maximum, that cell is designated as an obstacle. One of the major goals of this work is to modify SEXTANT to accommodate the mid-traverse designation and subsequent routing around of previously unknown obstacles. In order to accomplish this, we added to SEXTANT methods for adding sets of cells to and removing them from the matrix of designated obstacles. The functionality to automatically designate obstacles based on gradient was preserved.

3.2.5 Persisting Terrain and Path State

SEXTANT-2018 was more of a library than a standalone application. It provided functionality for DEM loading and path finding to external applications such as the xGDS and the SEXTANT WebApp, but kept no internal state of its own. Neither xGDS nor the SEXTANT WebApp have been updated since the completion of Norheim’s work in 2018. For path re-calculations to function well, certain data must persist beyond the completion of the path-finding algorithm. Due to this necessity, we modified SEXTANT to be a stand-alone application that maintains the terrain state (cost and heuristic caches, obstacle matrix, etc.) within its own block of application memory.

3.2.6 Conversion to Client-Server Model with TCP Socket IPC

As an application, SEXTANT is meant to read and process terrain information. Python is a good language for performing such tasks, and – as evidenced by the A* C++ external module – can be made to conform to performance needs when required. Though Python does have libraries for creating information displays, as a language it is less well-suited for such visual tasks. Additionally, implementing a means of displaying path and terrain information directly within the SEXTANT codebase would be needlessly limiting. For these reasons, we chose to change SEXTANT's architecture into that of a server.

When loaded, the SEXTANT server immediately creates a listen socket that waits for client connection requests. When a connection request is received, SEXTANT creates another TCP socket that handles all future communications with the just-connected client. With the personal socket connection established, the client makes requests to the server (e.g. fetch terrain data, find path between A and B, etc.) and the server creates and returns an appropriate response.

3.3 Pathfinder

3.3.1 Overview

Our original intention for *Pathfinder* was to build upon the Holo-SEXTANT work done by Anandapadmanaban (2018) and Norheim (2018); that is, we wanted to use the learnings garnered from Holo-SEXTANT to create an augmented reality tool that could be used and tested by researchers in the field. Due in part to time constraints and in part to the COVID-19 pandemic, we unfortunately were unable to construct such a field-ready application. Instead, we built *Pathfinder* into a tool to be used for evaluating visualizations, interface elements, and application features that field geologists (or any person who frequently traverses rugged terrain) might find useful. When we made the decision to transition *Pathfinder* from a field-usable to an evaluation tool, we also elected to transition the application from AR to VR. We reasoned that VR applications are simpler to implement, have fewer technological dependencies, and are easier to test (both from a software and a research perspective). In order to keep user interactions as similar as possible to what they would have been in a proper AR application, we chose to incorporate input mechanisms common to AR (e.g. device tracking). In addition, *Pathfinder* maintains the first-person viewing angle inherently present in all AR applications. That is, the view depicted on the tablet screen is the view that the user would 'see' if they occupied the virtual space depicted by *Pathfinder*.

As an application, *Pathfinder* has two primary functions. The first is to help a user navigate through rugged (virtual) terrain from a start location to an end location, and the second is to allow a user during a traverse to designate certain sections of terrain as being non-traversable (i.e. mark them as 'obstacles'). Though distinct, these two functions are intended to interact and work closely with one another. *Pathfinder* creates a path visualization based on the elevation and obstacle information it has at startup. If at any point a user specifies that a certain portion of the terrain is an obstacle, *Pathfinder* will update the displayed path visualization to reflect the updated terrain features.

Because of its robust library of AR and VR support, an extremely active development community, and personal familiarity, we chose to use the Unity game engine to develop *Pathfinder*. Multi-platform support is something that Unity provides "out of the box", meaning that *Pathfinder* would likely work on a broad array of devices. Even so, *Pathfinder* was developed for and exclusively tested on a Microsoft Surface Go tablet. A screenshot from *Pathfinder* with interface element callouts is shown in Figure 3.3.

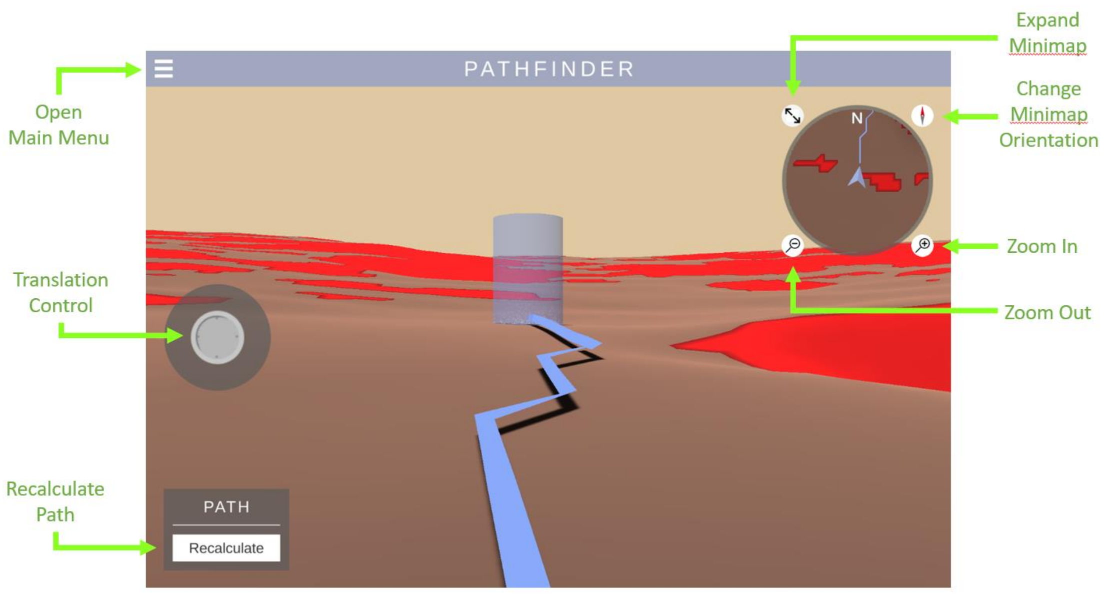


Figure 3.3: Screenshot from the *Pathfinder* application. Major elements of the user interface are called out with green arrows and text labels.

3.3.2 Terrain

The virtual world displayed by *Pathfinder* is a simple visualization of whatever DEM is currently loaded by SEXTANT. Each grid cell of the displayed DEM corresponds to a vertex in a programmatically-generated mesh. The z and x coordinates of a vertex are – respectively – the resolution-scaled row and column indices of the grid cell it represents. The y-coordinate of the vertex is the elevation value stored in the grid cell. Standard (i.e. non-obstacle) grid cells are assigned a vertex color of brown, while obstacle grid cells are given a red vertex color. The results of this positioning and coloration system is shown in Figure 3.4.

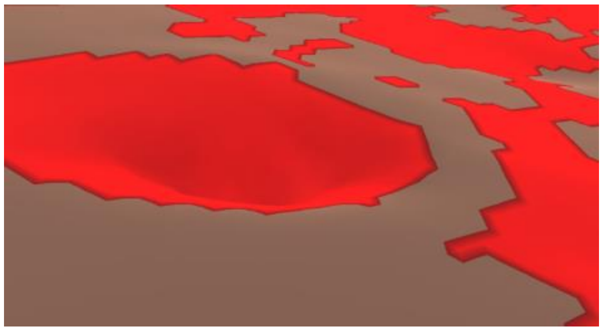


Figure 3.4: Terrain as drawn by *Pathfinder*. Brown regions are those deemed by the SEXTANT server to be passable, while red regions are impassable obstacles.

3.3.3 Controls

A user interacts with *Pathfinder's* virtual environment in two primary ways: by moving through it, and by designating regions as obstacles. The latter will be discussed in section 3.3.5. In *Pathfinder*, movement through the virtual world entails both changes in position and changes in orientation. Position changes are performed using a virtual thumbstick (shown in Figure 3.5).

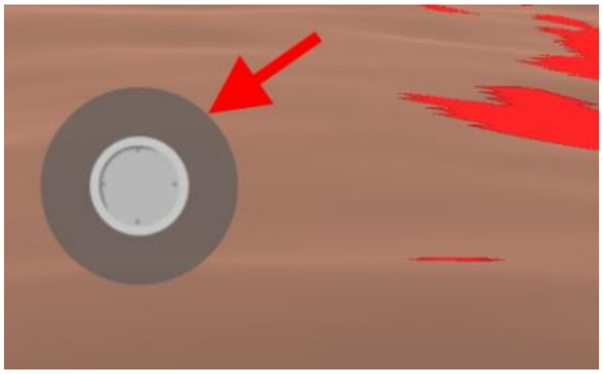


Figure 3.5: Virtual thumbstick used to affect the position of the user in *Pathfinder*

Tilting the thumbstick up, down, left, and right translates the user forwards, backwards, left, and right, respectively. All translations are with respect to the direction the user is currently facing. Like all stick-based input devices (e.g. a joystick), the virtual thumbstick does not restrict movement to the four cardinal directions. For example, tilting the thumbstick in a direction 30 degrees from ‘up’ will induce a corresponding translation in a direction 30 degrees from the user’s current heading.

There is no on-screen interface element for altering orientation. Instead, changing the orientation of the tablet directly changes the 1st-person viewing direction. Yaw, pitch, and roll changes of the tablet correspond 1-to-1 with yaw, pitch, and roll changes to the viewing direction. A diagram illustrating the recommended way of achieving viewing direction changes is shown in Figure 3.6.

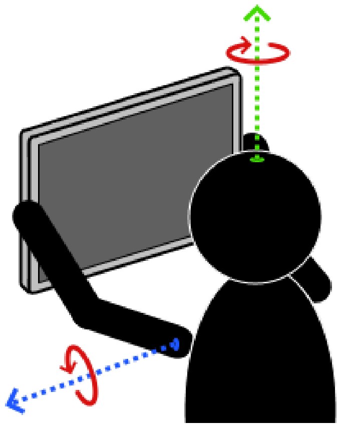


Figure 3.6: The *Pathfinder*-recommended way for a user to move their body in order to change the orientation of the tablet and in turn affect the virtual viewing direction. Changes in yaw are achieved by rotating in place, while pitch changes are achieved by tilting the arms up and down.

3.3.4 Path-Following Visualizations

As discussed previously, one of *Pathfinder’s* primary functions is to help users follow a calculated path from a starting location to an ending location. It does so by presenting to the user one of several different path visualizations. For the purposes of *Pathfinder*, a ‘path’ is simply a list of two-dimensional coordinates. Each coordinate pair corresponds to a grid cell in whatever DEM is currently loaded. Because of this, the almost singular goal when designing *Pathfinder’s* different path visualizations was to find meaningful ways to represent a series of connected three-dimensional points.

The most obvious way to represent a *Pathfinder* path is by connecting every point in that path with a series of line segments. This visualization was employed by Holo-SEXTANT, as can be seen from Figure 2.10. *Pathfinder's* implementation of the visualization, which was dubbed the 'line' visualization, is shown in Figure 3.7.

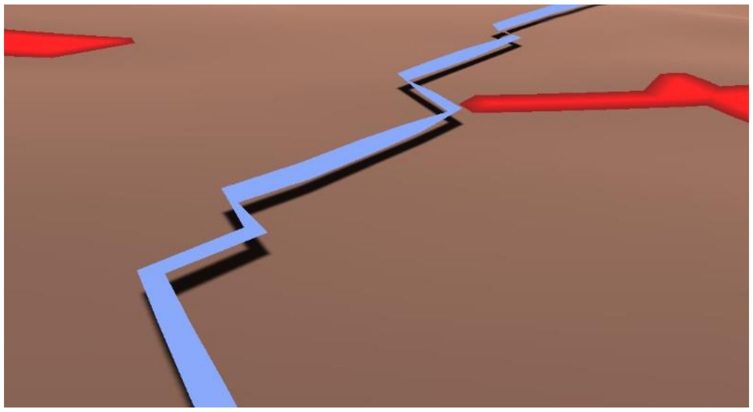


Figure 3.7: *Pathfinder's* line visualization. Line segments connect every point in the path.

As this figure shows, a path as displayed by the line visualization is rarely straight. This is a byproduct of the A* pathfinding algorithm and its ability to move only in the four cardinal and four inter-cardinal directions (e.g. N, NW, W, etc.). Smoothing the jaggedness of the line visualization by using the path points as control points of a spline was considered but never implemented.

A *Pathfinder* path is only as good as the precursor data used to generate it. DEMs may have inaccuracies or have a less-than-desirable resolution, or information on terrain composition (e.g. sand or dust content, which can make traversal difficult) may be unavailable. Additionally, as was noted in previous sections, visual and tactile observations made by path-walkers during a traverse will sometimes be used to override the original design of a path. Knowing this, we suspected that the surplus of information supplied to path-walkers through the line visualization (i.e. showing them every calculated path point) was somewhat misleading; the line describes a level of informational accuracy that simply is not present. In an attempt to create a visualization that would more accurately represent the fidelity of the data upon which it was built, we created a 'waypoint' visualization. For the purposes of *Pathfinder*, a waypoint is a visual icon that is assigned and appears at a location in three-dimensional space. A waypoint is intended to serve much the same purpose as does a cairn on a real-world hiking trail; that is, it is intended to generally guide a user along a route, while allowing that user to manage route micro-decisions (e.g. foot placement, etc.) themselves. The three-dimensional icon we chose to represent a waypoint is shown in Figure 3.8.

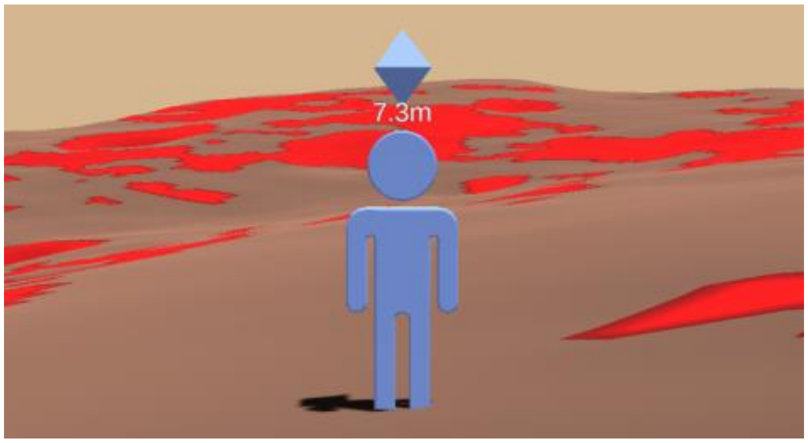


Figure 3.8: Waypoint icon used in *Pathfinder*

We chose to use an abstracted human form – which in *Pathfinder* world units is 1.8m tall – in order to give users a familiar object that they could use to better gauge distances. Though all waypoint positions are calculated when a path is first calculated, only three are ever drawn (drawing all waypoints would add unnecessary visual clutter). The waypoint to which the user is next intended to traverse looks like the humanoid form in Figure 3.8, while the two subsequent waypoints are drawn as semi-transparent octahedrons. When the user moves sufficiently close to the humanoid waypoint, it disappears and the new 'next' waypoint – which moments before was a semi-transparent octahedron – transforms into its humanoid waypoint form.

Of great import when using waypoints is determining what algorithm to use for selecting which of the many points in a path should be assigned a waypoint. For *Pathfinder*, we used two different algorithms. The first – and simpler – of the two algorithms is called 'equidistant' waypoint placement. In equidistant mode, each waypoint is placed an equal distance away from both of its neighboring waypoints (previous and next). That is, the distance separating each waypoint in the path is identical. The algorithm is illustrated by the image in Figure 3.9.

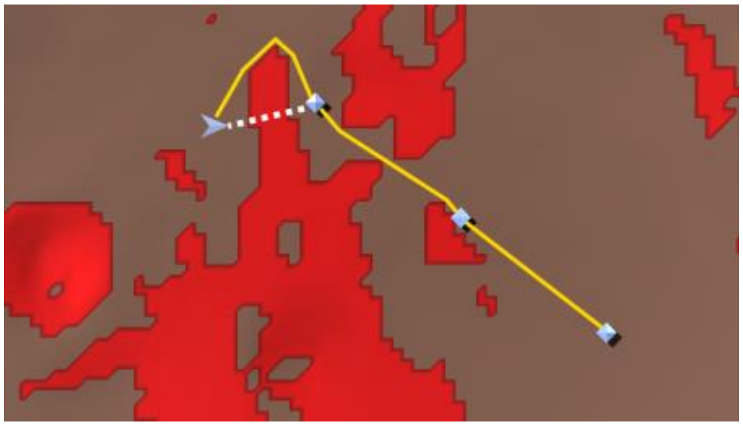


Figure 3.9: Illustration of the ‘equidistant’ method of waypoint placement. The distance traced by each of the yellow lines is approximately equal.

Figure 3.9 emphasizes the foundational feature of the equidistant waypoint placement method: if straightened, the yellow line segments connecting the waypoints are of approximately equal length.

The second method of waypoint placement is called ‘clear path’. In clear path mode, a next waypoint is placed such that a straight line path between it and its predecessor is clear of obstacles. That is, a straight line drawn between any two adjacent waypoints never goes through an impassable region. The algorithm is illustrated by the image in Figure 3.10.

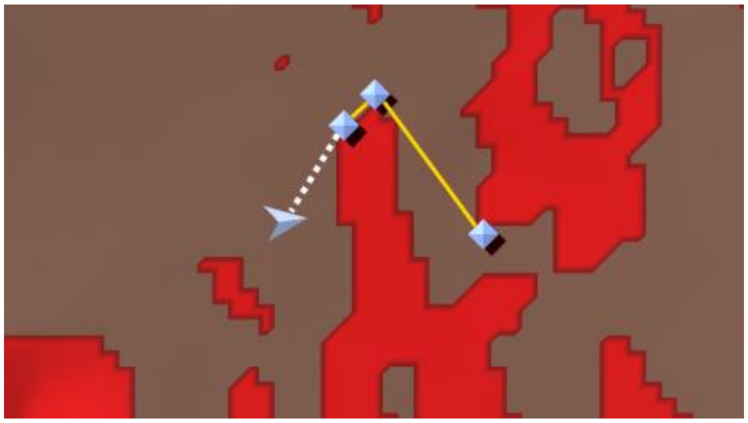


Figure 3.10: Illustration of the ‘clear path’ method of waypoint placement. The lines between the user and next waypoint (dashed white line) and those connecting subsequent waypoints do not pass through an impassable region.

