Interfacing Timeliner Procedural Sequencing System with the SONY AIBO Robot

by
Jacqueline Anne Lai
S.B. Computer Science at MIT, 2004

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Masters of Engineering in Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2006

©2006, Jacqueline A. Lai. All Rights Reserved.

The author hereby grants permission to M.I.T. to reproduce and
distribute publicly paper and electronic copies of this thesis in whole
or in part.

Author

Department of Electrical Engineering and Computer Science
22, 2006

Certified by ............

Robert Brown
Timeliner Task Leader, Charles Stark Draper Laboratory
Thesis Supervisor

Certified by ....................

Tomás Lozano-Pérez
Chairman, Department Committee on Graduate Students
Interfacing Timeliner Procedural Sequencing System with the SONY AIBO Robot

by

Jacqueline Anne Lai

Submitted to the
Department of Electrical Engineering and Computer Science
August 22, 2006
In Partial Fulfillment of the
Requirements for the degree of
Masters of Engineering in Computer Science

Abstract

Timeliner is a procedural sequencing system currently in use on the International Space Station. Although robust and easy-to-use, the system has never been used outside the context of sequence planning. This project demonstrates Timeliner as a control system for robots by commanding a Sony AIBO to walk, detect a wall and avoid collision with it. Two robotic programming structures are explored for implementation in Timeliner.

Thesis Supervisor: Robert Brown
Title: Timeliner Task Leader, Charles Stark Draper Laboratory

Thesis Supervisor: Tomás Lozano-Pérez
Title: Professor of Computer Science and Engineering, MIT
Acknowledgments

I would like to begin by thanking Javier Castro (MIT) and Eric Kow (LORIA) for their invaluable comments while proofreading. Thank you to Greg Andrews in the Robotics Group for being very supportive, especially when first working with the Yobotics! AIBO Simulation (which, unfortunately, never worked out). Thanks also to Richard Brunet, who was another great support! He always gave me helpful comments and simple encouragement.

Dan Swanton, another Masters of Engineering Draper Fellow, also greatly helped me with understanding Timeliner itself and learning about different Timeliner components.

Emily Braunstein is an amazing, intelligent and efficient lady who helped me by letting me look at some of her notes on Timeliner and giving me tips for the AIBOChannel application.

I would like to specially acknowledge Joseph Bondi from the Timeliner group for his patience and willingness to share - he is an inexhaustive source of knowledge and was very patient with my constant questions; I would also like to specially acknowledge Christopher Rakowski from the Robotics Group for always being there, helping me out with the AIBOs, answering any question that I asked him, and attentively making sure that I had everything I needed.

My advisors, Robert Brown and Tomás Lozano-Pérez, were very important in helping me through each step of the project. Bob always amazed me by seeing clearly through to whatever problem I was having, and often helped me to see things I had been bumbling over for weeks. Tomás was always available to help despite his busy schedule and was a major force in shaping my work to allow me to even finish. Without Tomás I would still be lost in the details.

And without my parents, who always supported and encouraged me, I would be quite incapable of even writing this small acknowledgment. Thank you!

This thesis was prepared at The Charles Stark Draper Laboratory, Inc., under Contracts NNJ06HC37C and NAS9-01069, sponsored by the NASA Manned Spaceflight Center, Houston, Texas.

Publication of this thesis does not constitute approval by Draper or the sponsoring agency of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

nne Lai

5
# Contents

1 Introduction .................................................. 11
   1.1 Organization ............................................. 12
   1.2 More About Timeliner ................................. 13
   1.3 The AIBO Robot .......................................... 14
   1.4 The project ............................................. 15
   1.5 Results .................................................. 17

2 Timeliner ................................................... 19
   2.1 The Structure of Timeliner ............................ 19
   2.2 Timeliner User Interface Language ................. 21
   2.3 Timeliner Ground Database ........................... 24
   2.4 Timeliner Script Compiler ............................. 26
   2.5 Timeliner Executor ..................................... 26
   2.6 Target System .......................................... 27

3 The Robot ................................................... 29
   3.1 Tekkotsu ................................................ 30
   3.2 Draper Algorithms ..................................... 32

4 Controlling the AIBO in Timeliner ......................... 35
   4.1 Control Levels ......................................... 36
4.2 Programming Architecture .......................... 42
4.3 Interfacing Timeliner with the AIBO ................. 47

5 Results and Discussion ............................... 53
5.1 Goal ............................................. 53
5.2 Experimentation .................................. 54
5.3 Results ......................................... 57
5.4 Lessons Learned .................................. 62

6 Future Work ......................................... 65
6.1 The Robot as a Component .......................... 65
6.2 Minimizing Timeliner .............................. 65
6.3 Handling Sensor Malfunction ........................ 66

A Timeliner System Details ............................. 67
A.1 Thorough Timeliner Overview ....................... 67
A.2 GDB Definition Documents ......................... 67
A.3 Defining Ports .................................. 81
A.4 Timeliner Scripts ................................ 81

B AIBOChannel Specifications .......................... 89
B.1 Communication .................................. 89
B.2 SimpleMonitor ................................... 91

C AIBO Specifics ...................................... 93
C.1 Draper Leg Definitions ............................ 94

D Useful References ................................... 97
List of Figures

1-1 Sony’s AIBO ERS-7 Robot .............................................. 14

2-1 Four components of the Timeliner System: Ground Database, Script Compiler, Timeliner Executor and Target System. ............... 20

2-2 This example Timeliner script prints a notifying message to the user if the measured temperature reaches 100 degrees. Otherwise, it prints that the system has cleared. The variable TEMP is a Timeliner CVT variable defined in degrees, and the text BEFORE 360 indicates that the sequence runs for 360 seconds. ............................................. 22

2-3 Example sequences that have ambiguous results due to parallel execution in Timeliner. ................................................... 23

2-4 Usage of a subsequence that solves the ambiguity problem in Figure 2-3. 24

2-5 A diagram of how the GDB is defined and created with four configuration files. ............................................................... 25

4-1 The 3 levels of control present in this system. The first lies with the AIBO itself and operates on the order of milliseconds, the second with Timeliner on the order of seconds, and the third with a human operator operating on the order of hours or days. The boundary cases are those which may be controlled at more than just one level. ............. 36

4-2 A diagram of closed-loop control of robots, described by [20]. .... 37
4-3 This figure shows how the processing that occurs on the AIBO can function as a complete circle. Timeliner provides additional functionality by giving directions to the computer controlling the actuators.

4-4 This Timeliner script demonstrates one method of running two sequences in parallel. The commands \texttt{ESTOP\_TOGGLE} and \texttt{WALK} have been compiled into the GDB, and the variables \texttt{STATUS\_LB\_CONTACT}, \texttt{STATUS\_LF\_CONTACT}, \texttt{STATUS\_RB\_CONTACT}, \texttt{STATUS\_RF\_CONTACT} and \texttt{MONITOR\_ESTOP\_ON} have been compiled into the executor's CVT.