Note that neither the yellow line segments connecting the waypoints nor the straight line connecting the user and next waypoint (dashed white line) goes through an impassable region.

In addition to the actual path visualizations, we added several features to *Pathfinder* to assist the user with following a designated path. The waypoint visualizations both come with a compass, which is shown in Figure 3.11.

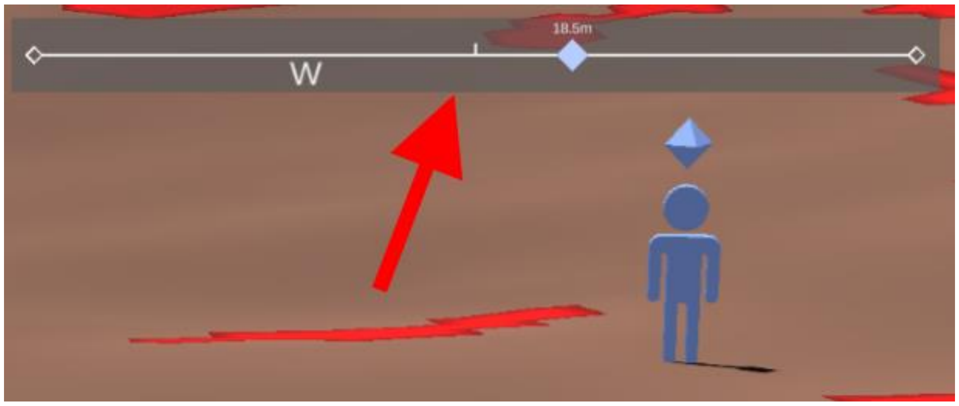


Figure 3.11: *Pathfinder's* Compass widget, which can optionally be enabled when using a waypoint visualization. Cardinal directions and waypoints both appear on the compass.

The compass indicates the position of the next waypoint. The bar of the compass represents a 60 degree 'cone of vision' directly in front of the user. If the next waypoint falls within this cone, it is displayed along the bar as blue diamond. If it falls outside the cone, the blue diamond locks to one of the extreme ends of the compass bar and sprouts a small arrow that indicates the direction the user must turn in order to see the waypoint. Another feature – the offscreen waypoint indicator – serves a similar purpose to the compass. Shown in Figure 3.12, the indicator is intended to help users locate a waypoint if it falls somewhere outside of the current viewing area.

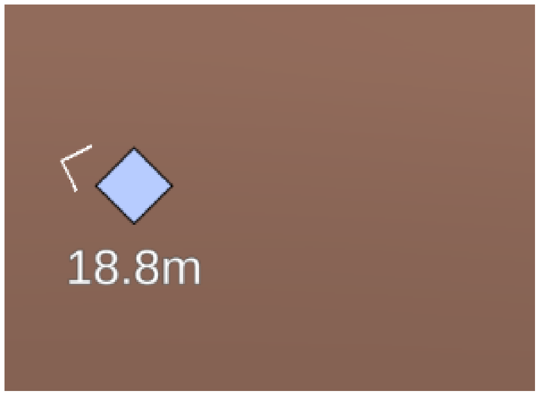


Figure 3.12: The offscreen waypoint indicator appears whenever the next waypoint falls outside of the current viewing area; it points in whatever direction the user must turn in order to see the waypoint.

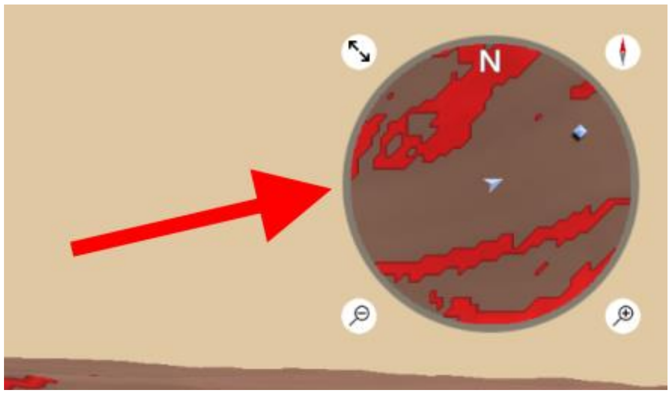


Figure 3.13: *Pathfinder's* minimap, which shows a top-down view of the user's current location.

Regardless of what path visualization is currently in use, *Pathfinder* provides users with a minimap (shown in Figure 3.13). The minimap displays an overhead view of the user and their nearby surroundings; it is always centered on the user's current location. Regular terrain and obstacles are shown in the minimap, and – depending on the currently active path visualization – the line or waypoints will also appear. The minimap has zoom-in, zoom-out, maximize, and minimize functionality, and its orientation can be toggled between 'north oriented' and 'user oriented'.

3.3.5 Obstacle-Editing Interfaces

Marking certain regions of terrain as either passable or impassable – a process referred to as 'obstacle-editing' – is the second of two (the first being 'movement') primary ways in which a user can interact with the virtual world of *Pathfinder*. In a real-world situation, the need to designate a certain region of terrain as an obstacle would come about naturally. For example, while a path-walker was traversing a path created during mission planning, they may come across a section of terrain that was believed to have been solid rock but instead consists of many small, loose rocks. Not wanting to walk through terrain of this composition, the path-walker could mark it as an obstacle and immediately calculate a new path that avoids the region. In the virtual world of *Pathfinder*, the terrain data used to calculate the path perfectly represents the terrain that is displayed on screen (since both draw from the same source). Because of this, we added 'unmarked obstacle' data into *Pathfinder*. *Pathfinder* draws regions of unmarked obstacles with a special material, making it visually distinct from normal terrain and obstacles. A region of unmarked obstacles is shown in Figure 3.14.

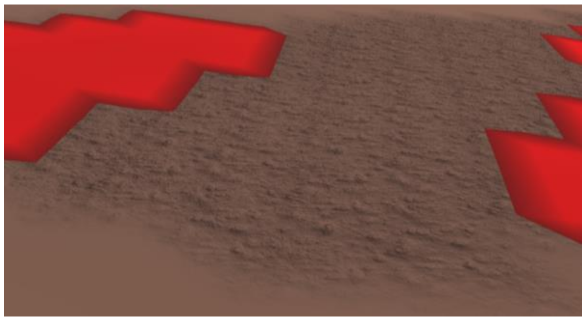
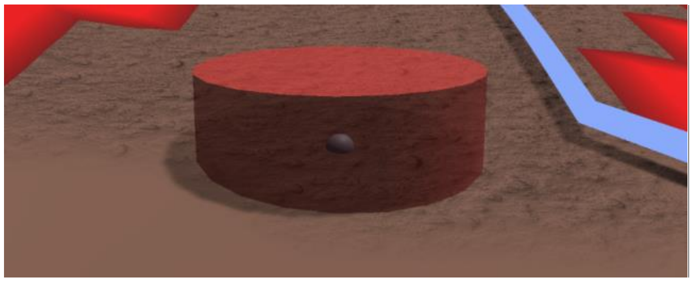


Figure 3.14: A region of unmarked obstacles. A bump map was applied to the mesh to make it appear visually distinct from normal terrain.

SEXTANT's pathfinder algorithm is unaware of unmarked obstacles and because of this is able to calculate paths that go through them. Users are instructed to treat regions of unmarked obstacles as if they were regular obstacles (i.e. they are told to not walk through them). This simulates the data imperfections and inadequacies that would exist in a real-world scenario.

Two different methods for editing obstacles exist in *Pathfinder*: the 'look-at' method and the 'map' method. Both of these methods allow users to not only add but also to remove obstacle designations from terrain. Due to the coloration of obstacles, a user who is marking obstacles appears very much to be using a brush to paint the terrain red. Because of this, many parts of *Pathfinder's* obstacle-editing interface are described using painterly terms. For example, the tool used to add and remove obstacles – shown in Figure 3.15 – is called 'the brush'.



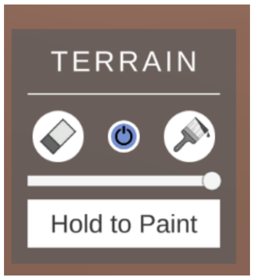


Figure 3.15: (Left) Brush tool used to add and remove obstacles from terrain.

Figure 3.16: (Right) Interface element in *Pathfinder's* heads up display used to active, deactivate, and interact with obstacle-editing brush.

Regardless of the method being used, the state (either painting, erasing, or off) and size of the brush can be adjusted using the interface elements in the 'Terrain' panel of *Pathfinder's* heads up display (shown in Figure 3.16).

When a user activates the look-at mode of obstacle-editing, the brush appears directly in front of the user, on top of whatever terrain happens to be in the center of the screen. The brush remains in the center of the screen as long as look-at mode remains active. Moving the device changes what section of terrain is at the center of the viewing area, which in turn changes what piece of terrain the brush is over. Once a user positions the brush over the desired section of terrain, they begin painting obstacles by tapping and holding the ‘Hold to Paint’ button.

When the map mode of obstacle-editing becomes active, the minimap expands into its enlarged form. This enlarged state is shown in Figure 3.17.

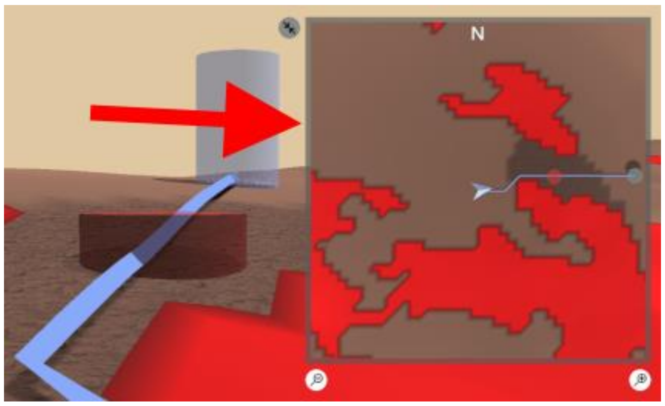


Figure 3.17: Enlarged state of the minimap. When the map mode of editing obstacles is active, users can paint obstacles by dragging their finger over the desired section of terrain in the minimap.

To paint a certain region of terrain, a user touches and drags their finger over the corresponding region in the expanded minimap. The action of adding or removing obstacles using the map mode is very much like finger painting. When map mode is deactivated, the minimap returns to its original size.

3.3.6 Testing

Several important features of *Pathfinder* augment neither the path-following nor the obstacle-editing capabilities of the application, but rather are used exclusively for testing and evaluation. As mentioned in section 2.3.4.1, an important part of geological surveys is to always be on the look-out for targets of opportunity (TOPs). As such, an important evaluation criterion for a path-following visualization is how well it supports the ability to scan for TOPs during a traverse. To support this evaluation criteria, we created a virtual representation of a TOP. This virtual representation is shown in Figure 3.18.

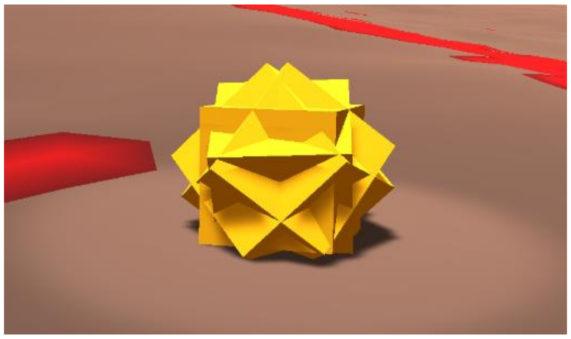
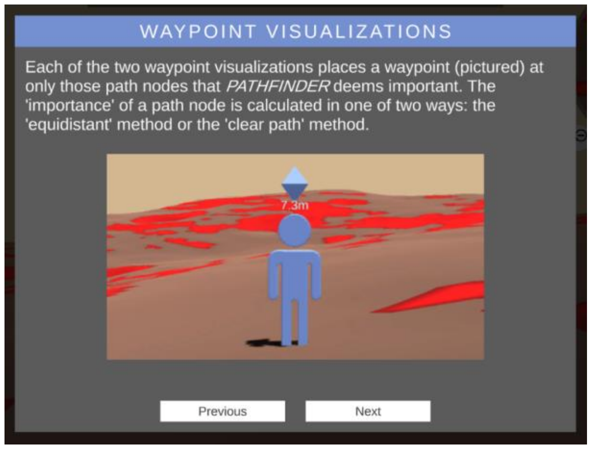


Figure 3.18: A target of opportunity as displayed by *Pathfinder*. During path-following trials, subjects were instructed to maintain a mental count of the number of nearby TOPs.

During a path-following trial, 18 TOPs are placed at pre-determined locations along a traverse. So that too many TOPs are never visible at any given moment, TOPs are programmed to only appear when the user moves within 30m of them. A TOP will again disappear if a user moves out of its 30m range.

In order to help users learn to use the application, a tutorial system was implemented for *Pathfinder*. Two tutorials exist in *Pathfinder*: one for learning how to use the path-following visualizations, and one for learning how to use the obstacle-editing interfaces. Tutorials consist of several learning modules, each of which is preceded by a series of informative dialogues. One such tutorial dialogue is shown in Figure 3.19.



WAYPOINT VISUALIZATIONS

Each of the two waypoint visualizations places a waypoint (pictured) at only those path nodes that *PATHFINDER* deems important. The 'importance' of a path node is calculated in one of two ways: the 'equidistant' method or the 'clear path' method.

Figure 3.19: A tutorial dialogue, this one intended to teach users about *Pathfinder's* waypoint visualizations.

Dialogues are also employed to guide users through the testing process. Before performing a particular trial, a subject is shown a dialogue explaining what is expected of them and what their priorities should be during that trial.

4. Hypotheses

4.1 Gaps

The following two knowledge gaps were identified in the realm of path-planning for human explorers over rough terrain:

GAP 1

Relatively few modes of path display have been implemented and evaluated for AR applications in general and for SEXTANT-based projects in particular. The literature indicates that existing display modes – while useful – still have much room for improvement. In relation to SEXTANT, only a single iteration of an AR interface has been implemented and evaluated.

GAP 2

As of yet, no attempt has been made to create a version of SEXTANT that allows for real-time, in-situ terrain updates (nor has any attempt been done elsewhere in the literature for human traversal over rough terrain). The inability to perform in-situ path re-planning leaves users with a path that is only as good as the data available during the initial planning phase.

4.2 Research Questions

RESEARCH QUESTION 1

What set of visual elements, which when combined depict a pre-calculated path with known start and end locations, …

a. allows users to most quickly traverse to a goal location?

b. helps users to traverse a path that most closely resembles the optimal path?

c. provides the lowest mental workload during path traversal?

d. provides the highest environmental situational awareness during path traversal?

RESEARCH QUESTION 2

When used in conjunction with autonomous path recalculation, what terrain modification interface…

a. allows users to most quickly traverse to a goal location?

b. helps users to traverse a path that most closely resembles the optimal path?

c. allows users to most quickly make terrain modifications?

d. allows users to most accurately mark obstacles?

e. provides the lowest mental workload during terrain modification?

4.3 Hypotheses

HYPOTHESIS 1.A

The line path visualization is the simplest and most information rich of all the path visualizations. In addition, the visual state of the line never changes during a traverse; this means that the user will never unexpectedly lose track of which direction they ought to be heading. Because of this, it is hypothesized that users will complete a traverse most quickly when using the line visualization.

HYPOTHESIS 1.B

The line path visualization displays every point contained by that path. With this complete set of path information, it is hypothesized that users using the line visualization will follow a path that most closely resembles the planned optimal path.

HYPOTHESIS 1.C

The two waypoint visualizations (equidistant and clear path) attempt to provide the user with the minimum required amount of information needed to complete a traverse. The clear path visualization removes any mental effort the user might have to exert with respect to small scale route planning; the next waypoint can always be reached by walking directly towards it. Because of this, it is hypothesized that the clear path waypoint visualization will produce the lowest mental workload.

HYPOTHESIS 1.D

The line visualization, with its surplus of path information, has the potential to provide a constant distraction to the user. The user might be unable to focus on the surrounding environment because the line itself captures too much of their attention. Both waypoint visualizations only require the user to be generally aware of the direction they need to travel, leaving them free to visually observe their surroundings. The equidistant waypoint visualization sometimes requires users to perform small-scale pathing calculations (e.g. if an impassable region is between the user and the next waypoint). Such calculations require that close attention be paid to obstacle boundaries. The next waypoint in the clear path waypoint visualization is always reachable via a straight line trajectory, so users of that visualization will not have to spare any attentional resources to perform small-scale pathing calculations. Because of this, it is hypothesized that users of the clear path waypoints visualization will have the highest environmental situational awareness.

HYPOTHESIS 2.A

The map mode of obstacle-editing allows users to add obstacles to large sections of terrain without having to pause their traverse to open and close the tool multiple times. A user of the look-at mode of editing obstacles may have to close the interface, move to a new position to gain a more-complete perspective of an unmarked obstacle, and re-open the interface several times to paint an entire region. Because of this, it is hypothesized that users using the map mode of obstacle-editing will reach the goal location more quickly than those using the look-at mode of obstacle-editing. Without access to any obstacle-editing tools, users are forced to manually route around unmarked obstacles; this has the potential to be very time consuming. Because of this, it is hypothesized that users that have access to either obstacle-editing mode will reach the goal location more quickly than those who have no access to obstacle-editing features.

HYPOTHESIS 2.B

The map mode of obstacle-editing allows users to add obstacles to sections of terrain to which they are not particularly close (due to the alternate viewport and zoom features of the minimap). The look-at mode of obstacle-editing sometimes requires that the user get very close to the region they wish to paint, which may affect the route that they traverse. Users without access to either obstacle-editing tool will have to route around unmarked obstacles without the guidance of a path visualization. Because of this, it is hypothesized that users using the map mode of obstacle-editing will more closely follow the calculated optimal path than will users of the look-at mode of obstacle-editing, and that users of either obstacle-editing mode will more closely follow the calculated path than those who have no access to obstacle-editing features.

HYPOTHESIS 2.C

The look-at mode of obstacle-editing does not require the user to move their focus away from the main viewing area in order to mark obstacles. The map mode of obstacle-editing requires the user to look from the main viewing area to the minimap, and in doing so requires that users perform a mental frame-of-reference change. These focus changes and required mental calculations will likely increase the amount of time a user spends editing obstacles. Because of this, it is hypothesized that users using the look-at mode of obstacle-editing will perform their edits more quickly than users of the map mode of obstacle-editing.