4-5 The Basic interaction scheme between the Timeliner executor and the AIBO.

4-6 An example command definition of \texttt{LEGSTOP} for the \texttt{commands.txt} file. This definition tells Timeliner byte-by-byte what to send to the target system when the command is called from a sequence.

4-7 An example definition of datatype \texttt{AIBO\_BUTTONS} for the \texttt{attributes.txt} file.

4-8 Example instance declarations for \texttt{instances.txt} which allocate space in the Timeliner CVT to the data types defined in \texttt{attributes.txt}.

4-9 An diagram of the interaction scheme between the Timeliner executor and the AIBO (updated from Figure 4-5). The job of the AIBOChannel application is to translate data that goes between the AIBO and the Timeliner executor.

A-1 A diagram depicting Timeliner data flow throughout its components and the target system.

C-1 4-bit representation of the dog's legs for the Draper algorithms. Leg 0 is represented by the highest order bit and leg 3 by the lowest order bit.
Chapter 1

Introduction

This project demonstrates that Timeliner [13], an existing system used for procedural sequencing, is capable of controlling a robot to respond to its environment. In order to understand the utility of this statement, we first ask and answer some questions.

What is Timeliner? Timeliner is a system which consists of a scripting language, a compiler and an executor. It is used specifically for processing data from and sending commands to target systems. These commands are used to automate procedural tasks within the target system. Timeliner is currently used to control procedures on board a space station, allowing the maintenance of certain onboard tasks to only require a minimum of human intervention. For example, instead of having a human regularly check the temperature of a test chamber, the temperature of the test chamber can be automatically monitored and controlled by Timeliner. However, in cases where the system cannot perform a task, such as to replace a burnt out light bulb, humans can be alerted to perform the task.

The U.S. and Japan both employ Timeliner on the International Space Station (ISS) to automate onboard operations and experiments. The goal of this project is to extend Timeliner’s capabilities to include robotic control such that the robot reacts to its environment. Especially for systems that are already controlled by Timeliner, this
extension will be useful for performing the monotonous tasks that are still performed by humans. The Timeliner scripting language may be simple, but it has all of the capabilities for basic control flow found in more complex programming languages.

**How can we control a robot?** There are computers both inside and outside a robot, communicating so as to control the robot. The Timeliner Executor is one such program that can send commands to control machinery externally. Use of the motion and walking algorithms developed by Draper Laboratory’s Robotic Group on the robot allows Timeliner to simply send initiation commands to the robot to make it move.

**What robot?** Sony’s AIBO is a powerful and flexible dog-shaped robot. This power and flexibility make it an ideal research tool that is relatively inexpensive, and many institutions have used it for robotic research. The AIBOs even have an exclusive league in the annual RoboCup soccer tournament! The AIBO provides an ideal environment for initial experiments in extending Timeliner to robots, especially because basic functionality such as walking, stopping and turning is readily available for download onto the robot.

### 1.1 Organization

The rest of this introductory chapter gives more background on Timeliner, the AIBO and the goals and results of the project.

The second chapter explores the Timeliner system in detail. The third chapter describes some relevant robot programming research and the current state of the AIBO. The fourth chapter describes the process of using Timeliner to program the AIBO. The fifth chapter discusses the results of the project, and the sixth chapter presents future possibilities.
The appendix includes files and information useful for replicating the process of integrating Timeliner with another robot. For those not interested in actually developing a system with Timeliner, the appendix can be skipped.

1.2 More About Timeliner

Timeliner is a system that includes the Timeliner scripting language, the Timeliner compiler, the ground database and the Timeliner executor. It is used to control procedures, such as temperature regulation or fault detection and recovery, in systems by monitoring sensors and sending commands.

The Timeliner scripting language, also referred to as the Timeliner User Interface Language (UIL), is especially suited for people without programming backgrounds, because it is an English-based language with a simple structure. Each Timeliner script consists of a bundle, broken into sequences and subsequences. The compiler processes these scripts, using a user-defined ground database mapping to relate references in the scripts to components of the target system. The output of the compiler is an executable, one for each bundle written in the Timeliner language. A single 'program' may include many bundles. The Timeliner executor runs sequences from bundles installed in memory and can run multiple sequences in parallel. These sequences can also start and stop other sequences. Since control systems generally depend on simultaneously monitoring the environment and operating the system, Timeliner's capability to interact with sequences from within other sequences simplifies programming of control systems.

Another advantage of developing with Timeliner is that it is a proven tool for procedural automation. ISS operators with NASA are already familiar with the Timeliner system and scripting language because they have been using it since the early 1990s.
1.3 The AIBO Robot

The AIBO is a dog-shaped robot, shown in Figure 1-1, with a lot of physical motion and data gathering capability. It has full use of its four legs, head and tail, and can detect pressure on its paws, back, jaw and head. The dog is equipped with a forward-facing camera, IR distance sensors, microphones and a speaker.

Draper Laboratory’s Robotics Group, which has had success developing movement algorithms with other robots, also developed a walking algorithm and some leg control of their own for the AIBO. The Draper robots are AIBO ERS-7s installed with the Draper algorithms and with the functionality found on the downloadable memory stick from the Carnegie Mellon Tekkotsu website [17]. Tekkotsu is an open-source application development framework developed at Carnegie Mellon University for controlling robots. It has been developed especially for the AIBO, and functions on top of Sony’s Open-R API on the robot. No changes were made to the robot or its “brain”, the memory stick, during this project.
1.4 The project

The goal of this project is to demonstrate that Timeliner can be used for supervisory control of robots such that they perform tasks and respond to their environment. Since Timeliner itself is a system especially built for middle-level control – decision making on the scale of seconds, as opposed to milliseconds or hours – it is important to keep in mind that the robot will be controlled at the higher and lower levels by other factors. For example, the simple act of moving a paw from its current location to another location near the shoulder requires deciding how much to move the shoulder joint and how much to move the elbow joint. This sort of low-level joint control, also called *kinematics*, is more appropriate for control directly on the robot than supervisory control in Timeliner. On the other hand, commanding the AIBO to walk a certain number of steps is appropriate for Timeliner. With such limitations in mind, we can focus on Timeliner performing its job at the most appropriate level of control.¹ We can also focus on the boundary cases that allow us to define the limits of Timeliner’s control over the robot.

For systems currently using Timeliner, the extension of Timeliner to include robotic control will simplify the addition of robots to the system in two ways. First, the operators are already familiar with Timeliner, so the cost of learning and remembering how to work with an entirely new control system is avoided; second, overhead associated with integrating the two control systems will be avoided. For example, one alternative to using Timeliner would be to remotely control the AIBO using the Java ControllerGUI which was developed to work with the Tekkotsu API [17]. If we wanted to be able to communicate between Timeliner and the ControllerGUI, however, it would be necessary to develop a way of sharing information between both systems. Instead of using a new system which runs side-by-side with Timeliner to

¹For more reading about Timeliner and its position in the control hierarchy, see Daniel Swanton’s thesis on Timeplanner [16] and Erann Gat’s paper on three-layer architecture [8].
interactively control the robot, Timeliner can simply be adapted to control the AIBO, and act as the interface between human operators and the AIBO.

I developed Timeliner’s robotic control using build DE.028 of Timeliner and the ERS-7 AIBOs from Draper Laboratory’s Robotics Group. In addition to integrating Timeliner to control the AIBO, I was also able to demonstrate the functionality of TIDE, the Timeliner Integrated Development Environment developed by a previous Master’s Thesis at MIT, Maya Dobuzhkaya. [4]

1.4.1 Objective

The key objective of this thesis is to show Timeliner’s capability to establish a closed-loop control system (see Figure 4-2) with the robot. Control of the AIBO progressed step-by-step, starting with basic tasks and advancing to more complicated tasks. The AIBO can currently follow commands sent by Timeliner to perform the following tasks, demonstrating that Timeliner can indeed be used for robotic control.

- Walk in a straight line for a specified distance.
- Respond to pressure on its foot pads.
- Rotate at different speeds.
- Walk in a straight line, stop and then turn left.
- Walk in a straight line intersected by a wall, detect the wall and turn to walk in another direction.

The Timeliner executor is also capable of running many sequences in parallel, which makes adding and removing tasks from control of the robot simple. For example, it is possible to combine the tasks of walking and responding to foot pressure to make the AIBO walk straight when there is pressure on all feet, and rotate otherwise.
The ultimate goal for this project was for Timeliner to control the AIBO to walk on a table without falling off the edge. Although the goal was not accomplished, this project provides a framework and suggestions for its achievement.

1.5 Results

The results of using Timeliner to control the AIBO show that Timeliner is capable of performing the high level decision-making necessary to control a robot. However, for full integration of the robot, it is necessary to ensure that the robotic system understands instructions sent by Timeliner. In order to avoid actually reprogramming the robot, an intermediary program (AIBOChannel) worked as a bidirectional translator between Timeliner and the AIBO:

- From the AIBO to Timeliner, sensor data is reformatted to be understood by Timeliner.
- From Timeliner to the AIBO, instructions sent by Timeliner are translated into packets understandable by the robot.

The Timeliner UIL itself is pretty flexible and allows for the user to decide what sort of control structure he would like to use for the system.

In the future, a better way of using Timeliner to control a robot would be with high-level control scripts in the Timeliner executor that directly command functions on the robot. This requires direct programming of the robot to provide an interface that Timeliner can work with. The robot should be programmed to provide an interface with the necessary basic functions to give Timeliner (or another control system) the flexibility to manipulate the robot freely, without the need for a messy intermediary program.
Chapter 2

Timeliner

Timeliner was originally developed to automate procedural tasks at Draper Laboratory. In 1982, it was developed and used to simulate onboard crew actions for tests of the Space Shuttle system. In 1992, NASA chose Timeliner for use on Space Station Freedom, and eventually on the International Space Station. Currently, the United States and Japan both use Timeliner for core and payload operations on the International Space Station. Core operations maintain the normal functioning state of the station and payload operations are for testing and experimentation that take place on the station.

2.1 The Structure of Timeliner

Timeliner acts as an interface between its human operator(s) and a target system. Timeliner connects to a target system, receives data from the target system, and sends commands to the target system based on that data and human operator input. It is specially suited for control on a “human” scale, as suggested by the fact that Timeliner was engineered to simulate human actions in a space shuttle to test the shuttle’s space-worthiness. Most of these procedures require decisions on the order of seconds. It has not been used for making decisions on the orders of either microseconds.
or hours and days. In fact, the current build (DE.028) of Timeliner has a running cycle of 1 Hz, so passes (see Section 2.5) executed in Timeliner run at most once per second.

Timeliner itself is composed of many components. The following main components are diagrammed in Figure 2-1 and described in this chapter in further detail.

**Timeliner User Interface Language (UIL)** The UIL is a key part of Timeliner’s usability, because it is similar to written English. The language was found to be easy-to-learn by space shuttle operators unfamiliar with computer programming. Control sequences run by Timeliner are written in this language.

**Ground Database (GDB)** The GDB stores binary definitions of commands that can be executed by the target system, as well as variables used for describing the state of the target system. The Ground Database is not to be confused with the Gnu Debugger, which is not mentioned in the rest of this document.
**Timeliner Compiler** Timeliner scripts are written in the Timeliner UIL and are compiled by referencing commands and variables in the GDB. The compiler outputs Timeliner executables.

**Timeliner Executor** This component runs the executables output by the compiler. The executor is what actually sends commands to the target system by running one or more sequences described in the Timeliner scripts. The executor also processes telemetry into its Current Value Table (CVT), which is either sent from the target system or set by running sequences.

**Target System (AIBO)** In this case, the target system is the AIBO. In other cases, the target system could be an experiment that requires constant monitoring or fuel canister refilling on board a space shuttle. The target system must have an interface that Timeliner can work with to control.

### 2.2 Timeliner User Interface Language

The Timeliner language has a relatively simple structure. It contains bundles, sequences and subsequences for modularizing scripts. The sample script in Figure 2-2 shows some of the time-based control constructs that are prominent in Timeliner. Other commands include EVERY, which functions similarly to a for loop; WITHIN, which functions like a while loop; and WHENEVER, which is like a looped if/else command. The Timeliner UIL Programmer’s Reference [12] is the best place to look for more detailed examples and specific syntax of Timeliner script.

Each script file contains one bundle, which can contain multiple sequences and subsequences. Sequences are constructs that can be started, stopped and resumed externally by other sequences running parallel to them. They can also be started automatically when their parent bundle is installed (see Section 2.5) with the ACTIVE keyword. When the bundle in Figure 2-2 is installed by the executor, the TEMP STEADY
Figure 2-2: This example Timeliner script prints a notifying message to the user if the measured temperature reaches 100 degrees. Otherwise, it prints that the system has cleared. The variable TEMP is a Timeliner CVT variable defined in degrees, and the text BEFORE 360 indicates that the sequence runs for 360 seconds.

sequence is automatically activated because it uses the ACTIVE keyword. Subsequences are lines of script that, when called, act as if they have been inserted into the calling script and run inline instead of parallel. Subsequences are started with the CALL keyword.

2.2.1 Running Sequences in Parallel

The capacity for running sequences in parallel is key in allowing Timeliner users to easily write scripts which effectively control their target systems. However, the usual problem associated with shared resources and actions running in parallel still applies.