HYPOTHESIS 2.D

The map mode of obstacle-editing provides users with an excellent perspective for making obstacle edits. The top-down view it provides allows users to easily make out the borders and the shapes of regions of unmarked obstacles. Additionally, map mode's 'finger painting' method of marking obstacles is a natural motion that can be performed precisely. The look-at mode of obstacle-editing requires that a user looks at the main viewing area to edit obstacles. This 'ground level' viewing perspective can make marking specific regions of obstacles – particularly those on downhill slopes or those semi-occluded by other terrain elements – very difficult. Additionally, brush movements in look-at mode are induced by viewing perspective changes. Recall that the viewing perspective in *Pathfinder* is altered with bodily motions based at the shoulders and the waist – these motions tend to be less precise than hand or finger motions. For these reasons, it is hypothesized that users of the map mode of obstacle-editing will more accurately mark obstacles than users of the look-at mode of obstacle-editing.

HYPOTHESIS 2.E

The map mode of obstacle-editing allows users to mark obstacles with the touch of finger. Marking a surface with a finger is common task both in the real world and when using touchscreen applications. The brush controls in the look-at mode of obstacle-editing require users to perform actions not common to everyday life. The arm-tilting and hip-pivoting motions will likely be unfamiliar, meaning that the user will have to expend mental resources in order to concentrate and achieve the desired result. Both obstacle-editing interfaces provide the benefit of not having to mentally calculate a route to the goal location. Because of this, it is hypothesized that a user’s mental workload when using the map mode of obstacle-editing will be lower than when using the look-at mode of obstacle-editing, and that a user’s mental workload while using either of the obstacle-editing modes will be lower than that when they have no access to obstacle-editing features.

5. Methods

5.1 COVID-19

The COVID-19 pandemic of 2020 required the original study design – which had users testing a proper AR interface in the field – to be altered significantly. The original design required human subjects to travel to a specific location and to interact closely with project researchers. Given the governmental urgings to refrain from non-essential travel and to follow social distancing guidelines, we ultimately concluded that conducting our human subjects testing as originally outlined was socially irresponsible.

5.2 Objective

SEXTANT has been modified to allow for real-time, mid-traverse path recalculations and to provide users with the ability to specify sections of terrain as either passable or impassable. The *Pathfinder* frontend was built to evaluate different methods of utilizing and interacting with SEXTANT's new features. *Pathfinder* provides a means for both path-following and real-time obstacle updating. Three different path-following visualizations were developed: line, equidistant waypoints, and clear path waypoints (these are discusses in detail in section 3.3.4). Additionally, two interfaces for obstacle-editing were created: look-at mode and map mode (these are discussed in detail in section 3.3.5). Two separate experiments were performed in this study, one for evaluating the path-following visualizations and one for evaluating the obstacle-editing interfaces.

5.3 Hardware

The device on which we chose to evaluate both the SEXTANT server and the *Pathfinder* frontend was a Microsoft Surface Go tablet. The Surface Go was equipped with an Intel Pentium Gold Processor 4415Y, an Intel HD Graphics 615 GPU, and 8 GB of RAM. An image of the testing device is shown in Figure 5.1.

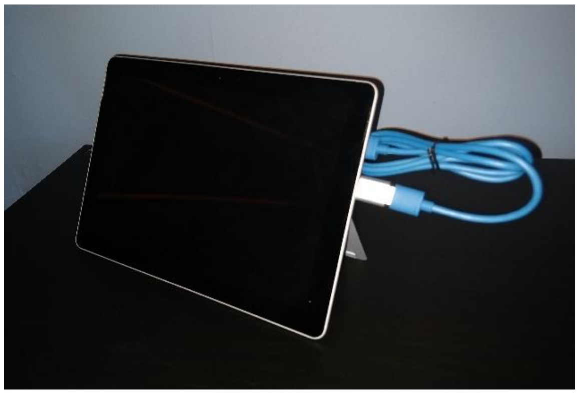
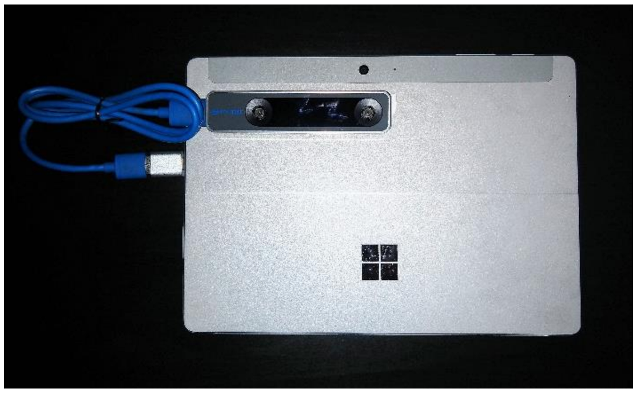



Figure 5.1: (Left) Surface Go tablet used to run SEXTANT and *Pathfinder*. The blue cable connects the tablet to the RealSense T265 motion tracking camera mounted on the back.

Figure 5.2: (Right) The RealSense T265 mounted with a hook-and-loop fastener to the back of the Surface Go tablet.

In order to control their orientation within *Pathfinder's* virtual environment, users are instructed to alter the orientation of the testing device. Such a control system requires a motion tracking device. The motion tracking device we chose to use was Intel's RealSense T265 motion tracking camera. The device is shown in Figure 5.2 mounted to the back of the Surface Go tablet. The T265 is outfitted with two fisheye lens sensors, an inertial measurement unit (IMU), and an Intel Movidius Myriad 2 visual processing unit (VPU). The T265's V-SLAM (visual simultaneous localization and mapping) algorithm is entirely self-contained and runs directly on the VPU. Because *Pathfinder* ultimately became a simulated AR application (as opposed to an actual AR application), we likely could have simply used the Surface Go's on-board gyroscope and accelerometer to accomplish our goals. However, we elected to use the T265 because it had already been integrated into our hardware setup by the time the decision was made to convert *Pathfinder* to a simulated AR application.

5.4 Independent Variables

5.4.1 Path-Following

The path-following experiment had two independent variables (IVs): path visualization and path route. The path visualization IV was comprised of three different levels (the line, equidistant waypoint, and clear path waypoint visualizations discussed in section 3.3.4), and the path route

IV was also comprised of three different levels. Top-down views of the three path routes are shown in Figure 5.3.

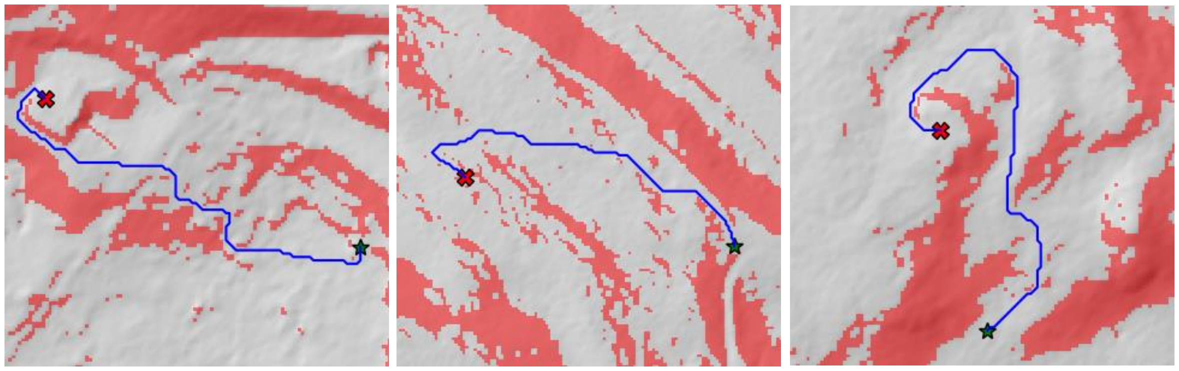


Figure 5.3: The first (left), second (middle), and third (right) path routes used during the path-following experiment.

Each subject received three different path-following treatments. To form the treatments, the order of the three visualizations and the order of the path routes were randomized. The first visualization was matched with the first path route, the second visualization with the second route, and the third visualization with the third route. A single subject always experienced all three path visualizations and all three path routes, though the pairing and the order in which they were received varied from subject to subject. The path routes were all designed to have approximately equal length and traverse difficulty. Path route 1, path route 2, and path route 3 had optimal lengths of 149.7m, 150.6m, and 150.4m, respectively. Each route had wide open sections (i.e. sections largely devoid of obstacles), narrow sections, and sections with many small, interspersed obstacles. Different path routes were used so that subjects would not have the chance to learn any single route over the course of the experiment. The order in which the visualizations were received was randomized in order to mitigate the effects of learning to better use the tool over the course of the experiment, and visualization-route pairings were randomized to mitigate the effects of non-equal route traversal difficulty. If a particular route turned out to be easier to traverse than the others, we wanted to make sure that no single interface received the exclusive benefit of always being paired with that route.

5.4.2 Obstacle-Editing

Like the path-following experiment, the obstacle-editing experiment also had two IVs: obstacle-editing interface and path route. The obstacle-editing interface IV was comprised of three different levels (the look-at and map obstacle-editing interfaces discussed in section 3.3.5, as well as a ‘none’ interface where no obstacle-editing capabilities were present), and the path route IV was also comprised of three different levels. Top-down views of the three path routes are shown in Figure 5.4.

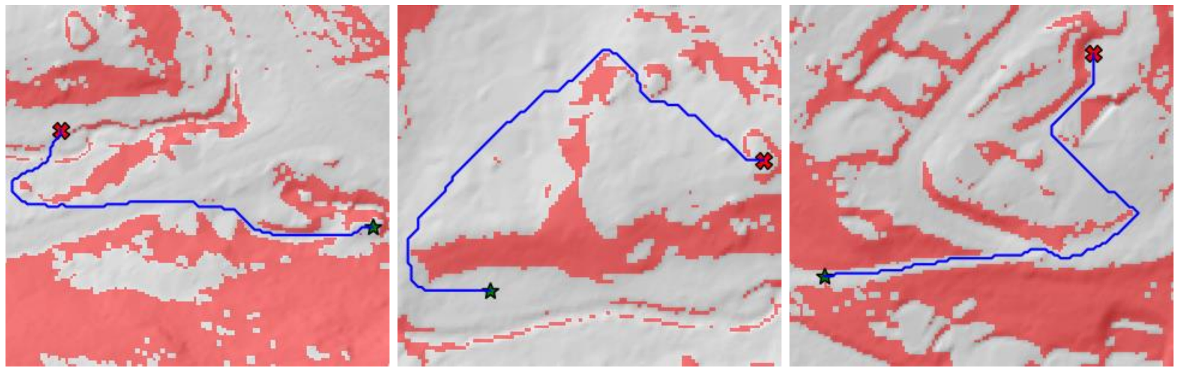


Figure 5.4: The first (left), second (middle), and third (right) path routes used during the obstacle-editing experiment.

Treatments in the obstacle-editing experiment were formed in a similar way to those of the path-following experiment. Both the interface order and the path route order were randomized. The first interface was matched with the first path route, the second interface with the second route, and the third interface with the third route. A single subject always experienced all three obstacle-editing interfaces and all three path routes, though the pairing and the order in which they were received varied from subject to subject. Path routes were all designed to have approximately equal length, traverse difficulty, and areas of unmarked obstacles. Path route 1, path route 2, and path route 3 had optimal lengths of 187.6m, 182.4m, and 189.3m, respectively. Each route was largely comprised of wide-open regions (i.e. regions with few obstacles) and had a single region of unmarked obstacles through which the displayed path would pass. The line visualization was used to display the calculated path in each treatment. The order in which the interfaces were received was randomized in order to mitigate the effects of learning to better use the tool over the course of the experiment, and interface-route pairings were randomized to mitigate the effects of non-equal route traversal and obstacle-painting difficulties.

5.5 Dependent Variables

5.5.1 Path-Following

Path-following performance was measured by traverse time, traverse efficiency, and a situational awareness (SA) score calculated from the number of correctly identified targets of opportunity. Traverse time is the time from trial start to trial end, minus the amount of time spent interacting with the target of opportunity freeze probe dialogue.

$$time_{traverse} = time_{end} - time_{start} - time_{freeze\ probe}$$

Traverse efficiency is the length of the path traversed by the user divided by the optimal path length (as calculated by SEXTANT). Traverse efficiency is always greater than or equal to 1, and a value is 'more efficient' the closer it is to 1.

$$efficiency = \frac{length_{traverse}}{length_{optimal}}$$

The SA score is calculated as the number of correctly identified on-screen TOPs (positive identification when the TOP was actually present) minus the number of incorrectly identified on-screen TOPs (positive identification when the TOP was not present), divided by the number of actual on-screen TOPs. For example, if the user noted that 6 TOPs were visible at the time of the freeze probe, but only 5 actually were, their TOP score would be (5 – 1) / 5 = 0.8.

$$TOP\ score = \frac{correct - incorrect}{actual}$$

This method of calculating an SA score was inspired by Endsley's Situational Awareness Global Assessment Technique (SAGAT) (Endsley, 1988). If no TOPs were visible at the time of the freeze probe, the awarded TOP score would either be a 1 (for correctly marking 0) or 0 (for marking any number other than 0). The minimum allowable TOP score is 0 (in case of negative numbers). The quality of a path visualization was measured by the workload produced by that visualization and through an assessment of the visualization's usability. Workload was measured using the NASA Task Load Index (TLX, appendix C.4) survey (S. G. Hart & Staveland, 1988), and usability was measured using the System Usability Scale (SUS, appendix C.5) survey (Brooke, 1996). Though TLX generally begins by asking subjects to assign weights to its six subscales (based on the believed influence each subscale will have on workload, given the task), for this study we omitted that step. Instead, we used a common variation of NASA TLX called Raw TLX (RTLX). A workload value for RTLX is calculated by averaging the six TLX subscale values (no weighting). According to Hart, various studies have shown RTLX to be “either more sensitive, less sensitive, or equally sensitive” to regular TLX (S. G. Hart, 2006). We elected to use RTLX since it takes less time to complete than regular TLX and appears to be no less valid.

5.5.2 Obstacle-Editing

Obstacle-editing performance was measured by traverse time, time spent using the editing tool, traverse efficiency, and the accuracy with which users painted unmarked obstacles. Traverse time is the time from trial start to trial end, minus the amount of time the obstacle editing tool was active.

$$time_{traverse} = time_{end} - time_{start} - time_{tool}$$

Traverse efficiency was calculated in the same way as it was in the path-following experiment.

$$efficiency = \frac{length_{traverse}}{length_{optimal}}$$

For the obstacle-editing experiment, the optimal length used in the denominator of the efficiency calculation is taken from the optimal path that results when all unmarked obstacles are assumed to be marked as actual obstacles. It is expected that efficiency values from the obstacle-editing experiment will be higher (less efficient) than those of the path-following experiment because – up until the user paints the region of unmarked obstacles – the path displayed to the user will be non-optimal (and also incorrect – it will be passing through the region of unmarked obstacles). Obstacle painting accuracy is calculated as the number of correctly painted grid cells (painted grid cells that were either obstacles or unmarked obstacles) divided by the total number of painted grid cells.

$$accuracy = \frac{painted_{correct}}{painted_{total}}$$

The quality of an obstacle-editing interface was measured by the workload produced by that interface and through an assessment of the interface’s usability. Workload was measured using the NASA TLX survey (again, as RTLX), and usability was measured using the SUS survey.

5.6 Data Collected

Data from each trial was saved as a JSON file. The following elements were recorded as part of the JSON object:

- Trial Type: A string indicating whether the trial was a path-following or an obstacle-editing trial
- Subtrial: An integer indicating the number of the subtrial (0, 1, or 2)
- Scenario: A string indicating which path route was used
- Path Visualization: A string indicating which path visualization was used
- Obstacle Edit Mode: A string indication with obstacle-editing interface was used
- Traverse Time: A floating point number indicating how many seconds a subject spent traversing a path
- Idle Time: A floating point number that has a different meaning depending on the trial type. For path-following trials, it indicates how much time the user spent interacting with the target of opportunity dialogue. For obstacle-editing trials, it indicates how much time the user spent with the obstacle-editing tool activated.
- Time in Unmarked Obstacles: A floating point number indicating how much time during the trial a user was inside an unmarked obstacle.
- Time in Marked Obstacles: A floating point number indicating how much time during the trial a user was inside a marked obstacle.
- Time Not in Obstacles: A floating point number indicating how much time during the trial a user was on normal terrain.
- Counted Targets Of Opportunity: An integer indicating the number of targets of opportunity recorded by the user via the TOP freeze probe dialogue. For obstacle-editing trials, this number is always -1.
- Added Obstacles: A list of coordinates; each coordinate indicates a DEM grid cell whose final add/remove state was 'added' (i.e. the last obstacle 'add' or 'remove' action taken on that cell was 'added').
- Removed Obstacles: A list of coordinates; each coordinate indicates a DEM grid cell whose final add/remove state was 'removed' (i.e. the last obstacle 'add' or 'remove' action taken on that cell was 'removed').

- Traversed Path: A list of objects; each object contains the coordinates of a grid cell the user entered during a traverse and a corresponding timestamp for when that grid cell was entered.

In addition to this JSON object, a screenshot was taken in the path-following trials just before the target of opportunity freeze probe dialogue appeared. This screenshot was used as a "ground truth" against which the number of TOPs a subject claimed to have observed could be compared.

5.7 Subject Instructions

The first part of this study seeks to evaluate the effectiveness of several different interfaces for navigating SEXTANT-created paths. A path-following interface's 'effectiveness' is determined through consideration of path traversal time, completion of secondary task goals, and the results from a set of workload and interface satisfaction assessments.

The second portion of the study seeks to determine which among several terrain modification interfaces is the most effective. 'Effectiveness' here is determined through consideration of the time required to make terrain modifications, the precision with which those modifications are made, and the results from a set of workload and interface satisfaction assessments.

All interfaces were built using the Unity game engine and shared many of the same elements and features; the only difference between them was the way in which paths were visualized and terrain was modified.

Prior to arriving at the test location, subjects were given an informed consent form to review, a demographic information survey (appendix C.1) to complete, and a COVID-19 safety protocol (appendix C.2) to read. At the designated time and day of the test, participants were asked to arrive at the 7th floor of Draper's Hill building.