The example in Figure 2-3, while it does not accomplish much, demonstrates a case in which the result of running sequences A and B is ambiguous. If the CVT (CVT variables are global variables that can be accessed by any sequence run by the executor, see Section 2.5) contains a variable X, running sequences A and B in parallel is different from independently running A first and then B or B first and then A.

If sequence A is executed first, the resulting value of X is 1, and both messages SEQUENCE C and SEQUENCE B are printed. On the other hand, if the Timeliner executor could not run these sequences in parallel, the result would be X = 1 and only
SEQUENCE A
  STOP C
  IF X = 0 THEN
    WAIT 1
    START C
    SET X AS 1
  END IF
CLOSE SEQUENCE

SEQUENCE B
  IF X = 0 THEN
    SET X = 2
    MESSAGE "SEQUENCE B"
  END IF
CLOSE SEQUENCE

SEQUENCE C
  MESSAGE "SEQUENCE C"
CLOSE SEQUENCE

Figure 2-3: Example sequences that have ambiguous results due to parallel execution in Timeliner.

To avoid or control situations like these, access to shared resources such as the CVT variables must be limited to a single sequence of tasks or else strictly regulated. In the above example, if sequence B were unable to access X until sequence A had finished, there would be no conflict. One way to enforce this would be to STOP B before A accesses X, and then RESUME B after A modifies X. Another option would be to encapsulate access to X in a subsequence which tests which sequence is currently active to decide what action to take, demonstrated in Figure 2-4.

This works because the execution of sequences and subsequences is serialized, as described in Section 2.5. Whenever possible, however, it is best to limit read/write access to a shared resource to one sequence.

In this project, sharing resources does not pose a problem, because the CVT variables are only modified once every pass by telemetry data coming from the AIBO.
Since the sequences do not modify the CVT variables, they are free to read the variables as much as they would like, as long as actions taken on account of the variables occur in the same cycle as the read of the variable.

2.3 Timeliner Ground Database

The ground database stores information about the target system, information which is referred to by the compiler to ensure that the scripts will work correctly when used with the target system. So, the GDB needs to know about commands and their syntax, and any telemetry that will be sent from the target system to be processed by Timeliner.

The GDB is generated by the gdb.exe executable, which takes as input four configuration files in order to generate the database, as illustrated in Figure 2-5.

- **gdb_input.script** Tells gdb.exe where to put the binary output files and where to find the three main input files. There are only 4 lines in this file. When using TIDE, this file is also automatically generated.

- **commands.txt** Describes how the commands sent from Timeliner must be
Figure 2-5: A diagram of how the GDB is defined and created with four configuration files.

- **attributes.txt** Builds some complex datatypes out of the basic Timeliner datatypes [11] to describe the variables that will be in the CVT.

- **instances.txt** States how many of each of the datatypes described in attributes.txt are in the CVT and what their names are. Timeliner actually has two execution environments, a test and a real environment. Since the two environments often require different definitions of their CVT variables, there are two instances.txt files, one for each environment.

  gdb.exe's binary output files can be found in the path specified by gdb_input.script. There are other output files that specifically describe the addresses and sizes of contents of the CVT.
2.4 Timeliner Script Compiler

The Timeliner script compiler checks the validity of the Timeliner scripts and compiles them into executables readable by the executor. In order to be able to check the validity of the scripts, however, the compiler also needs to know about the commands and variables for the target system that are referenced by the script. The compiler references this information from the GDB.

2.5 Timeliner Executor

The Timeliner executor loads the executable files (*.tix files) created by the compiler and executes them as requested by the operator. The executor is the component of Timeliner that does the work of running sequences, sending UDP commands to the target system, and storing and evaluating data that arrives from the target system. As described in Section 2.2.1, the executor has the capacity for running sequences in parallel. The executor functions by running through its memory and executing all active sequences in order. A sequence may be skipped if it is blocked by a WAIT command or calls upon a busy subsequence – in which case the executor goes on to the next active sequence. This run-through of all active sequences is called a pass. Once the executor has finished one pass, it waits until it is time for the next pass, which is decided by the cycle speed.

A single sequence A cannot be executed in parallel with itself. If it is already running and sequence B makes the call START A, nothing will change. Neither can a single subsequence a be executed in parallel with itself. If sequence A calls a and then B calls a while A is still running a, B halts until the first call of a finishes.

The executor also allows for dynamic installation and uninstallation of bundles in memory, so that the executor may continue to run while the human operator modifies and adds scripts for installation and execution. A unique feature that results from
the simplicity of loading and unloading bundles from the executor and the ability
to start and stop sequences manually is that debugging Timeliner script is relatively
simple. It is simple to test and debug each sequence in a standalone fashion by simply
running it in the executor.

The Current Value Table (CVT) stores global variables that the executor can
reference and update. The scripts written by the human operator can update the
CVT, and telemetry sent by the target system can update the CVT as well. The
CVT contains the only variables that are global to the entire system, because the
scope of a bundle variable is limited to its bundle and the scope of a sequence variable
to its sequence.

2.5.1 Cycle Speed

Timeliner generally runs at 1 Hz. This speed was determined with the ISS demands
in mind, but Timeliner can actually handle much higher speeds. The commercial
version of Timeliner, Auspice [1], uses a cycle speed of 50 Hz for its normal operation.

2.6 Target System

The target system must be able to give Timeliner the input it needs and handle the
output coming from Timeliner. The target system sends Timeliner its telemetry data,
described by the attributes and instances files from the GDB. Timeliner requires that
incoming data follow a specific format: first is the address of the data (where it is
located in the CVT), second is the data’s size in bytes, last is the data. Timeliner’s
commands are sent to the target system as UDP packets, formatted as described in
the commands GDB file. In order to interpret the commands, the target system must
be able to receive UDP packets.
Chapter 3

The Robot

The AIBO ERS-7 is a powerful dog-shaped robot capable of a multitude of actions. The annual Robocup tournament even includes a 4-legged league composed entirely of AIBOs. Conducting research with the AIBOs is made even simpler by the many resources made available by universities and research groups worldwide. Appendix C has full details of the functionality of the AIBO ERS-7.

Draper's AIBOs are installed with Tekkotsu [17], an open source programming framework developed at Carnegie Mellon, within which Draper’s own motion algorithms are integrated. The robot is equipped with a wireless network card that allows it to connect to a wireless network and accept commands sent to its IP address (coded into the memory stick). It also sends a continuous stream of data from its sensors to a specified UDP port when the data is requested. The following two sections describe the systems and their wireless communication further.

A note about the AIBO is that its processor uses the big-endian byte order (highest order byte comes first), which is the opposite of PC/Intel byte order.
3.1 Tekkotsu

The Tekkotsu framework handles control of the AIBO’s motion, sensors and sound, as well as interaction with control from outside of the AIBO. The interface works with button presses on the robot and/or wireless communication over TCP and UDP.

3.1.1 Commanding the AIBO

When a computer makes a TCP connection to the AIBO server, the robot sends menus back to the computer. The menus list the capabilities of the robot, and can be navigated to open up different functionalities. Table 3.1 lists the options found in the TekkotsuMon menu.

Tekkotsu also allows the user to control the AIBO by pressing its buttons to navigate the menus, in the case that there is no computer from which the user can control the robot.

3.1.2 Emergency Stop

The emergency stop (estop) is an important function for any robot. If the robot performs movement that can be harmful to itself or its surroundings, the quickest way to save the robot is by activating the emergency stop to halt the robot’s movement. On the AIBO, the emergency stop is activated manually by a double-tap on the back button sensor. When the emergency stop is activated, the back sensor lights pulse orange. Emergency stop commands can also be sent to the AIBO through the wireless connection, as indicated in the above menu listing. The EStopController option is activated by default. Since the emergency stop can be activated remotely, Timeliner can use this very important feature to stop the robot when necessary.

The emergency stop causes the robot to freeze and hold its current position. The held position may also cause stress on the robot’s joints, so the controller should
<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draper UDP Control</td>
<td>Listens to aibo3d control commands coming in from port 20051 (this option is activated by default, so whenever the AIBO is on, it is ready to receive the draper commands described in 3.2).</td>
</tr>
<tr>
<td>Head Remote Control</td>
<td>Listens to head control commands coming in from port 10052.</td>
</tr>
<tr>
<td>Walk Remote Control</td>
<td>Listens to walk control commands coming in from port 10050.</td>
</tr>
<tr>
<td>View WMVars</td>
<td>Brings up the WatchableMemory GUI on port 10061 (connects to WMMonitor Behavior, this just launches the GUI).</td>
</tr>
<tr>
<td>Watchable Memory Monitor</td>
<td>Bidirectional control communication with WMMonitor on port 10061.</td>
</tr>
<tr>
<td>Aibo 3D</td>
<td>Listens to aibo3d control commands coming in from port 10051.</td>
</tr>
<tr>
<td>World State Serializer</td>
<td>Sends sensor information to port 10031 and current pid values to port 10032.</td>
</tr>
<tr>
<td>Raw Cam Server</td>
<td>Forwards images from camera over port 10012.</td>
</tr>
<tr>
<td>Seg Cam Server</td>
<td>Forwards segmented images from camera over port 10012.</td>
</tr>
<tr>
<td>EStopController</td>
<td>Listens to estop (see Section 3.1.2) commands coming in from port 10053.</td>
</tr>
</tbody>
</table>

Table 3.1: Options found in the TekkotsuMon menu on the AIBO.
make a decision about getting the robot to move normally as quickly as possible. Once the controller decides how to proceed with the robot, the emergency stop can be stopped, but unfinished commands will continue to run on the robot unless they are cleared. The commands can be cleared from the robot if the module that receives the commands is deactivated.

### 3.1.3 Sensor Data

The AIBO’s many sensors, joints and buttons all provide feedback, which can also be sent wirelessly to an external computer if the computer requests that feedback. The computer sends the string “connection request” as a UDP packet from an open port to port 10031 on the AIBO, which starts a continuous stream of packets from the robot to the computer, containing sensor data such as leg rotator positions, elevator positions, knee positions, button status, head position, tail position, IR distance detection, accelerometer data and remaining power. A complete listing of the data sent in the telemetry packet can be found in Appendix A.2.2 as the class AIBOREAL-TYPE.

Section 4.3.2 describes how Timeliner handles the stream of data.

### 3.2 Draper Algorithms

The Draper algorithms installed on the robot consist of the following commands:

- **Walk** (legsUsed, walkHeight, speed, numSteps, legLiftHeight, forwardFoot, aftFoot, walkWidth, gaitType, forceThreshold, crabAngle, completionThreshold): A walking motion that moves the AIBO one leg at a time.

- **Legstop** (legsUsed): Stop the leg(s) indicated by legsUsed from moving.
- **Legarc** (legsUsed, rotationAxis, arcX, arcY, arcZ, arcAngle, turnRate, forceThreshold): Move the leg(s) indicated by legsUsed in the arc defined by the parameters.

- **Leglinear** (legsUsed, footX, footY, footZ, speed, forceThreshold): Move the leg(s) indicated by legsUsed in a linear motion to the position (footX, footY, footZ) relative to the respective shoulder joint.

- **Legquadratic** (legsUsed, footX, footY, footZ, liftHeight, speed, forceThreshold): Move the leg(s) indicated by legsUsed in a quadratic motion to the position (footX, footY, footZ) relative to the respective shoulder joint.

These commands can be initiated by UDP packets sent to the dog. These commands were integrated with the Tekkotsu framework, and the functionality is active by default, so no special configuration is necessary before sending the commands to the AIBO. The full command formats are specified in Appendix A.2.
Chapter 4

Controlling the AIBO in Timeliner

Interfacing Timeliner with the AIBO and testing Timeliner’s control over the robot consisted of the following stages:

1. Defining Timeliner commands to interact with the AIBO,
2. Defining the executor’s CVT variables for interaction with the AIBO,
3. Building the architecture of scripting and programming in Timeliner for supervisory control of the AIBO, and
4. Writing scripts to perform certain tasks with the AIBO.

An important aspect of the project was determining how Timeliner would be most useful for controlling the AIBO to draw distinctions among three types of control: control on the AIBO itself, control that Timeliner had over the robot, and control attributed to humans over Timeliner and the robot. Control on the AIBO was pre-set to what existed on the robot, as described in the previous section. It was decided not to change the programming of the AIBO itself, but in hindsight this decision may have been the wrong one (see Section 5.4.1).

Another important aspect was determining a good architecture for giving Timeliner supervisory control over a robot. Timeliner’s language of bundles and sequences
Figure 4-1: The 3 levels of control present in this system. The first lies with the AIBO itself and operates on the order of milliseconds, the second with Timeliner on the order of seconds, and the third with a human operator operating on the order of hours or days. The boundary cases are those which may be controlled at more than just one level.

is good for abstracting actions that run parallel to each other, and is flexible enough to allow for scripting of different types of architecture.

This project used a Draper Lab AIBO ERS-7, a Timeliner machine, TIDE [4] and wireless equipment.

The rest of this chapter discusses a three level control structure with Timeliner inserted in the middle, a programming structure with Timeliner's scripting language, and details of configuring Timeliner for the AIBO.