Before any subjects were tested, a researcher went to the testing location and set up all necessary testing equipment. The device was plugged in and placed on a table, directional signs were created and displayed at key locations in the Hill building, sanitization wipes were placed in an easily accessible location, and informed consent forms were placed in individually-labeled subject folders. The informed consent forms were placed in separate folders in order to reduce the number of shared objects subjects would be asked to touch. Setup was handled in such a way that no on-site researcher would be required to administer the test (researchers remained at a remote location during testing to reduce the risk of COVID-19 transmission).

At the testing location, subjects found their individually-labeled informed consent form and the testing device itself (a tablet equipped with a position tracking camera). Just after arriving at the testing location, subjects used a personal device to join a video call created by a remote researcher. Once the video call was established, subjects were asked to read and sign their informed consent form and to then place the signed form on the stack of other signed forms. Next, subjects were instructed to turn on the testing device and to login to the 'Subject' user account. On the tablet's desktop were two icons – one for running the SEXTANT (backend) server, and one for running the *Pathfinder* (frontend) application. Subjects were asked to first start the SEXTANT server and, once it was ready, to run *Pathfinder*. Once *Pathfinder* loaded, subjects – now ready to begin their testing – were directed through the following series of tasks:

1. Path-Following Tutorial: Subjects were first presented with a tutorial designed to prepare them for the path-following trial. The tutorial covered basic movement controls, explained the various visualizations used for path-following, and introduced the 'targets of opportunity' (TOPs) concept.

2. Path-Following Trials: Subjects were next asked to complete a set of three path-following trials (one for each path visualization). Trials differed from one another only by which path visualization was used; the actions required of a subject in each trial were identical. During each trial, subjects were given the following set of priority-ordered tasks: avoid all impassable regions, follow the path closely, reach the endpoint quickly, and keep a mental count of the number of currently visible targets of opportunity. Timestamped coordinates of a subject's traversed path were recorded automatically by the application. In addition, once per sub-trial, a subject's traverse was interrupted by a freeze probe; a dialogue opened and occluded the screen, traverse time was paused, and subjects were asked how many TOPs were currently in view. The application recorded the result and un-paused the traverse. After completing a trial, subjects were asked to fill out a NASA TLX survey and a SUS survey.

3. Obstacle-Editing Tutorial: After completing the path-following trials, subjects were instructed to run through a second tutorial. This tutorial was designed to teach subjects how to edit obstacles (i.e. how to designate regions of virtual terrain as impassable), and when it is appropriate to do so.

4. Obstacle-Editing Trials: Lastly, subjects were asked to complete a set of three obstacle-editing trials (one trial for each obstacle-editing interface, plus one for manual re-routing). As in the path-following trials, each of these trial asked subjects to move from a designated start to a designated end location. Trials differed from one another only by the obstacle-editing interface elements displayed; the actions required in each trial were identical. Additionally, each trial used the same path-following interface: the line visualization. During each traverse, subjects encountered a region where the designated path passed through a region of unmarked obstacles. Subjects were instructed to use the interface to inform the application of the impassable region and to then find a new route to the end location. During each trial, subjects were given the following set of priority-ordered tasks: avoid all impassable regions, reach the endpoint quickly, paint the unmarked obstacles quickly, and paint the unmarked obstacles accurately. For one of the trials, subjects were given no terrain modification tools and instead were asked to manually (i.e. without interface guidance) re-route to the end location. Timestamped coordinates of the traversed path and the coordinates of all modified obstacles were recorded automatically by the application. After completing a trial, subjects were asked to fill out a NASA TLX survey and a SUS survey.

After a subject completed all of the trials, they were asked to fill out a post-test survey (appendix C.3).

Because of the ongoing COVID-19 pandemic, there were several pre-test and post-test sanitization steps subjects were asked to perform. At the beginning and end of a test session, subjects were asked to spray and wipe down all shared equipment (tablet, laptop, pen, etc.) with a sanitization wipe. In addition, subjects were asked to wipe down the surfaces on which the shared equipment resided and any door handles that needed to be operated in order to arrive at the testing location.

Each Position, timing, and secondary goal result datum collected from each of the trials was stored on the hard drive of the tablet. A unique identifier (UID) was automatically created for each subject, and all of a subject's trial data was associated with that UID. All surveys were hosted by and filled out using Qualtrics; a subject's survey answers were only accessible by members of the research group. Identifying information (e.g. name) was never associated with any of the collected data.

5.8 Participants

There were a total of six participants for this study. A larger number of participants would have been useful in order to better evaluate the significance of the results, but recruitment proved to be difficult with so many people working from home due to the COVID-19 pandemic. All subjects were Draper employees. All subjects were either male or female and ranged between 26 and 41 years of age. Subjects came from various different professional roles (e.g. designer, engineer, etc.).

6. Results

6.1 Path-Following

Analyses were conducted on traverse time, traverse efficiency, workload (as calculated by NASA TLX), and usability (as calculated by SUS). No situational awareness analysis was performed; the reasons for this are discussed in section 6.1.3. An alpha level of 0.05 was used for all statistical tests. Repeated measure analysis of variance (ANOVA) tests were applied to the results to determine the significance level of any perceived differences in mean in the levels of a factor. If the results of an ANOVA test determined that a statistically significant difference in means did exist, a post hoc paired t-Test was performed on all combinations of factor levels to determine which levels differed from the others. Six different subjects (n = 6) participated in the path-following trials. A summary of all the data collected during the path-following trials is given in Table 6.1.

Table 6.1: Summary of all data collected during the path-following trials. Mean values for traverse time, traverse efficiency, RTLX score, and SUS usability score are given, broken down by the different levels of the path visualization, path route, and trial order factors.

	Path Visualization			*Path Route*			*Trial Order*		
	Line	Equidistant	Clear Path	Route 1	Route 2	Route 3	Trial 1	Trial 2	Trial 3
Traverse Time	89.5s	96.9s	92.7s	111.1s	82.6s	85.4s	114.2s	90.9s	74.0s
Traverse Efficiency	1.21	1.30	1.27	1.27	1.25	1.26	1.28	1.26	1.25
Workload (RTLX)	45.5	40.0	35.3	45.5	37.4	41.0	37.9	40.6	45.4
Usability (SUS)	73.3	72.5	79.6	70.8	73.3	81.25	81.2	72.9	71.25

6.1.1 Traverse Time

Traverse time for each subject was computed by taking the difference between a subject's start time (time of first movement during a trial) and end time (time when the user reached the endpoint of a traverse) and subtracting out the total time the subject spent interacting with the freeze probe TOP dialogue.

$$time_{traverse} = time_{end} - time_{start} - time_{freeze\ probe}$$

Three one-factor, repeated measure ANOVA tests were performed to determine if traversal times differed significantly with respect to the different levels the path visualization, path route, and trial order factors. No significant difference in mean traverse time was found to exist among the levels of any of the factors. Results of the ANOVA tests are given in Table 6.2.

Table 6.2: Results from three one-factor, repeated measure ANOVA tests; the tests analyzed traverse time differences among the levels of the path visualization, path route, and trial order factors.

	Path Visualization	*Path Route*	*Trial Order*
F	0.067	1.52	3.15
p-value	0.94	0.26	0.087

The mean traverse times for subjects using the line visualization, equidistant waypoint visualization, and clear-path waypoint visualization are 89.5s, 96.9s, and 92.7s, respectively; these results are shown in Figure 6.1. The mean traverse times for subjects traversing path route 1, path route 2, and path route 3 are 111.1s, 82.6s, and 85.4s, respectively; these results are shown in Figure 6.2. The mean traverse times for subjects during the 1st, 2nd, and 3rd trials are 114.2s, 90.9s, and 74.0s, respectively; these results are shown in Figure 6.3.

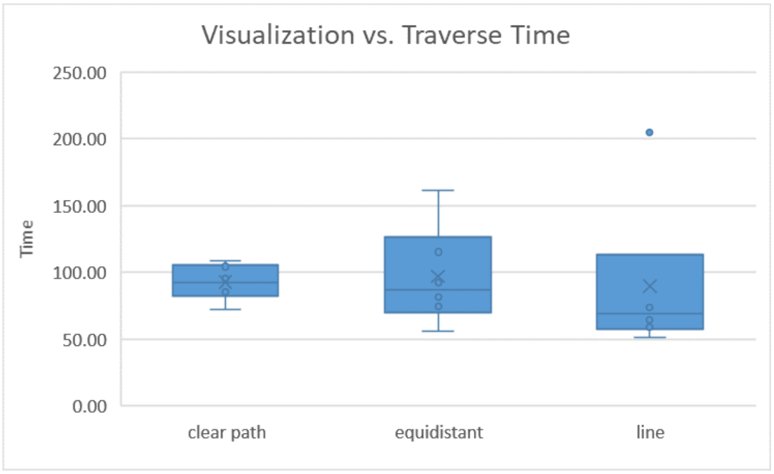


Figure 6.1: Box-plot of path visualization vs. traverse time

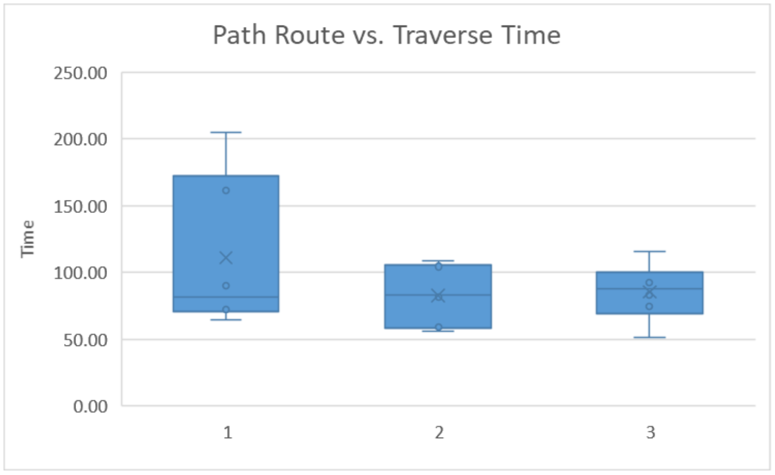


Figure 6.2: Box-plot of path route vs. traverse time

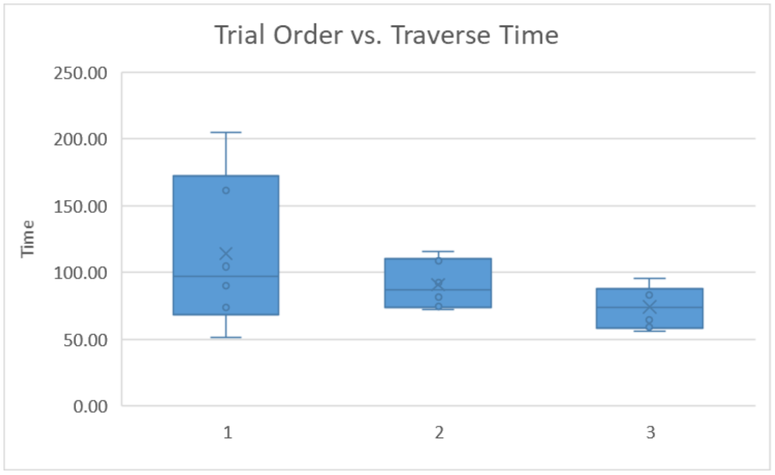


Figure 6.3: Box-plot of trial order vs. traverse time

6.1.2 Traverse Efficiency

Traverse efficiency for each subject was computed by dividing the actual distance traveled by a subject during a traverse by the length of the optimal path for that traverse.

$$efficiency = \frac{length_{traverse}}{length_{optimal}}$$

Three repeated measure ANOVA tests were performed to determine if traverse efficiency differed significantly among the levels of the path visualization, path route, and trial order factors. The results from these tests are shown in Table 6.3.

Table 6.3: Results from three one-factor, repeated measure ANOVA tests; the tests analyzed traverse efficiency differences among the levels of the path visualization, path route, and trial order factors.

	Path Visualization	*Path Route*	*Trial Order*
F	3.79	0.13	0.24
p-value	0.059	0.88	0.79

No significant difference was found to exist among the different levels of the path route and trial order factors. Though the p-value from path visualization ANOVA test was larger than 0.05, it was small enough to warrant further investigation. Paired t-Tests were performed to compare the efficiencies of the line visualization vs. the equidistant waypoint visualization, the line visualization vs. the clear path waypoint visualization, and the equidistant waypoint visualization vs. the clear path waypoint visualization. The results from these paired t-Tests revealed a statistically significant difference between the traverse efficiencies of the line visualization and the equidistant waypoint visualization ($p = .0091$); the t-Tests revealed no other significant results. The mean traverse efficiencies for subjects using the line visualization, equidistant waypoint visualization, and clear-path waypoint visualization are 1.21, 1.30, and 1.27, respectively; these results are shown in Figure 6.4. The mean traverse efficiencies for subjects traversing path route 1, path route 2, and path route 3 are 1.27, 1.25, and 1.26, respectively; these results are shown in Figure 6.5. The mean traverse efficiencies for subjects during the 1st, 2nd, and 3rd trials are 1.28, 1.26, and 1.25, respectively; these results are shown in Figure 6.6.

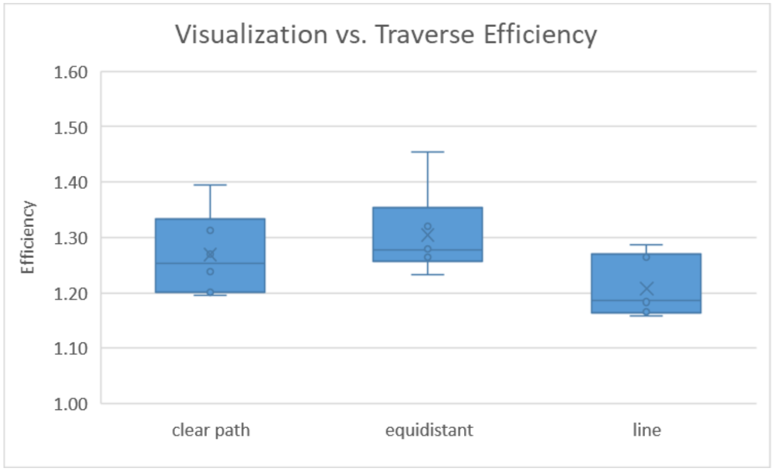


Figure 6.4: Box-plot of path visualization vs. traverse efficiency

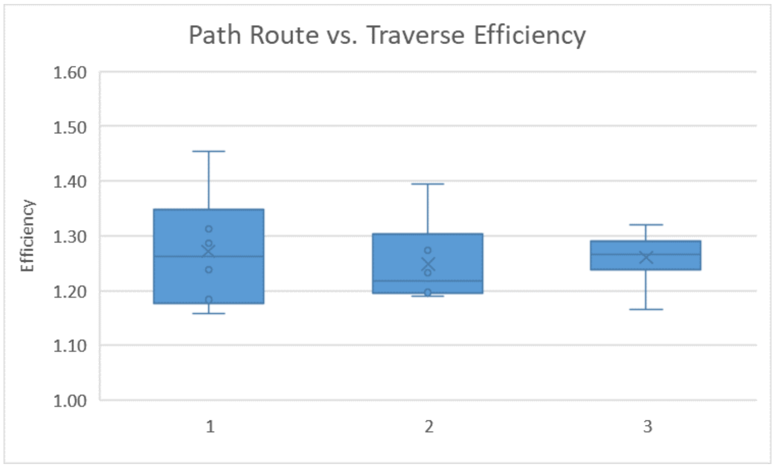


Figure 6.5: Box-plot of path route vs. traverse efficiency

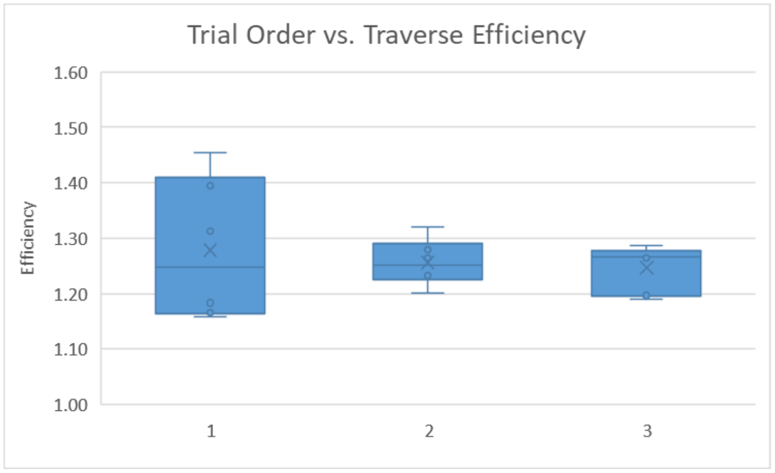


Figure 6.6: Box-plot of trial order vs. traverse efficiency

6.1.3 Situational Awareness

We originally intended to use the results from the targets of opportunity freeze probe questions to perform an evaluation of situational awareness. We elected ultimately to forgo such an analysis for several reasons, the primary of which is that our source of "ground truth" (the screenshots taken just before the freeze probe dialogue appeared) was faulty.

Pathfinder is intended to be run as a full-screen, landscaped application on the Surface Go tablet. The aspect ratio of the Surface Go is 3:2, meaning that the target aspect ratio of *Pathfinder* is also 3:2. Prior to the onset of subject testing, we failed to put the tablet into 'lock rotation' mode. When the first subject was loading the *Pathfinder* application, they tilted the Surface Go enough to cause it to switch from landscape to portrait mode. This caused *Pathfinder* to display in a 1:1

aspect ratio window, even after the tablet was returned to a landscape orientation. Though restarting *Pathfinder* fixed the aspect ratio issue from the subject's perspective, all screenshots taken during that initial and all future testing periods were in this 1:1 aspect ratio. The original purpose of the screenshots was to act as "ground truth" of the number of actual on-screen TOPs for the freeze probe. However, because the screenshots did not accurately represent what subjects actually saw (1/3 of the viewing area had been cut-off along the sides of the screenshot), they could not be used for this purpose. Example screenshots are shown in Figure 6.7.