4.1 Control Levels

A goal of this project was to make Timeliner functional with the AIBO while allowing a maximum of flexibility for the user at the scripting level. In order to give flexibility to the user, however, it was necessary to clearly decide what tasks fell under the control of the robot, Timeliner, or the human user. The interfaces between these three levels of control had to be well-defined, and the boundary cases between different control levels, mentioned in Section 1.4, had to be designated to their appropriate control categories to clearly understand the scope of Timeliner’s control over the robot.
The control that exists on the robot should take into account time sensitive situations that require a faster response time than is possible with Timeliner. For example, in one of the walking algorithms used by the AIBO, the robot has to temporarily lose balance because it lifts one of the four legs it normally depends upon. The time required for the robot to receive and carry out subsequent lift leg and lower leg commands from Timeliner indicates that the robot must be stable in the three-legged position while the fourth leg is lifted and then repositioned. If the robot is not stable in the three-legged position, it will fall when the fourth leg is lifted. This means that it is best for walking commands to be directly programmed into the robot, instead of relying on Timeliner to direct the AIBO in each motion necessary for walking.

Figure 4-2, borrowed from Wahl and Thomas [20], demonstrates a typical closed-loop structure in robotic programming, where the programming is abstracted away from the real-world aspects of robotics. In this project, this loop exists on the robot itself, as well as off the robot, as shown in Figure 4-3.

The control that Timeliner has over the robot can best be defined as a type of *supervisory control* that directs the robot to perform actions to accomplish one or
Figure 4-3: This figure shows how the processing that occurs on the AIBO can function as a complete circle. Timeliner provides additional functionality by giving directions to the computer controlling the actuators.

more goals. For example, Timeliner can run a script that tells the AIBO to walk a specified route and to bark whenever it sees a marker. A more useful script might notify the system whenever the AIBO’s battery level has reached a certain level, and direct the AIBO to the nearest charging station. The AIBO is able to walk on its own, but Timeliner is the entity that tells it where to walk, where to turn, or when to stop by processing the data that the AIBO sends to Timeliner detailing its sensed environment.

The human operator should only need to intervene in situations that Timeliner cannot control. In the event that the AIBO is unable to reach a charging station before its battery dies, an operator is needed to replace the battery. Another case where human interaction with the AIBO is necessary is when the robot has fallen and cannot get up. In both of these situations, Timeliner needs to notify its user to restore the AIBO to a functioning position. A human operator’s main job in this system is that of adding new functionality and goals to the robot by writing the Timeliner scripts that control the AIBO.
4.1.1 Supervisory Control

Sheridan [15] defines supervisory control by comparing it to a human supervisor’s interactions with his subordinates.

The term [supervisory control] is derived from the close analogy between the supervisor’s interaction with subordinate human staff members in a human organization and a person’s interaction with “intelligent” automated subsystems. A supervisor of humans gives directives that are understood and translated into detailed actions by staff subordinates. In turn, subordinates collect detailed information about results and present it in summary form to the supervisor, who must then infer the state of the system and make decisions for further action. The intelligence of the subordinates determines how involved their supervisor becomes in the process. (Introduction [15])

Timeliner’s role in the Timeliner-AIBO system is that of the supervisor over its subordinate, the AIBO. Timeliner gives the AIBO tasks to complete, and the AIBO gives Timeliner data that reflects the state of the task. On a higher level, the human operator functions as the supervisor to Timeliner, who gives Timeliner instructions and to whom Timeliner gives data when necessary.

4.1.2 Boundary Cases

Control in most situations clearly lies with one of these three control levels: the AIBO, Timeliner or the human operator. Those situations for which it is less clear are boundary cases.

The following are some example boundary cases which have been designated to be controlled by one of the control levels. It is important to designate such boundary cases to allow for a more concrete definition of the domains of each type of control.
These are important, for instance, in clarifying the job of Timeliner.

In the case where Timeliner loses contact with the AIBO (the robot falls out of the network or the AIBO battery dies), the job of Timeliner is to notify an operator to fix the problem and allow Timeliner to regain contact with the AIBO. This is a case for which the control needs to be clarified because, for instance, in a larger system, Timeliner might instead contact another robot to find the AIBO and reset it or recharge it, thereby leaving the operator out of the problem entirely. For this project in particular, the task of locating the AIBO and putting it back online is defined under control of the operator.

The AIBO itself should be able to override commands that tell it to do conflicting motions, such as moving two paws to the same position, but this control may fail. In the event that this control fails, Timeliner must be able to catch such failures and send the AIBO an emergency stop command. This is an especially time-critical situation because the robot may harm itself by losing stability and falling, so preferably, the AIBO prevents such situations directly. If the AIBO’s built-in safeguards fail to prevent such dangerous situations, it is the job of Timeliner to detect such cases using the telemetry sent by the AIBO and handle the situation by sending an emergency stop command, or in the most extreme cases, turning the AIBO off.

For this project, Timeliner is being used to command the AIBO to walk around. The most important boundary cases here are the ones where the AIBO is in collision with an obstacle, or is about to fall off the edge of a table. Ideally, the robot itself can use its many sensors to determine when there is an obstacle in its path or when it no longer has ground to walk upon. In fact, the AIBO does sense thresholds of pressure on its joints and stops its movement until the pressure releases. However, the built-in walk commands do not look for holes in the AIBO’s path to stop the robot from falling. The control is then pushed up to Timeliner. The AIBO has a camera in its head that provides vision, as well as 3 IR distance sensors in its chest, but the IR
distance detection data sent by Tekkotsu is insufficient for fully detecting obstacles and the camera data is not part of the default telemetry packet. Experiments suggest that combining the camera and IR data works better than either one of the sensors alone for obstacle detection. [19] In order to detect holes in the ground without using the camera and IR data, the AIBO can detect that its paws are no longer pushing on the ground, but by that point, it is already too late to prevent the robot from walking into a dramatic fall.

Another case that falls under Timeliner control is one in which the AIBO stops moving while performing a command sequence. Timeliner can handle this case by running a parallel monitoring sequence that checks the telemetry data for cases where none of the joints are moving. In this case, Timeliner should then stop currently active motion sequences and start a sequence which commands the AIBO to back up, turn and walk in another direction. In this project, this case falls under Timeliner control, but it is a boundary case because in another situation, the case could be handled by the robot. For now, the case falls under the domain of Timeliner control because of the limitations of the programming currently present on the AIBO.

A case for which control lies in the hands of the human supervisor is one in which the AIBO manages to fall, thereby releasing all of its paw buttons. Timeliner detects if the AIBO is no longer supporting itself on its paws by testing to see if more than two paw buttons are being pushed in (by the ground). This simple test tells Timeliner to send a message to its human supervisor that the AIBO probably needs some maintenance. The MONITOR sequence in Figure 4-4 implements this test. Unfortunately, in this project, Timeliner is unable to reset the robot on its feet, but in another situation, Timeliner might be able to command another robot to push the AIBO back onto its feet.
4.2 Programming Architecture

This section discusses two perpendicular robot programming architectures and describes how to use the features of Timeliner’s UIL to create a supervisory control system.

4.2.1 Robotic Programming Concepts

A few basic necessities for robotic programming (Introduction to Robotics [3], p. 346) are that the language have:

- the ability to send **commands** necessary for directing the robot
- **flow control**, such as testing and branching, looping, subroutines and interrupts
- **parallel processing** ability, such as signal or wait
- ways to **monitor state** using some sort of sensor output

The Timeliner scripting language and executor have all of these capabilities, and manage to maintain simplicity for the user. The UIL has all of the necessary flow control constructs, and the executor can run multiple sequences in parallel. The executor also supports inter-sequence control; for example, sequence A can **START**, **STOP** or **RESUME** sequence B. (see Figure 4-4)

Another important aspect of robotic programming is the ability to gather environmental data through sensors. Two of the purposes of gathering such data, initiating and terminating motions, and choosing among alternative actions [14], imply two different scripting structures in Timeliner. The first is a structure in which the data is continuously monitored so that changes in the data activate Timeliner sequences. The second structure references the data in the CVT to determine how to continue. These
two structures are how Timeliner makes use of incoming sensor data that updates the CVT.

Sensor data is also used for modeling the state of the world or the environment of the robot. The programming structure that uses a model of the world to assist in planning the accomplishment of a task is called **Sense-Plan-Act**. Another robotic programming structure, **Subsumption**, focuses less on the model of the environment of the robot, and more on building up functionality of the robot with distinct behaviors.

### 4.2.2 The Sense-Plan-Act and Subsumption Architectures

The traditional horizontal levels of robotic control [2] are:

- **Perception** Data obtained from the sensors
- **Modeling** Model the data to create a concept of the world around the robot
- **Planning** Planning what the robot will do to achieve its goal(s).
- **Task Execution** Break down of the plan(s) into tasks for the robot.
- **Motor Control** Control of the motors to accomplish the desired tasks.

The motor control and the outside environment then cause the perception to change and affect the chain of events that lead to more motor control, and so on. Within the Timeliner-AIBO system, the Perception and Motor Control levels are handled by the AIBO, but the rest fall under Timeliner and Human control.

A *supervisory system* is a system that evaluates whether local controllers satisfy prespecified performance criteria, diagnoses causes for deviation from the performance criteria, plans actions, and executes the planned actions. [22]
BUNDLE WALK_AND_MONITOR
SEQUENCE MONITOR_ACTIVE
WHENEVER STATUS.LB_CONTACT +
    STATUS.LF_CONTACT +
    STATUS.RB_CONTACT +
    STATUS.RF_CONTACT < 3 THEN
    IF MONITOR.ESTOP_ON = 1 THEN
        COMMAND ESTOP_TOGGLE
    END IF
    MESSAGE "HELP! I’M FALLING!"
END WHENEVER
CLOSE SEQUENCE

SEQUENCE WALK_ACTIVE
COMMAND WALK, FR_SEQ_NUM => 0,
    ROBOT_ID => 0,
    LEGS_USED => 15,
    WALK_HEIGHT => 0.13/0.001,
    SPEED => 0.2/0.0001,
    NUM_STEPS => 12,
    LEG_LIFT_HEIGHT => 0.057/0.001,
    PFOOT => -0.04/0.002,
    AFOOT => 0.008/0.002,
    WALK_WIDTH => 0.005/0.001,
    GAIT_TYPE => 0,
    TURN_RATE => 0.0/0.001,
    FORCE_THRESH => 120/0.01,
    CRAB_ANGLE => 0.0,
    COMPLETION_THRESH => 0.0/0.0001
CLOSE SEQUENCE
CLOSE BUNDLE

Figure 4-4: This Timeliner script demonstrates one method of running two sequences
in parallel. The commands ESTOP_TOGGLE and WALK have been compiled into the GDB,
and the variables STATUS.LB_CONTACT, STATUS.LF_CONTACT, STATUS.RB_CONTACT,
STATUS.RF_CONTACT and MONITOR.ESTOP_ON have been compiled into the executor’s
CVT.
This information indicates that the job of Timeliner can be broken down into three levels of tasks: modeling or analyzing data, planning actions to achieve prespecified goals, and executing those actions. This sort of breakdown is known as a horizontal system, also called **Sense-Plan-Act (SPA)** [9], a deliberative approach which uses functional decomposition in a sequential manner.

The Timeliner UIL supports scripting to follow the SPA architecture, because it contains flow constructs that allow the Timeliner system to sense, to plan and to act. First, the UIL has access to the CVT, which should contain updated sensor data and other stored data about the environment. Values in the CVT can be tested and analyzed, which leads to planning. The scripts compiled and installed in Timeliner give the system a purpose, or task, as well as ways to accomplish that task. In an imaginary situation, if the task of the system was to find a red ball, the active sequences would tell the system how to classify and handle each situation it sensed. One classification would divide between seeing red (a high enough concentration of red pixels) and not seeing red (a too low concentration of red pixels). If the system sensed that it had seen red, it could perform the sequence `SAW.RED`, otherwise, it could perform the sequence for `SAW.NOR.RED`. The `SAW.RED` sequence would send a command to the robot to walk forward, which is the action part of SPA. An active sequence would monitor the presence of red in the robot’s vision, and if the robot went from seeing red to not seeing red, the plan and subsequent action would be revised to achieve the desired task. A very important classification for any task divides the states of having achieved the task and not.

Another system architecture proposed by Brooks [2] is **Subsumption**, which is based on a behavior breakdown instead of a functional breakdown. In this architecture, each level of behavior performs a complete task, and the most basic level of behavior can actually run by itself independently of the other levels. In general, subsumption does minimal modeling with the sensor data. This sort of programming
is considered to be vertical as opposed to the horizontal approach of SPA.

Scripting in the Timeliner UIL could also follow the Subsumption architecture. The basic task of walking or roaming could be set as the zeroth behavior layer in the Timeliner-AIBO system. Then, an additional behavioral layer could be added to the system, telling the robot to walk towards red objects. This layer would interrupt and take over the initial layer’s sequences using STOP and RESUME. This red object layer would monitor the sensor data and if vision indicated something red, the new layer would suspend the roaming/turning options in the initial walking layer, and force the robot to walk towards the red. The capability to execute sequences in parallel and control other sequences is especially useful when programming for subsumption.

For this project, the first step in demonstrating control over the robot is accomplished by closing the control loop, in which sensor data from the robot triggers a Timeliner sequence to send a command to the robot to perform an action in response to the sensor data. This first step is a simple reactive control which, once complete, will allow more complicated tasks to be programmed into the Timeliner system. It is these more complicated tasks, such as avoiding obstacles, that require special architecture like SPA or Subsumption.

One of the advantages of Timeliner is that the structure of the UIL is unbiased towards both of the discussed robotic programming architectures. In fact, as demonstrated, there are aspects of the language that make both possible. It would even be possible to combine both architectures. In one of the behavior layers, the scripts may take note of certain telemetry data and store information about the environment in the CVT. Another script might then suspend other behaviors and use that data to plan a trajectory for a specific task, only to restart the other behaviors when the task is complete.