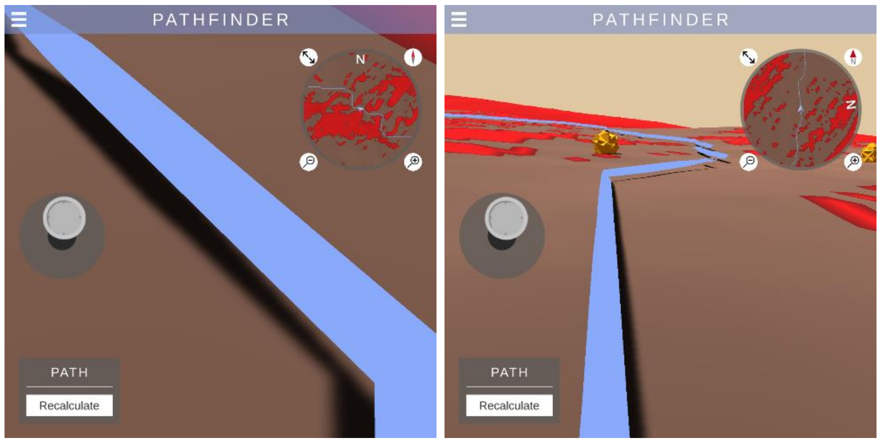


Figure 6.7: Screenshots taken during a path-following trial. Note the 1:1 aspect ratio and the large regions of the viewing area that have been 'cut-off' along the sides of the images.

Additionally, even had the screenshots been recorded at the proper aspect ratio, it is unlikely that TOP score would have offered significant insight into a subject's current level of situational awareness. Many subjects commented – and many screenshots revealed – that they would often look almost directly downward when traversing through obstacle dense regions. Subjects noted that adopting this down-looking technique helped them better understand their location with respect to nearby obstacle boundaries. When looking straight down, a subject is unable to see much of the virtual environment (including the TOPs). This would likely negatively affect environmental situational awareness, but – because accurately recording 'zero TOPs visible' during the freeze probe will grant the maximum possible value – may have had the opposite effect on TOP score.

6.1.4 Workload

Workload was evaluated for each path-following visualization by having subjects fill out a NASA TLX survey after completing a traverse. The calculated Raw TLX (RTLX) score – which is the unweighted average of the six TLX subscores – was used as the single quantitative measure of workload.

Three repeated measure ANOVA tests were performed to determine if workload differed significantly among the levels of the path visualization, path route, and trial order factors. No significant difference in mean RTLX score was found to exist among the levels of any of the factors. The results from these tests are shown in Table 6.11.

Table 6.4: Results from three one-factor, repeated measure ANOVA tests; the tests analyzed RTLX score differences among the levels of the path visualization, path route, the trial order factors.

	Path Visualization	*Path Route*	*Trial Order*
F	2.53	0.71	0.59
p-value	0.13	0.52	0.57

The mean RTLX scores for subjects using the line visualization, equidistant waypoint visualization, and clear-path waypoint visualization are 48.6, 40.0, and 35.3, respectively; these results are shown in Figure 6.8. The mean RTLX scores for subjects traversing path route 1, path route 2, and path route 3 are 45.5, 37.4, and 41.0, respectively. The mean RTLX scores for subjects during the 1st, 2nd, and 3rd trials are 37.9, 40.6, and 45.4, respectively.

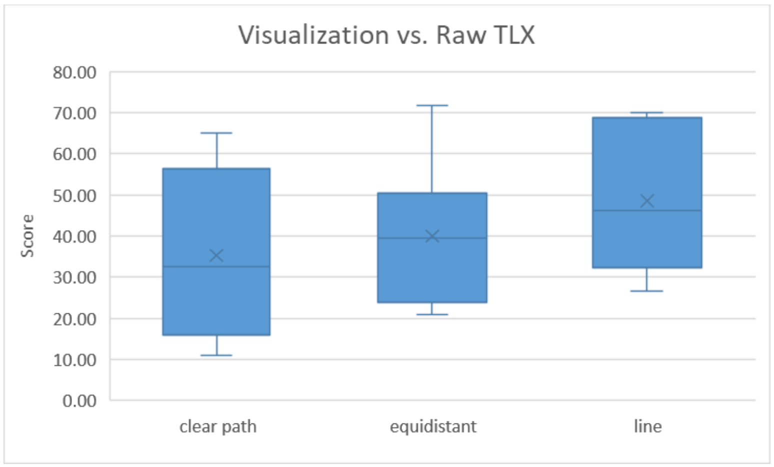


Figure 6.8: Box-plot of path visualization vs. RTLX score

Distribution data for all six of the TLX subscores – grouped by path visualization – is shown in Figure 6.9.

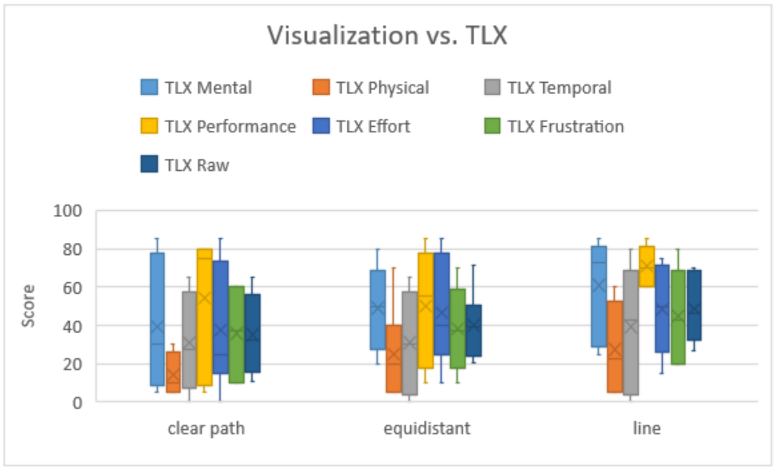


Figure 6.9: Box-plot of path visualization vs. TLX subscores

6.1.5 Usability

Usability of each path-following interface was evaluated using the system usability scale (SUS). The SUS presents 10 usability-related statements to the user and asks them to state their level of agreement (on a five-point scale) with each of those statements. A subject's responses are assigned numerical values and combined (using a SUS-specific algorithm) to form a single, numerical usability score.

Three repeated measure ANOVA tests were performed to determine if usability differed significantly among the levels of the path visualization, path route, and trial order factors. No significant difference in mean usability score was found to exist among the levels of any of the factors. The results from these tests are shown in Table 6.5.

Table 6.5: Results from three one-factor, repeated measure ANOVA tests; the tests analyzed SUS score differences among the levels of the path visualization, path route, and trial order factors.

	Path Visualization	*Path Route*	*Trial Order*
F	0.429	0.92	0.89
p-value	0.66	0.43	0.44

The mean SUS scores for subjects using the line visualization, equidistant waypoint visualization, and clear-path waypoint visualization are 73.3, 72.5, and 79.6, respectively; these results are shown in Figure 6.10. The mean SUS scores for subjects traversing path route 1, path route 2, and path route 3 are 70.8, 73.3, and 81.25, respectively. The mean SUS scores for subjects during the 1st, 2nd, and 3rd trials are 81.2, 72.9, and 71.25, respectively.

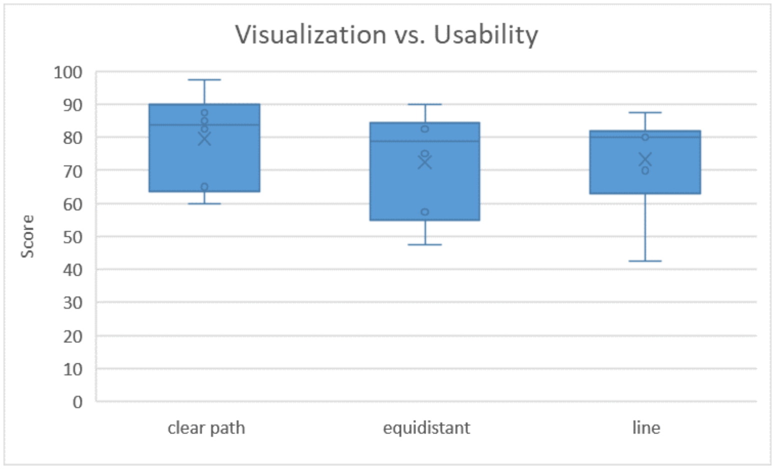


Figure 6.10: Box-plot of path visualization vs. SUS usability score

6.2 Obstacle-Editing

Analyses were conducted on traverse time, obstacle-editing time, traverse efficiency, obstacle-painting accuracy, workload (as calculated by NASA TLX), and usability (as calculated by SUS). An alpha level of 0.05 was used for all statistical tests. Repeated measure analysis of variance (ANOVA) tests were applied to the results to determine the significance level of any perceived differences in mean. If the results of an ANOVA test determined that a statistically significant difference in means did exist, a post hoc paired t-Test was performed on all combinations of factor levels to determine which levels differed from the others. Even though six subjects completed the obstacle-editing trials, the sample size used for analysis is only five (n = 5). The results from one subject were discarded because in each trial they walked directly through the region of unmarked obstacles. Traversing such paths was expressly forbidden by the trial instructions because by doing so subjects were essentially taking a ‘short cut’ to the goal location. As is the case in real-world timed locomotion challenges, short-cutting during a trial allowed subjects to achieve lower than normal traverse times and super-optimal (less than 1) traverse efficiencies. A summary of all the data collected during the path-following trials is given in Table 6.6.

Table 6.6: Summary of all data collected during the obstacle-editing trials. Mean values for traverse time, obstacle-editing time, traverse efficiency, obstacle-painting accuracy, RTLX score, and SUS usability score are given, broken down by the different levels of the obstacle-editing interface, path route, and trial order factors.

	Obstacle-Editing Interface			*Path Route*			*Trial Order*		
	Look-At	Map	None	Route 1	Route 2	Route 3	Trial 1	Trial 2	Trial 3
Traverse Time	76.8s	79.1s	94.4s	80.5s	84.9s	84.9s	86.5s	86.4s	77.4s
Editing Time	36.0s	32.5s	-	15.7s	23.2s	29.6s	20.4s	20.1s	28.0s
Traverse Efficiency	1.27	1.28	1.36	1.22	1.30	1.23	1.28	1.35	1.28
Painting Accuracy	0.84	0.77	-	0.88	0.83	0.89	0.85	0.89	0.86
Workload (RTLX)	52.5	44.5	47.3	48.2	48.0	48.2	46.2	45.7	52.5
Usability (SUS)	60.0	74.0	59.5	67.0	61.5	65.0	72.5	60.0	61.0

6.2.1 Traverse Time

Traverse time for each subject was computed by taking the difference between a subject's start time (time of first movement during a trial) and end time (time when the user reached the endpoint of a traverse) and subtracting out the total time that the obstacle-editing tool was active. The obstacle-editing tool was considered active if the brush was in either the obstacle adding (i.e. painting) state or the obstacle removing (i.e. erasing) state.

$$time_{traverse} = time_{end} - time_{start} - time_{tool}$$

Three one-factor, repeated measure ANOVA tests were performed to determine if traversal times differed significantly with respect to the different levels of the obstacle-editing interface, path route, and trial order factors. No significant difference in mean traverse time was found to exist among the levels of any of the factors. Results of the ANOVA tests are given in

Table 6.7.

Table 6.7: Results from three one-factor, repeated measure ANOVA tests; the tests analyzed traverse time differences among the levels of the obstacle-editing interface, path route, and trial order factors.

	Obstacle-Editing Interface	*Path Route*	*Trial Order*
F	3.32	0.13	0.62
p-value	0.089	0.88	0.56

The mean traverse times for subjects using the look-at mode of obstacle-editing, the map mode of obstacle-editing, and no obstacle-editing mode at all are 76.8s, 79.1s, and 94.4s, respectively; these results are shown in Figure 6.11. The mean traverse times for subjects traversing path route 1, path route 2, and path route 3 are 80.5s, 84.9s, and 84.9s, respectively. The mean traverse times for subjects during the 1st, 2nd, and 3rd trials are 86.5s, 86.4s, and 77.4s, respectively.

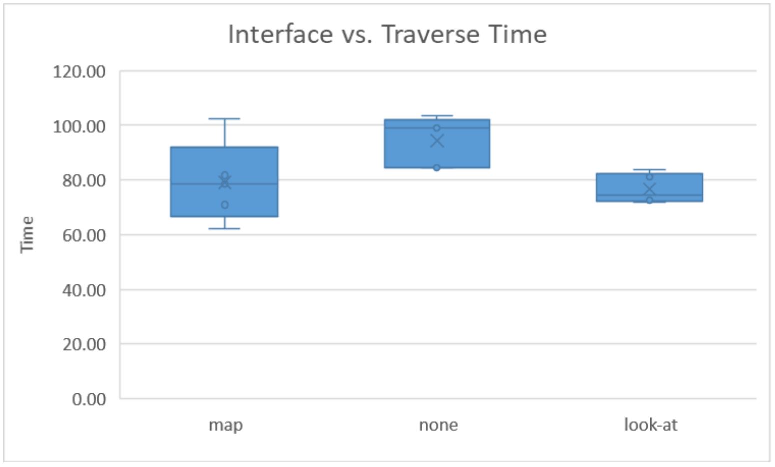


Figure 6.11: Box-plot of obstacle-editing interface vs. traverse time

6.2.2 Obstacle-Editing Time

Obstacle-editing time is the amount of time that the obstacle-editing tool was active during a traverse. It is calculated as a running sum that accumulates once during each of *Pathfinder's* update cycles. Once every update cycle (i.e. every ~1/60th of a second), *Pathfinder* checks if the obstacle-editing tool is active. If it is, the elapsed time since the last update is added to a 'tool active time' value.

Three repeated measure ANOVA tests were performed to determine if obstacle-editing time differed significantly among the levels of the obstacle-editing interface, path route, and trial order factors. No significant difference in mean obstacle-editing time was found to exist among the levels of any of the factors. The results from these tests are shown in Table 6.8.

Table 6.8: Results from three one-factor, repeated measure ANOVA tests; the tests analyzed obstacle-editing time differences among the levels of the obstacle-editing interface, path route, and trial order factors.

	Obstacle-Editing Interface	*Path Route*	*Trial Order*
F	0.19	0.36	0.14
p-value	0.69	0.71	0.87

The mean obstacle-editing times for subjects using the look-at mode of obstacle-editing and the map mode of obstacle-editing are 36.0s and 32.5s, respectively (obstacle-editing time for ‘none’ is always 0 seconds); these results are shown in Figure 6.12. The mean obstacle-editing times for subjects traversing path route 1, path route 2, and path route 3 are 15.7s, 23.2s, and 29.6s, respectively. The mean obstacle-editing times for subjects during the 1st, 2nd, and 3rd trials are 20.4s, 20.1s, and 28.0s, respectively.

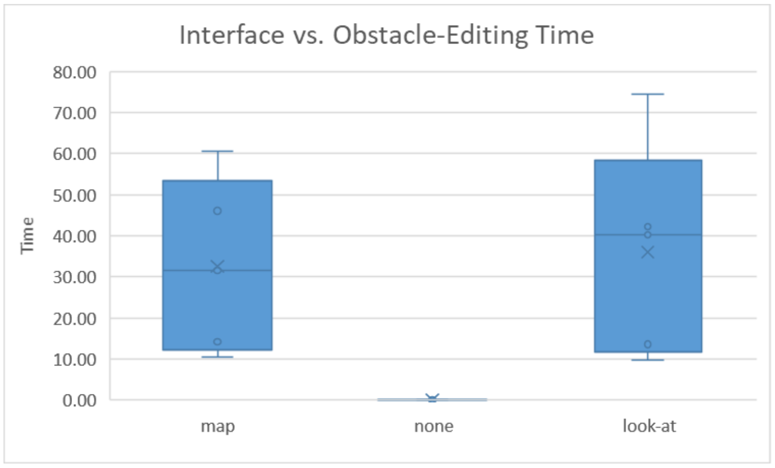


Figure 6.12: Box-plot of obstacle-editing interface vs. obstacle-editing time

6.2.3 Traverse Efficiency

Traverse efficiency for each subject was computed by dividing the actual distance traveled by a subject during a traverse by the length of the optimal path for that traverse.

$$efficiency = \frac{length_{traverse}}{length_{optimal}}$$

Three repeated measure ANOVA tests were performed to determine if traverse efficiency differed significantly among the levels of the obstacle-editing interface, path route, and trial order factors. No significant difference in mean traverse efficiency was found to exist among the levels of any of the factors. The results from these tests are shown in Table 6.9.

Table 6.9: Results from three one-factor, repeated measure ANOVA tests; the tests analyzed traverse efficiency differences among the levels of the obstacle-editing interface, path route, and trial order factors.

	Obstacle-Editing Interface	*Path Route*	*Trial Order*
F	1.96	2.77	1.12
p-value	0.20	0.12	0.37

The mean traverse efficiencies for subjects using the look-at mode of obstacle-editing, the map mode of obstacle-editing, and no obstacle-editing mode at all are 1.27, 1.28, and 1.36, respectively; these results are shown in Figure 6.13. The mean traverse efficiencies for subjects traversing path route 1, path route 2, and path route 3 are 1.22, 1.30, and 1.23, respectively. The mean traverse efficiencies for subjects during the 1st, 2nd, and 3rd trials are 1.28, 1.35, and 1.28, respectively.

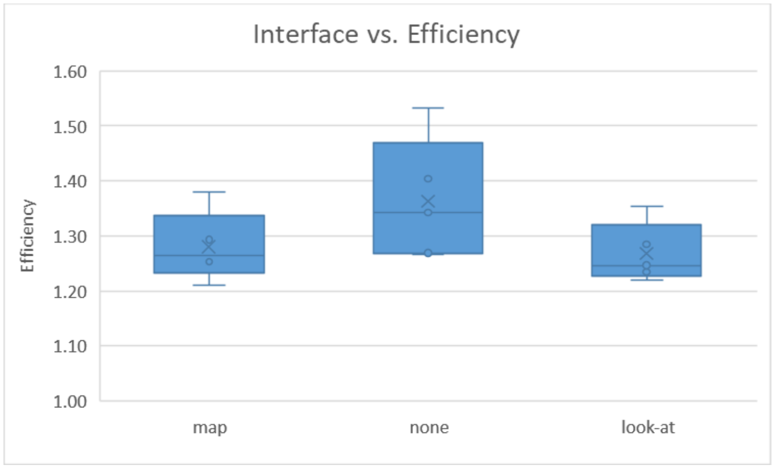


Figure 6.13: Box-plot of obstacle-editing interface vs. traverse efficiency

6.2.4 Obstacle-Painting Accuracy

Obstacle painting accuracy was measured by dividing the number of correctly painted grid cells (i.e. grid cells corresponding to either marked or unmarked obstacles whose final painted state is ‘added’) by the total number of painted grid cells (i.e. grid cells of any type whose final painted state is ‘added’).

$$accuracy = \frac{painted_{correct}}{painted_{total}}$$

Three repeated measure ANOVA tests were performed to determine if obstacle-painting accuracy differed significantly among the levels of the obstacle-editing interface, path route, and trial order factors. No significant difference in mean usability score was found to exist among the levels of any of the factors. The results from these tests are shown in Table 6.10.

Table 6.10: Results from three one-factor, repeated measure ANOVA tests; the tests analyzed obstacle-painting accuracy differences among the levels of the obstacle-editing interface, path route, and trial order factors.

	Obstacle-Editing Interface	*Path Route*	*Trial Order*
F	1.26	0.16	0.081
p-value	0.32	0.85	0.92

The mean obstacle-painting accuracy for subjects using the look-at mode of obstacle-editing and the map mode of obstacle-editing are 0.84 and 0.77, respectively (no accuracy score for ‘none’ was recorded); these results are shown in Figure 6.14. The mean obstacle-painting accuracy for subjects traversing path route 1, path route 2, and path route 3 are 0.88, 0.83, and 0.89, respectively. The mean obstacle-painting accuracy for subjects during the 1st, 2nd, and 3rd trials are 0.85, 0.89, and 0.86, respectively.

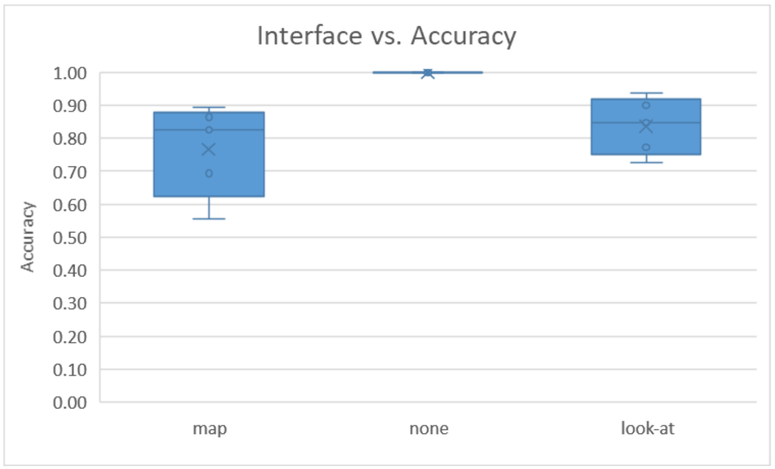


Figure 6.14: Box plot of obstacle-editing interface vs. obstacle-painting accuracy

6.2.5 Workload

Workload was evaluated for each obstacle-editing interface by having subjects fill out a NASA TLX survey after completing a traverse. The calculated RTLX score – which is the unweighted average of the six TLX subscores – was used as the single quantitative measure of workload.

Three repeated measure ANOVA tests were performed to determine if workload differed significantly among the levels of the obstacle-editing interface, path route, and trial order factors. No significant difference in mean RTLX score was found to exist among the levels of any of the factors. The results from these tests are shown in Table 6.11.

Table 6.11: Results from three one-factor, repeated measure ANOVA tests; the tests analyzed RTLX score differences among the levels of the obstacle-editing interface, path route, and trial order factors.

	Obstacle-Editing Interface	*Path Route*	*Trial Order*
F	2.10	0.00077	1.74
p-value	0.18	0.99	0.23

The mean RTLX scores for subjects using the look-at mode of obstacle-editing, the map mode of obstacle-editing, and no obstacle-editing mode at all are 52.5, 44.5, and 47.3, respectively; these results are shown in Figure 6.15. The mean RTLX scores for subjects traversing path route 1, path route 2, and path route 3 are 48.2, 48.0, and 48.2, respectively. The mean RTLX scores for subjects during the 1st, 2nd, and 3rd trials are 46.2, 45.7, and 52.5, respectively.

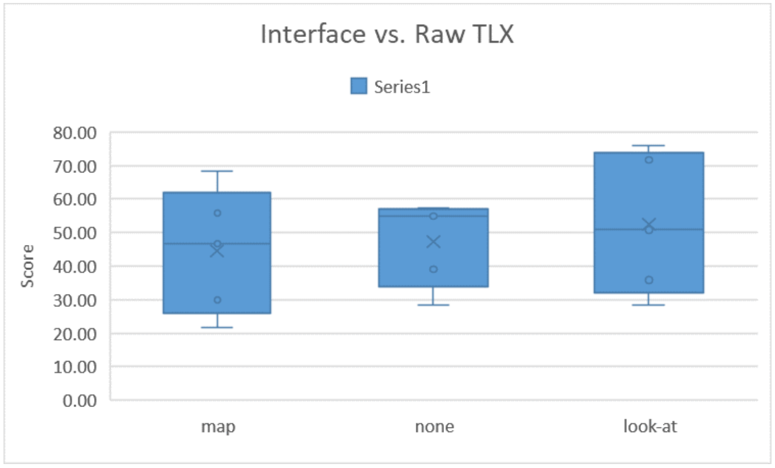


Figure 6.15: Box-plot of obstacle-editing interface vs. RTLX score

Distribution data for all six of the TLX subscores – grouped by obstacle-editing interface – is shown in Figure 6.16.

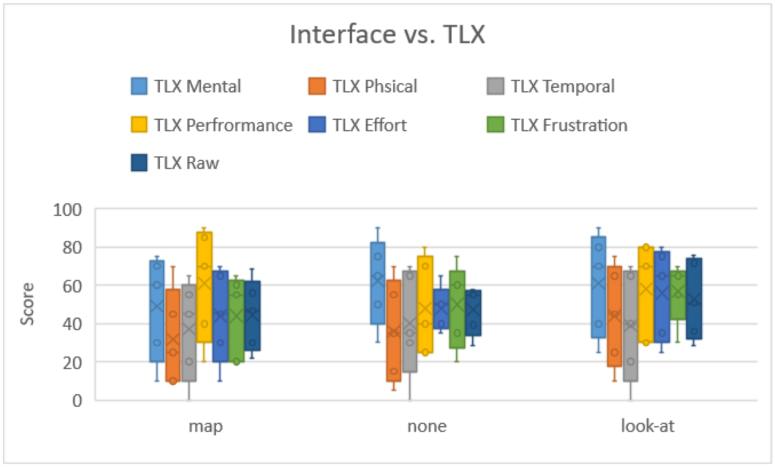


Figure 6.16: Box-plot of obstacle-editing interface vs. TLX subscores

6.2.6 Usability

Usability of each obstacle-editing interface was evaluated using the system usability scale (SUS). The SUS presents 10 usability-related statements to the user and asks them to state their level of agreement (on a five-point scale) with each of those statements. A subject's responses are assigned numerical values and combined (using a SUS-specific algorithm) to form a single, numerical usability score.

Three repeated measure ANOVA tests were performed to determine if usability differed significantly among the levels of the obstacle-editing interface, path route, and trial order factors. No significant difference in mean usability score was found to exist among the levels of any of the factors. The results from these tests are shown in Table 6.12.

Table 6.12: Results from three one-factor, repeated measure ANOVA tests; the tests analyzed SUS score differences among the levels of the obstacle-editing interface, path route, and trial order factors.

	Obstacle-Editing Interface	*Path Route*	*Trial Order*
F	2.92	0.20	1.72
p-value	0.11	0.82	0.24

The mean SUS scores for subjects using the look-at mode of obstacle-editing, the map mode of obstacle-editing, and no obstacle-editing mode at all are 60.0, 74.0, and 59.5, respectively; these results are shown in Figure 6.17. The mean SUS scores for subjects traversing path route 1, path route 2, and path route 3 are 67.0, 61.5, and 65.0, respectively. The mean SUS scores for subjects during the 1st, 2nd, and 3rd trials are 72.5, 60.0, and 61.0, respectively.

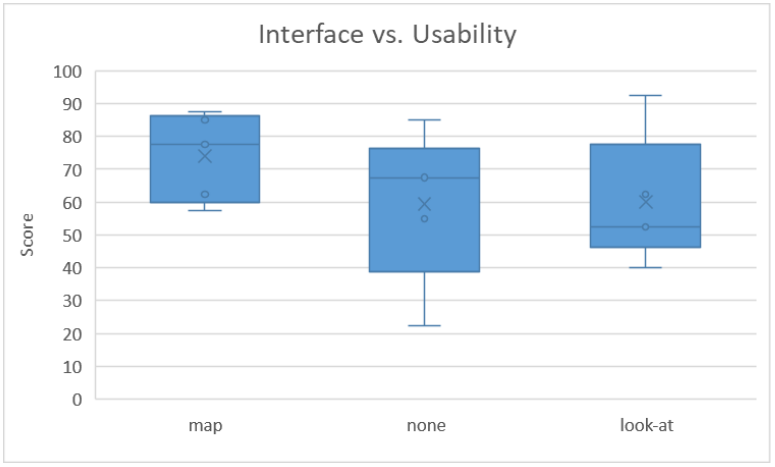


Figure 6.17: Box plot of obstacle-editing interface vs. SUS usability score

7. Discussion

The primary drawback of this study is its small sample size. Had *Pathfinder's* features been evaluated on more than six subjects, it is possible that more of the results outlined in section 6 would have statistical significance. As it stands, only a single significant result was revealed through analysis of subject data: that a user's traverse efficiency when using the line visualization is different than (in this case, less than) that when using the equidistant waypoints visualization. This result aligns with the stated hypothesis, which predicted that users of the line visualization would adhere most closely to the optimal path. Unlike the two waypoint visualizations, the line visualization presents users with a complete representation of the path; all information contained within a path is displayed by the line. This surplus of information would perhaps be disingenuous in a real-world setting (where precursor data is sometimes flawed or overly sparse); however, it is ideal for use in *Pathfinder's* perfectly constructed virtual world. This is because, at any point along a traverse, the line visualization allows users to gauge the distance separating them from the optimal path. That is, it allows users to constantly be aware of their current path-following error and to make needed adjustments based on that information.

Although the traverse efficiency metric is here taken to be a measure of how closely a user followed the designated path, a 'good' (close to 1) traverse efficiency could be achieved by traversing some other path whose shape is very different than – but close in length to – the optimal path. Considering the eight-directional movement constraints of the A* algorithm, such a path is fairly easy to construct. An example is given in Figure 7.1.

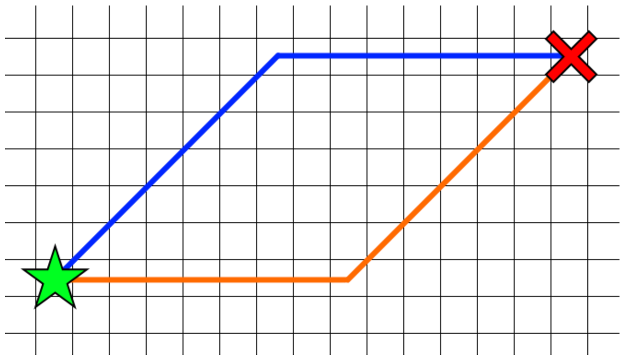


Figure 7.1: Example of two A* optimal length paths (orange and blue) that have different shapes

The blue and the orange paths are the same length and are both optimal (according to A*), but have very different shapes. Future studies should consider employing metrics that compare paths based on their shape, such as the Hausdorff and Fréchet distances (Seyler, Kumar, Thorpe, & Beckstein, 2015).

Because the difference in mean traverse efficiency values is the only statistically significant result, all further discussion points are based on user feedback and perceived trends in the data.

Figure 6.2, which compares mean traverse times among the different path routes, shows that subjects on average took more time to complete path route 1 (111s) than either path route 2 (82.6s) or path route 3 (85.4s). Equating longer traverse times with increasing traverse difficulty, this implies that the path route 1 is more difficult to traverse than either path route 2 or path route 3. Because the three different routes necessarily pass through different regions of terrain and are surrounded by obstacles of different sizes and shapes, such a difference in difficulty is likely. As path routes were being designed, there was an understanding that some difference in traverse difficulty would be present. Attempts were made to minimize this difference, though these results reveal that further path refinement was warranted. Close inspection of the geometry of the three path routes reveals a section of path route 1 that may have caused this heightened difficulty. This region is shown in the left-most panel of Figure 7.2.

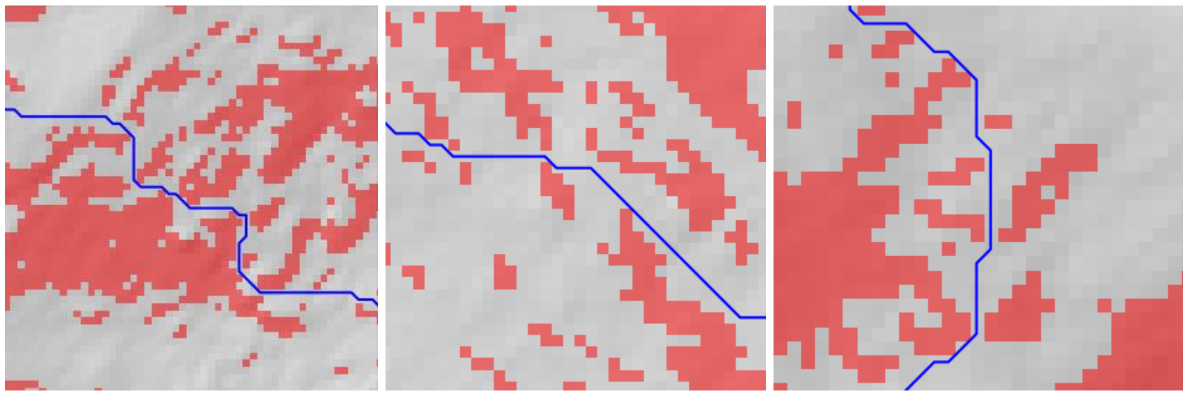


Figure 7.2: Sections of the three path routes that have users traverse through regions densely populated by obstacles. (Left) path route 1; (Middle) path route 2; (Right) path route 3.

In an attempt to keep traverse difficulty consistent between routes, all path routes were designed to have the same length and to contain the same set of basic region types. One such region type is an area containing many small obstacles and possible alternate paths. Path route 1's version of this region is larger than that of either path route 2 or path route 3, and its passable sections of terrain were also narrower. Navigating through this region in path route 1 was likely more difficult than navigating through the corresponding regions in path routes 2 and 3, elevating the comparative traverse difficulty of path route 1. In order to avoid route-based performance differences, future studies would do well to find a comprehensive and accurate way to evaluate path traverse difficulty.

As can be seen from Figure 6.3 and Figure 6.6, the means and variances of traverse time and traverse efficiency are larger in trial 1 than in either trial 2 or trial 3. In addition, the ANOVA test evaluating traverse time differences between trial orders returned a p-value of 0.087. Though larger than the 0.05 threshold, this value is indicative of a trend that is approaching significance (particularly given the small *n* of this study). When taken together, these results indicate an overall increase in traverse performance over the course of testing. Various comments made by subjects (e.g. several relating to discovering better 'movement techniques') during testing further reinforce the idea that learning may have played a significant role in traverse performance. During the development of *Pathfinder*, it was discovered that first-time users would often need to spend several minutes with the application in order to become proficient at using its movement controls. *Pathfinder's* path-following tutorial – in addition to supplying users with information on the three path visualizations – was intended to give subjects the opportunity to practice moving through virtual environments. In order to better combat the effects of learning on traverse performance, it is recommended that future studies provide subjects with more time to practice moving about the virtual world before starting the trials.

This study hypothesized that the mental workload produced by the clear path waypoint visualization would be lower than that of either the equidistant waypoint or line visualizations. The reasoning was that the clear path waypoint visualization presents users with the smallest possible set of information required to make forward progress along a path. In addition, the visualization never requires a user to make any small-scale navigation decisions; the next waypoint can always be reached by walking in a straight line. Results indicate that the mental workload hypothesis was correct. Figure 6.15 shows that the clear path waypoint visualization produced on average a lower mental workload than either the line or equidistant waypoints visualizations. the ANOVA test used to evaluate the significance of these differences produced of p-value of 0.13; though not significant, the value is small enough to indicate a trend.

Lewis and Sauro (2018) note that a SUS score of 68 is the benchmark against which an interface's usability should be measured. This value represents the 50th percentile score among 241 industrial usability studies; exceeding this value implies that the usability of the interface in question is higher than the industry average (Lewis & Sauro, 2018). Interestingly, the mean SUS scores for all three path visualizations – 73.3, 72.5, and 79.6 for the line, equidistant waypoint, and clear-path waypoint visualizations, respectively – are larger than 68. When filling out a SUS survey, users were instructed to take into account every aspect of *Pathfinder's* interface – not just those aspects unique to the currently-in-use path visualization. The uniformly high average SUS scores are perhaps indicative of a generally high usability level of the user interface elements common to all three path visualizations (e.g. the minimap, the movement control scheme, etc.). Though the clear path waypoint visualization had the highest mean usability score, the differences were not so large to indicate a convincing trend.

Unlike with the path-following trials, trial order appears to have had little effect on a subject's traverse performance (as measured by traverse time and traverse efficiency) in the obstacle-editing trials. Mean traverse times and mean traverse efficiency values remained relatively constant over the course of the three trials. This apparent lack of a learning effect may be a result of the obstacle-editing experiment being placed after the path-following experiment; by the time a subject began the obstacle-editing trials, they had accumulated 45 minutes of practice time with *Pathfinder's* movement controls.

The different levels of the obstacle-editing interface factor, however, did seem to exhibit different levels of traverse performance. As can be seen in Figure 6.11 and Figure 6.13, the mean traverse times and traverse efficiency values for subjects using no obstacle-editing interface (94.4s, 1.36) were larger than those when using either the look-at (76.8s, 1.27) or map (79.14s, 1.28) modes of editing obstacles. This trend aligns with the stated hypothesis that having access to obstacle-editing features will decrease traverse time and increase (make closer to 1) traverse efficiency. When users who had no access to obstacle-editing features encountered a region of unmarked obstacles, they were required to use some alternate means – sometimes features of *Pathfinder* that they had never before used – to manually find a route to the goal location.

Mentally calculating a path to the goal location, or determining the shortest route among several options, takes time. Additionally, subjects incur a time cost when interacting with the minimap (using maximize, zoom, and orientation features). Any amount of time not spent advancing towards the goal location will – by adding to traverse time – decrease overall traverse performance. The results here seem to indicate that having access to obstacle-editing tools improves traverse performance when compared to performing mental re-routing calculations and using tools not designed specifically for handling unmarked obstacles.

Numeric results concerning the effects different levels of the obstacle-editing interface factor had on obstacle-editing time is inconclusive. The mean obstacle-editing time of users using the look-at mode of editing obstacles (36.0s) is greater than that of those using the map mode of editing obstacles (32.5s), but the variance in those results is large. Some of the exceptionally long obstacle-editing times exhibited by subjects can likely be attributed to the way in which obstacle-editing time is calculated. Whenever the obstacle-editing tool is in its active state, obstacle-editing time is accumulating. This accumulation occurs even if the user is progressing towards the goal location and paying little attention to the obstacle editing tool (the tool itself does not restrict user movement). Notably, traverse time does *not* accumulate when the obstacle-editing tool is active. Tracking obstacle-editing time in this manner likely leads to over-estimations of the time spent actually editing obstacles and under-estimations of traverse time. In order to get a more accurate estimate of how much time it takes to perform obstacle edits, future studies would do well to find a more robust way of knowing when a user is interacting with the obstacle-editing tool. Further confounding the obstacle-editing time results are the usability problems that several subjects experienced while interacting with the obstacle-editing tool. Subjects generally seemed to have no issue with activating the tool, which was accomplished by clicking either the paint brush button or the eraser button (depending on the desired painting effect) found in the obstacle-editing panel. Deactivating the tool, however, did cause confusion among several subjects. Subjects noted that the icon used on the deactivation button – which is highlighted in Figure 7.3 – was not indicative of the function performed by that button.

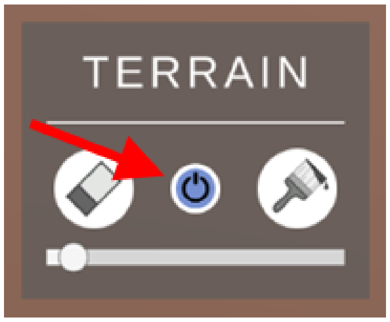


Figure 7.3: Button used to deactivate the obstacle-editing tool.

This icon is generally used on buttons that turn hardware (such as a television) on or off, as opposed to buttons that enable or disable functionality within an application. Several subjects noted that they had difficulty deactivating the obstacle-editing tool when using the map interface. When the map mode of editing obstacles is active, the minimap is forced to expand to its maximum size; in this state, the minimize/maximize minimap toggle is disabled. Some users attempted to disable obstacle-editing by using the disabled minimize toggle, which was ineffective. Users who experienced these deactivation issues spent more time with the obstacle-editing tool active than they otherwise would have, artificially inflating their obstacle-editing times.

Results concerning the effect obstacle-editing interface has on obstacle-painting accuracy slightly favor the look-at mode of obstacle editing. The mean obstacle-painting accuracy exhibited by subjects using the look-at interface is 0.84, while that of subjects using the map interface is 0.77. This goes against the hypothesis that the map mode of editing obstacles would allow users to more accurately paint obstacles. One subject mentioned that precisely painting regions of obstacles using the map tool was difficult because the region being painted was occluded by the finger doing the painting. Painting at a slight offset from the touch location (rather than directly under the finger) could improve the obstacle-painting accuracy rating of future iterations of the map mode of editing obstacles.

Though not statistically significant, results indicate that the map mode of editing obstacles produced a lower average workload (44.5) than did the look-at mode of editing obstacles (52.5) or having access to no obstacle-editing tools (47.3). The ANOVA test evaluating the statistical significance of the difference of these means generated a p-value of 0.18, which could be indicative of a trend towards significance. This result aligns with the hypothesis that the map mode of editing obstacles would produce the lowest mental workload of the three scenarios. Subjects noted that the map mode of editing obstacles felt natural and familiar, which perhaps lessened the level of mental effort required by the mode. The map interface's finger-based brush control scheme also seems to require less physical effort than controlling brush position through the arm and torso motions required by the look-at interface. Figure 6.16 confirms that on average users of the map interface expended less mental and physical effort than users of the look-at interface or users having no access to any obstacle-editing features. Lower physical and mental effort produces lower overall RTLX scores.

Additionally, the map mode of editing obstacles was found to have a higher mean usability score (74) than either the look-at mode of editing obstacles (60) or having no available obstacle-editing features. Interestingly, the map mode of editing obstacles is the only interface that exceeds the 68 point SUS median threshold. This result aligns with much of the verbal and written feedback received from subjects. Many subjects noted that painting obstacles with the map interface was 'fun' and 'intuitive'.

8. Conclusion

This work discussed changes that were made to SEXTANT to allow for real-time obstacle demarcation and path recalculation. Paramount among these changes was moving the bulk of SEXTANT's A* path-planning algorithm into a C++ external module. The creation and utilization of this external module resulted in an order of magnitude increase in the runtime performance of the algorithm. Another important modification was the pre-calculation and caching of all node-to-node traverse cost data and all node-to-goal heuristic cost data. Caching data in this way further improved runtime performance by replacing costly computations (e.g. square root) with simple array lookups. The last of the major SEXTANT changes was adding the ability to modify through user input the obstacle state of a given terrain grid location. Previously, the determination of a grid location's obstacle state (either 'passable' or 'impassable') was made at load time and was based only on a specified maximum traversable gradient.

To accommodate the hardware and software needs of this project, SEXTANT was transformed into a standalone application. This application has the option of operating as a listen server that communicates with clients over a TCP socket connection and can be run with or without a simple graphical user interface. Though the TCP socket connections could theoretically accommodate communication across a network, for this project they were used to accomplish inter-process communication between SEXTANT and the newly created *Pathfinder* application.

Built using the Unity game engine, *Pathfinder* was originally envisioned as an AR application that could be used by scientists conducting research in the field. In its initial design, *Pathfinder* would display path information and allow users to access the obstacle-editing features of SEXTANT. Because of time constraints and the COVID-19 pandemic, however, *Pathfinder* instead became a 'simulated AR' application whose primary intended use was as a tool for evaluating SEXTANT-enabling features and interface elements. This version of *Pathfinder* still displays path information and allows users to edit obstacles, but does so in an entirely virtual environment.

Three different path visualizations and two different obstacle-editing interfaces were evaluated in this work. Though the limited sample size of the study meant that few significant results were discovered, trends approaching significance did emerge. The line visualization was found to produce generally high path-following performance, while the waypoint visualizations – and particularly the clear path waypoint visualization – were found to produce lower mental workloads. Usability ratings for all three visualizations were high. Having access to obstacle-editing capabilities increased overall traverse performance, though neither interface was found to perform better than the other. The map interface was found to produce a lower mental workload than the look-at interface or when using no obstacle-editing interface at all. The map mode of editing obstacles was the only interface found to have a better than industry average usability score.

8.1 Future Work

From a SEXTANT perspective, investigating path-finding algorithms that produce more natural-looking results would be a great benefit. Recommended algorithms are Field D* or Theta*. Additionally, implementing a more sophisticated system of automatic obstacle identification would be helpful. SEXTANT's current gradient-based obstacle detection system is useful and likely should remain part of any future system, but its effectiveness could be improved by incorporating additional terrain-related inputs or by training a neural network to recognize patterns common to regions of obstacles.

For any future projects looking to build upon *Pathfinder* or any of the related AR learnings presented here, it is highly recommended that a proper AR application be constructed. Many of the design ideas incorporated into the path visualizations relied on the fact that the data used to construct any optimal path will not be a perfect representation of the real world. The potential benefits of these design concepts were likely not fully revealed by this study, however, since *Pathfinder's* virtual world did not incorporate the concept of data imperfection. To gain a real sense of how an AR application would perform in the field, some threat of tripping over a previously unknown obstacle or having unreliable data should be present.

9. References

Aaron, P., Zeller, M., & Wojciakowski, M. (2017). How inside-out tracking works. Retrieved from https://docs.microsoft.com/en-us/windows/mixed-reality/enthusiast-guide/tracking-system

Abd Algfoor, Z., Sunar, M. S., & Kolivand, H. (2015). A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games. *International Journal of Computer Games Technology, 2015*, 1-11.

Abercromby, A. F. J., Gerhnhardt, M. L., & Litaker, H. (2012). *Desert Research and Technology Studies (DRATS) 2009: A 14-Day Evaluation of the Space Exploration Vehicle Prototype in a Lunar Analog Environment*. Lyndon B. Johnson Space Center: National Aeronautics and Space Administration

Anandapadmanaban, E., Tannady, J., Norheim, J., Newman, D. J., & Hoffman, J. A. (2018). *Holo-SEXTANT: an Augmented Reality Planetary EVA Navigation Interface*. Paper presented at the 48th International Conference on Environmental Systems, Albuquerque, New Mexico.

Antoniac, P., & Pulli, P. (2001). *Marisil–mobile user interface framework for virtual enterprise*.

Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S., & MacIntyre, B. (2001). Recent advances in augmented reality. *IEEE Computer Graphics and Applications, 21*(6), 34-47.

Beaton, K. H., Chappell, S. P., Abercromby, A. F. J., Miller, M. J., Kobs Nawotniak, S. E., Brady, A. L., . . . Lim, D. S. S. (2019). Using Science-Driven Analog Research to Investigate Extravehicular Activity Science Operations Concepts and Capabilities for Human Planetary Exploration. *Astrobiology, 19*(3), 300-320.

Beaton, K. H., Chappell, S. P., Abercromby, A. F. J., Miller, M. J., Nawotniak, S. K., Hughes, S. S., . . . Lim, D. S. S. (2017, 4-11 March 2017). *Extravehicular activity operations concepts under communication latency and bandwidth constraints*. Paper presented at the 2017 IEEE Aerospace Conference.

Brooke, J. (1996). SUS: a “quick and dirty'usability. *Usability evaluation in industry*, 189.

Buchmann, V., Violich, S., Billinghurst, M., & Cockburn, A. (2004, 2004). *FingARtips: Gesture Based Direct Manipulation in Augmented Reality*, Singapore.

Carr, C. E., Newman, D. J., & Hodges, K. V. (2003). Geologic traverse planning for planetary EVA. *33rd International Conference on Environmental Systems (ICES)*(724), 2003-2416.

Connell, D., & La, H. (2017). *Dynamic path planning and replanning for mobile robots using RRT*.

Decker, J. C., Sarah. (2019). Microsoft HoloLens. Retrieved from https://docs.microsoft.com/en-us/hololens/

Dickey, R. M., Srikishen, N., Lipshultz, L. I., Spiess, P. E., Carrion, R. E., & Hakky, T. S. (2016). Augmented reality assisted surgery: a urologic training tool. *Asian Journal of Andrology*(18), 732-734.

Dunbar, B. (2017). Virtual Reality Laboratory. Retrieved from https://www.nasa.gov/centers/johnson/partnerships/eddc/ra/virtual-reality-laboratory

Endsley, M. R. (1988, 23-27 May 1988). *Situation awareness global assessment technique (SAGAT).* Paper presented at the Proceedings of the IEEE 1988 National Aerospace and Electronics Conference.

Felder, R. M., & Silverman, L. K. (1988). Learning and Teaching Styles In Engineering Education. *Engineering Education, 78(7)*, 674-681.

Ferguson, D., & Stentz, A. (2005). *The Field D* Algorithm for Improved Path Planning and Replanning in Uniform and Non-Uniform Cost Environments*. Retrieved from

Godden, D. R., & Baddeley, A. D. (1975). *CONTEXT-DEPENDENT MEMORY IN TWO NATURAL ENVIRONMENTS: ON LAND AND UNDERWATER*. Retrieved from https://pdfs.semanticscholar.org/d71d/381c6371f95b4b84baa2763f147709ab3d57.pdf

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics, 4*(2), 100-107.

Hart, S. G. (2006). *NASA-task load index (NASA-TLX); 20 years later*. Paper presented at the Proceedings of the Human Factors and Ergonomics Society Annual Meeting.

Hart, S. G., & Staveland, L. E. (1988). Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology* (Vol. 52, pp. 139-183): Elsevier.

Henderson, S., & Feiner, S. (2011). Exploring the Benefits of Augmented Reality Documentation for Maintenance and Repair. *IEEE Transactions on Visualization and Computer Graphics, 17*(10), 1355-1368.

Hodges, K. V., & Schmitt, H. H. (2011) A new paradigm for advanced planetary field geology developed through analog experiments on Earth. In*: Vol. 483. Special Paper of the Geological Society of America* (pp. 17-31).

ISECG. (2018). *Global Exploration Roadmap*. International Space Exploration Coordination Group, Mulheim/Ruhr, Germany

Johnson, A. W., Hoffman, J. A., Newman, D. J., Mazarico, E. M., & Zuber, M. T. (2010). *An Integrated Traverse Planner and Analysis Tool for Planetary Exploration.*

Jones, E. M. (1995). Apollo Lunar Surface Journal. Retrieved from www.hq.nasa.gov/alsj

Koenig, S., & Likhachev, M. (2002). *D*lite*. Paper presented at the Eighteenth national conference on Artificial intelligence, Edmonton, Alberta, Canada.

Krevelen, D. W. F., & Poelman, R. (2010). A Survey of Augmented Reality Technologies, Applications and Limitations. *The International Journal of Virtual Reality, 9*(2), 1-20.

Lewis, J. R., & Sauro, J. (2018). Item benchmarks for the system usability scale. *Journal of Usability Studies, 13*(3).

Lim, D. S. S., Abercromby, A., & Kobs Nowatniak, S. (2019). The BASALT Research Program: Designing and Developing Mission Elements

in Support of Human Scientific Exploration of Mars. *Astrobiology, 19*.

MagicLeap. (2018). Magic Leap 1 Control.

Marquez, J. J. (2007). *Human-Automation Collaboration: Decision Support for Lunar and Planetary Exploration.* (Doctor of Philosophy). Massachusetts Institute of Technology,

Marzo, A., Bossavit, B., & Hachet, M. (2014). *Combining multi-touch input and device movement for 3D manipulations in mobile augmented reality environments*. Paper presented at the Proceedings of the 2nd ACM symposium on Spatial user interaction, Honolulu, Hawaii, USA. https://doi.org/10.1145/2659766.2659775

Mathiesen, D., Myers, T., Atkinson, I., & Trevathan, J. (2012). Geological visualisation with augmented reality. *Proceedings of the 2012 15th International Conference on Network-Based Information Systems, NBIS 2012*(May 2014), 172-179.

Milgram, P., & Kishino, F. (1994). *A Taxonomy of Mixed Reality Visual Displays Augmented Reality through Graphic Overlays on Stereoscopic video View project A TAXONOMY OF MIXED REALITY VISUAL DISPLAYS*. Retrieved from http://vered.rose.utoronto.ca/people/paul_dir/IEICE94/ieice.html

Muehlberger, W. R. (1981). *Apollo 16 Traverse Planning and Field Procedures*.

Narzt, W., Gustav, A. E., Ae, P., Ferscha, A., Kolb, D., Reiner, A. E., . . . Lindinger, C. (2005). Augmented Reality Navigation Systems.

NASA. (2020). Moon to Mars Overview.

Nash, A., Daniel, K., Koenig, S., & Felner, A. (2007). *Theta^*: Any-angle path planning on grids*. Paper presented at the AAAI.

Norheim, J., Hoffman, J. A., Supervisor, T., & Balakrishnan, H. (2018). *Path Planning for Human Planetary Surface Exploration in Rough Terrain.* (Master of Science). Masachusetts Institute of Technology,

Oliphant, T. (2020). NumPy Reference. Retrieved from https://numpy.org/

Patrick, N., Kosmo, J., Locke, J., Trevino, L., & Trevino, R. (2010). *Extravehicular Activity Operations and Advancements*. Retrieved from https://www.nasa.gov/centers/johnson/pdf/584725main_Wings-ch3d-pgs110-129.pdf

Ponce, B. A., Menendez, M. E., Oladeji, L. O., Fryberger, C. T., & Dantuluri, P. K. (2014). Emerging Technology in Surgical Education: Combining Real-Time Augmented Reality and Wearable Computing Devices. *37*(11).

Protalinksi, E. (2017). Google’s speech recognition technology now has a 4.9% word error rate. Retrieved from https://venturebeat.com/2017/05/17/googles-speech-recognition-technology-now-has-a-4-9-word-error-rate/

Rosenberg, L. B. (1992). *The Use of Virtual Fixtures as Perceptual Overlays to Enhance Operator Performance in Remote Environments*. Retrieved from https://apps.dtic.mil/docs/citations/ADA292450

Seyler, S. L., Kumar, A., Thorpe, M. F., & Beckstein, O. (2015). Path Similarity Analysis: A Method for Quantifying Macromolecular Pathways. *PLOS Computational Biology, 11*(10), e1004568.

Sutherland, I. E. (1968). *A head-mounted three dimensional display.* Paper presented at the Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I.

Syberfeldt, A., Danielsson, O., & Holm, M. (2015). Visual Assembling Guidance Using Augmented Reality. *Procedia Manufacturing, 1*, 98-109.

The Universal Transverse Mercator (UTM) grid (157-99). (1999). Retrieved from Reston, VA: http://pubs.er.usgs.gov/publication/fs15799

Wolfenstetter, T. (2007). *Applications of Augmented Reality technology for archaeological purposes*.

10. Appendices

A. List of Abbreviations

- ASR: Automatic Speech Recognition
- APC: Armored Personnel Carrier
- AR: Augmented Reality
- BASALT: Biologic Analog Science Associated with Lava Terrains
- CRT: Cathode Ray Tube
- DRATS: Desert Research and Technology Studies
- DV: Dependent Variable
- EMU: Extravehicular Mobility Unit
- EVA: Extravehicular Activity
- GG: Google Glass
- HMD: Head-mounted Display
- IMU: Inertial Measurement Unit
- IV: Independent Variable
- IPP: Inflatable Penile Prosthesis
- LM: Lunar Module
- LRV: Lunar Roving Vehicle
- NASA: National Aeronautics and Space Administration
- NEEMO: NASA Extreme Environment Mission Operations
- OST: Optical See Through
- PLRP: Pavilion Lake Research Project
- SEXTANT (Surface Exploration Traverse Analysis and Navigational Tool
- SLAM: Simultaneous Localization and Mapping
- TOP: Target of Opportunity
- UTM: Universal Transverse Mercator coordinate system
- VPU: Visual Processing Unit

- VR: Virtual Reality
- VST: Video See Through
- xGDS: The Exploration Ground Systems

B. How to Install / Use Current Sextant

B.1. Repository Location

https://github.com/nanastas-mit/pextant

B.2. Required Software

- Windows 10
- Python 3
- Visual Studio 2019 (or higher)
- Git

B.3. Installation Instructions

Clone SEXTANT Repository

You can clone the repository @ https://github.com/nanastas-mit/pextant using whatever method you prefer (command line, GitHub Desktop GUI, etc.), but know this: Inside the pextant_cpp folder is the pybind11 Git submodule. In order to sync submodules, you must:

- Clone the pextant repo in whatever way you prefer
- Open a command prompt (you can probably do through this via a GUI as well)
- Navigate to the local directory that you cloned the pextant repo to
- Type the following commands:

```
git submodule init
git submodule update
```

Create the 'pextant' Environment

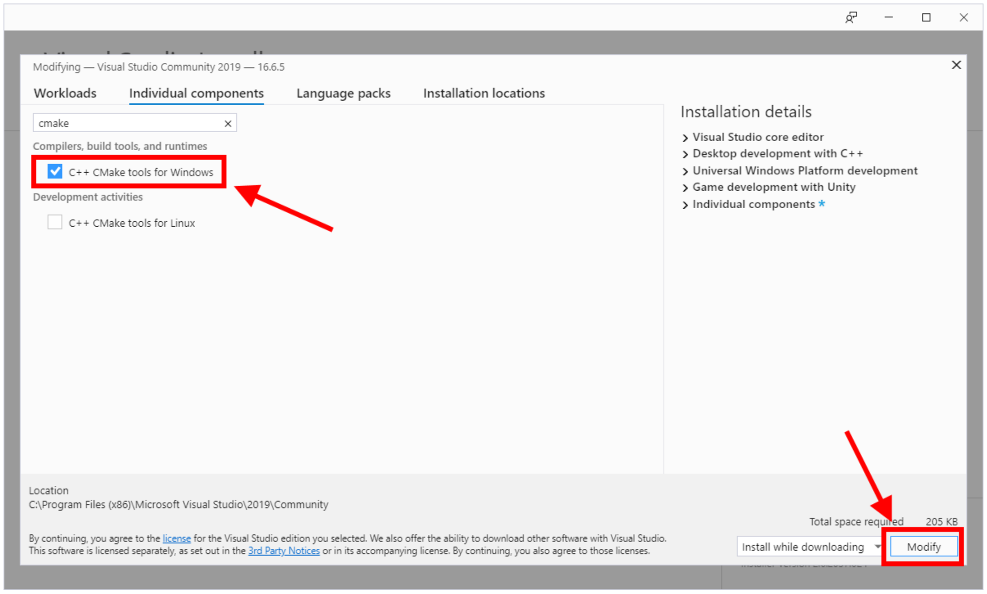
First, you need to create an anaconda environment that contains all of the necessary packages. This is most easily done using the environment.yml file. Open an anaconda prompt and type the following:

```
conda env create -f environment.yml
```

Build and Install 'pextant_cpp' package

Next, you need to build and install the custom python_cpp package. Using Visual Studio 2019 (or higher):

- Make sure C++ CMake Tools for Windows is installed
  - Open Visual Studio Installer
  - On the 'Installed' tab, click the 'Modify' button on the Visual Studio installation you'd like to use
  - On the 'Individual Components' tab, check 'C++ CMake Tools for Windows'
  - Click the 'Modify' button (should be in the lower right corner of the screen)


Workloads
Individual components
Language packs
Installation locations
Installation details
Visual Studio core editor
Desktop development with C++
Universal Windows Platform development
Game development with Unity
Individual components
Install while downloading
Modify

Modifying — Visual Studio Community 2019 — 16.6.5

cmake

Compilers, build tools, and runtimes

[✓] C++ CMake tools for Windows

Development activities

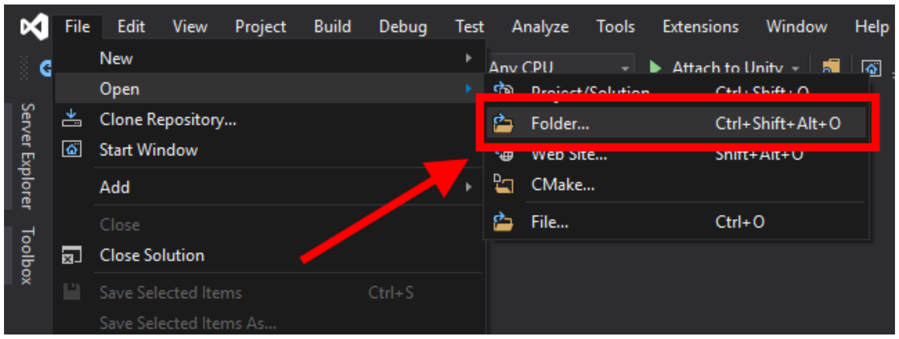
[ ] C++ CMake tools for Linux

Location
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

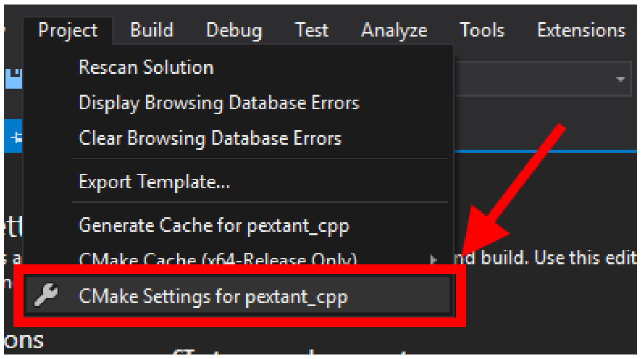
Total space required 205 KB

- Open the ‘pextant_cpp’ project in Visual Studio
  - In Visual Studio, use 'File->Open->Open Folder' on the pextant_cpp folder

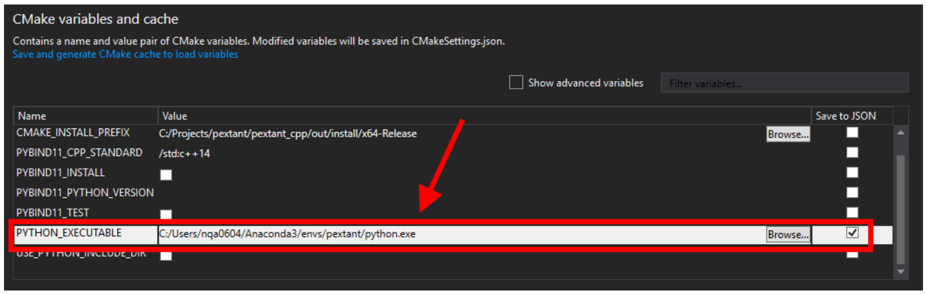


- for more info, see Microsoft 'CMake Projects in Visual Studio' documentation: https://docs.microsoft.com/en-us/cpp/build/cmake-projects-in-visual-studio?view=vs-2019

- Make sure the 'pextant_cpp' project points to the correct version of python.exe and CMake generator
  - Open the project settings using 'Project->CMake Settings for pextant_cpp' (just a special viewer for CMakeSettings.json)


Project
Build
Debug
Test
Analyze
Tools
Extensions
Rescan Solution
Display Browsing Database Errors
Clear Browsing Database Errors
Export Template...
Generate Cache for pextant_cpp
CMake Cache (x64-Release Only)
CMake Settings for pextant_cpp
nd build. Use this edit
ons

  - In 'CMake Variables and Cache', set PYTHON_EXECUTABLE to whatever python.exe you're using (You may have to check 'show advanced variables' to see PYTHON_EXECUTABLE)

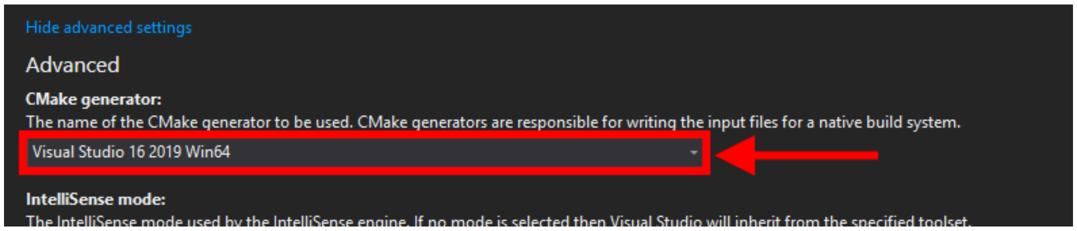

CMake variables and cache

Contains a name and value pair of CMake variables. Modified variables will be saved in CMakeSettings.json.
Save and generate CMake cache to load variables

☐ Show advanced variables Filter variables...

Name	Value	Save to JSON
CMAKE_INSTALL_PREFIX	C:/Projects/pextant/pextant_cpp/out/install/x64-Release Browse...	☐
PYBIND11_CPP_STANDARD	/std:c++14	☐
PYBIND11_INSTALL	☐	☐
PYBIND11_PYTHON_VERSION		☐
PYBIND11_TEST	☐	☐
PYTHON_EXECUTABLE	C:/Users/nqa0604/Anaconda3/envs/pextant/python.exe Browse...	✓
USE_PYTHON_INCLUDE_DIR	☐	☐

    - Under 'Show Advanced Settings', set 'CMake generator' to whatever version of visual studio you're using (e.g. Visual Studio 16 2019 Win64)



Hide advanced settings

Advanced

CMake generator:

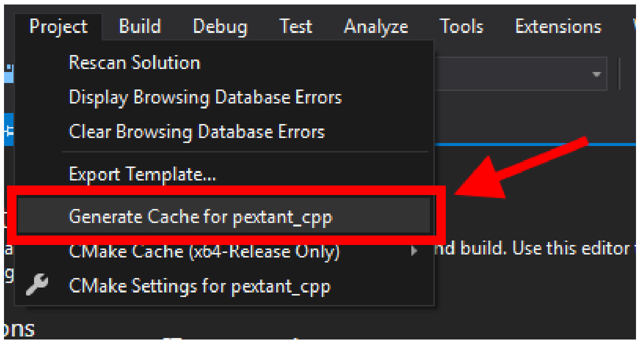
The name of the CMake generator to be used. CMake generators are responsible for writing the input files for a native build system.

Visual Studio 16 2019 Win64

IntelliSense mode:

The IntelliSense mode used by the IntelliSense engine. If no mode is selected then Visual Studio will inherit from the specified toolset.

- Build the project
    - click 'Project->Generate Cache for pextant_cpp'


Project
Build
Debug
Test
Analyze
Tools
Extensions
Rescan Solution
Display Browsing Database Errors
Clear Browsing Database Errors
Export Template...
Generate Cache for pextant_cpp
CMake Cache (x64-Release Only)
CMake Settings for pextant_cpp

    - if something goes wrong, refer to 'CMakeLists.txt' for more info
- Install module using setup.py
    - open Anaconda prompt
    - activate 'pextant' environment
    - navigate to folder containing the pextant_cpp folder/project
    - type:

```
pip install ./pextant_cpp
```

    - for this to work, you will need to have the CMake python package installed in your Python environment
- All done!
    - a pextant_cpp .pyd should now live in the site-packages folder of your python environment

Activate the 'pextant' Environment

Finally, activate the environment. In the anaconda prompt, type:

```
activate pextant
```

C. Subject Documents

C.1. Demographic Information Survey

1. Gender

   ☐ Male

   ☐ Female

   ☐ Other _____________

   ☐ Prefer Not to Say

2. Age

   ☐ 18-25

   ☐ 26-33

   ☐ 34-41

   ☐ 42-49

   ☐ 50-57

   ☐ 58-65

3. Are you right or left-handed?

   ☐ Right

   ☐ Left

   ☐ Neither / Ambidextrous

4. Do you wear glasses or contacts?

   ☐ Yes

   ☐ No

5. Are you colorblind?

   ☐ Yes

   ☐ Red-Green

   ☐ Blue-Yellow

   ☐ Monochromatic

6. How much prior experience do you have with Augmented or Virtual Reality?

☐ None

☐ Little

☐ Moderate

☐ Extensive

7. How often do you play video games?

☐ Never

☐ A few times a month

☐ A few times a week

☐ Daily

C.2. Subject Safety Protocol

Social Distancing

Current Draper protocols indicate that in-office personnel should coordinate their work spaces and schedules with their co-workers to ensure that a 6 foot minimum "safe separation distance" can be maintained at all times. Additionally, Draper is asking personnel to maintain that same 6 foot minimum "safe separation distance" as they enter, leave, and travel throughout the Draper buildings. Please follow these existing 'distancing' protocols to the best of your ability.

Pre/Post Testing

Please wash your hands immediately before and immediately after going through the test procedure. Also be sure to avoid touching any part of your face with your hands during testing.

Upon arriving and just before leaving the testing area, please spray and wipe down all shared equipment (e.g. tablet, laptop, etc.) with isopropyl alcohol or a sanitizing wipe. In addition, please wipe down the surfaces on which the shared equipment resides and any door handles that must be operated in order to arrive at the testing location.

Follow the Existing Protocol

Draper has its own COVID-19 safety protocol (which has been copied over to this document for convenience). In addition to the safety guidelines given above, please be sure to adhere to all of the guidelines in the official Draper 'Safety Protocol' document.

C.3. Post Test Survey

1. Did you find any of PATHFINDER's 'optional' features (e.g. minimap zooming, minimap orientation toggling, minimap resizing, compass, obstacle erasing) to be particularly useful, given the trial parameters and priorities? Why?

2. Did you find any of PATHFINDER's 'optional' features to be particularly use*LESS*, given the trial parameters and priorities? Why?

3. Do you think you would have used any of PATHFINDER's 'optional' features more often had the trial parameters or priorities been different (e.g. if there were no time constraint)? Why?

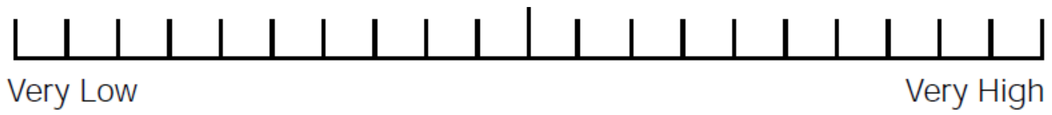
4. If you have any additional thoughts or comments about the trial or PATHFINDER in general, please write them here.

C.4. NASA Task Load Index (TLX)

Hart and Staveland's NASA Task Load Index (TLX) method assesses work load on five 7-point scales. Increments of high, medium and low estimates for each point result in 21 gradations on the scales.

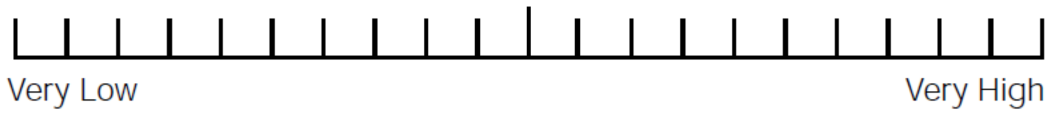
Name	Task	Date

Mental Demand How mentally demanding was the task?



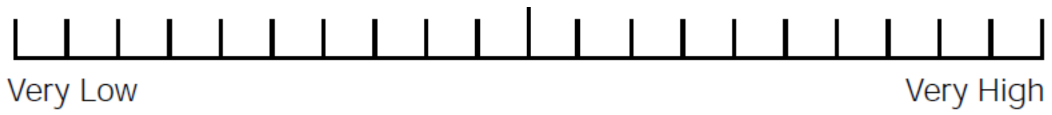
Physical Demand

How physically demanding was the task?



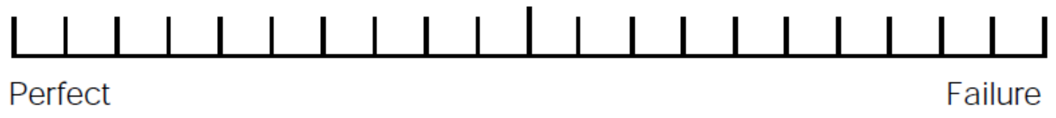
Temporal Demand

How hurried or rushed was the pace of the task?



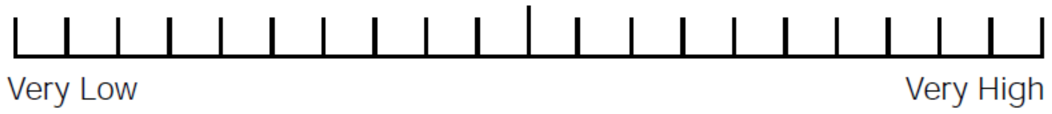
Performance

How successful were you in accomplishing what you were asked to do?



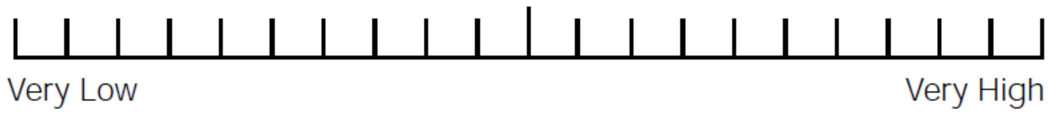
Effort

How hard did you have to work to accomplish your level of performance?



Frustration

How insecure, discouraged, irritated, stressed, and annoyed were you?



C.5. System Usability Scale (SUS)

This is a standard questionnaire that measures the overall usability of a system. Please select the answer that best expresses how you feel about each statement after using the tool today.

	Strongly Disagree	Somewhat Disagree	Neutral	Somewhat Agree	Strongly Agree
1. I think I would like to use this tool frequently.	☐	☐	☐	☐	☐
2. I found the tool unnecessarily complex.	☐	☐	☐	☐	☐
3. I thought the tool was easy to use.	☐	☐	☐	☐	☐
4. I think that I would need the support of a technical person to be able to use this tool.	☐	☐	☐	☐	☐
5. I found the various functions in this tool were well integrated.	☐	☐	☐	☐	☐
6. I thought there was too much inconsistency in this tool.	☐	☐	☐	☐	☐
7. I would imagine that most people would learn to use this tool very quickly.	☐	☐	☐	☐	☐
8. I found the tool very cumbersome to use.	☐	☐	☐	☐	☐
9. I felt very confident using the tool.	☐	☐	☐	☐	☐
10. I needed to learn a lot of things before I could get going with this tool.	☐	☐	☐	☐	☐