Figure 4-4 is an example of running two sequences in parallel, one of the features that is useful for both of the discussed architectures. It would have also been possible
Figure 4-5: The Basic interaction scheme between the Timeliner executor and the AIBO.

These sequences perform two different types of tasks, so they could have also been modularized into two different bundles.

### 4.3 Interfacing Timeliner with the AIBO

Figure 4-5 details the idealized closed-loop communication between the Timeliner Executor and the AIBO, which is similar to Figure 4-3. There were two parts involved in allowing the data to flow both directions to close the loop. First, the Timeliner executor needed specifications to define the data exchange between Timeliner and the AIBO. Then, since some of the data formats were incompatible between the two systems, the Java AIBOChannel application acts as their translator.

#### 4.3.1 Specifying Timeliner

The three GDB definition files (see Figure 2-5) were the main configuration files necessary for customizing Timeliner to the AIBO.

`commands.txt` holds actual bytecode of commands Timeliner can send, over UDP only, to the target system. The commands are currently fixed at a size of 128 bytes each, and allow for modifiable parameters. Figure 4-6 is one entry in the commands file that defines a command that tells the AIBO to stop moving its specified leg(s).
[COMMAND: LEGSTOP]
[MACHINE_TYPE: SUN]
[SHELL]

---
BT1-8 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 70H;
BT9-16 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT17-24 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT25-32 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT33-40 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT41-48 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT49-56 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT57-64 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT65-72 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT73-80 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT81-88 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT89-96 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT97-104 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT105-112 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT113-120 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
BT121-128 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, 00H;
---

[PARAMETERS]
--- Name, Type, Arrayness, Byte Offset, Bit Location

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Arrayness</th>
<th>Byte Offset</th>
<th>Bit Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_SEQ_NUM</td>
<td>INTEGER</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ROBOT_ID</td>
<td>INTEGER</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>LEGS_USED</td>
<td>INTEGER</td>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

--- frame sequence # [0-65535], robot id [0-255], legs used [1-3F]

Figure 4-6: An example command definition of LEGSTOP for the commands.txt file. This definition tells Timeliner byte-by-byte what to send to the target system when the command is called from a sequence.
[CLASS: AIBO_BUTTONS]

[ATTRIBUTES]

<table>
<thead>
<tr>
<th>--Name,</th>
<th>Type,</th>
<th>Arrayness, Read/Write</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_BUTTONS,</td>
<td>LONG_INTEGER,</td>
<td>1,</td>
<td>READ_WRITE, INTEL;</td>
</tr>
<tr>
<td>LF_CONTACT,</td>
<td>FLOAT,</td>
<td>1,</td>
<td>READ_WRITE, INTEL;</td>
</tr>
<tr>
<td>RF_CONTACT,</td>
<td>FLOAT,</td>
<td>1,</td>
<td>READ_WRITE, INTEL;</td>
</tr>
<tr>
<td>LB_CONTACT,</td>
<td>FLOAT,</td>
<td>1,</td>
<td>READ_WRITE, INTEL;</td>
</tr>
<tr>
<td>RB_CONTACT,</td>
<td>FLOAT,</td>
<td>1,</td>
<td>READ_WRITE, INTEL;</td>
</tr>
<tr>
<td>CHIN_BUTTON,</td>
<td>FLOAT,</td>
<td>1,</td>
<td>READ_WRITE, INTEL;</td>
</tr>
<tr>
<td>HEAD_BUTTON,</td>
<td>FLOAT,</td>
<td>1,</td>
<td>READ_WRITE, INTEL;</td>
</tr>
<tr>
<td>FB_BUTTON,</td>
<td>FLOAT,</td>
<td>1,</td>
<td>READ_WRITE, INTEL;</td>
</tr>
<tr>
<td>MB_BUTTON,</td>
<td>FLOAT,</td>
<td>1,</td>
<td>READ_WRITE, INTEL;</td>
</tr>
<tr>
<td>BB_BUTTON,</td>
<td>FLOAT,</td>
<td>1,</td>
<td>READ_WRITE, INTEL;</td>
</tr>
<tr>
<td>LAN_ENABLE SWITCH,</td>
<td>FLOAT,</td>
<td>1,</td>
<td>READ_WRITE, INTEL;</td>
</tr>
</tbody>
</table>

Figure 4-7: An example definition of datatype AIBO_BUTTONS for the attributes.txt file.

[INSTANCES]

<table>
<thead>
<tr>
<th>--Name,</th>
<th>Class,</th>
<th>Location,</th>
<th>Bit Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>(STATUS,</td>
<td>AIBO_BUTTONS,</td>
<td>00000001H,</td>
<td>0 )</td>
</tr>
<tr>
<td>(OTHER,</td>
<td>OTHER_TYPE,</td>
<td>+,</td>
<td>0 )</td>
</tr>
</tbody>
</table>

Figure 4-8: Example instance declarations for instances.txt which allocate space in the Timeliner CVT to the data types defined in attributes.txt.
The LEGSTOP command is defined for one of the Draper leg motion algorithms. This command definition uses the first two bytes to define how many bytes are in the entire command (0CH = 12), the next two are overwritten by the FR_SEQ_NUM parameter value, the next two are overwritten by the ROBOT_ID parameter value, bytes 7-8 identify the command as one of the basic motion commands, and so forth.

Since the Draper algorithms accept UDP commands, this command can actually be sent directly to the AIBO without translation and the AIBO will perform the command. The translator, described in Section 4.3.2, is used for commands for which the AIBO requires a TCP connection.

Note that for commands sent from Timeliner to the AIBO, the translator is not needed for switching the byte order from little-endian to big-endian, because Timeliner automatically performs the switch if the machine type is defined as SUN in the commands file. For telemetry information sent from the AIBO to Timeliner, the translator is unnecessary as well because the data pre-formatted to be little-endian.

For the telemetry coming from the AIBO to Timeliner, attributes.txt and instances.txt define the data types and instances corresponding to state information from the target system. These two files define the contents of the executor's CVT. As currently defined, Timeliner can handle up to 1400 bytes in its CVT.

In Figure 4-7, example data type AIBO_BUTTONS is defined, and in Figure 4-8, AIBO_BUTTONS and an imaginary type named OTHER_TYPE are allocated space in the CVT. Note that an address of zero is not possible because the addresses in Timeliner are 1-based, so the instance of AIBO_BUTTONS must start at least at 1. (An address of 0 points to nothing.)

Timeliner also needs to know where to send its data, and where to receive data from the AIBO. In this case, the AIBOChannel buffer is where Timeliner sends its commands and where it gets data from. These specifications are described further in Appendix A.3.
4.3.2 AIBOChannel

The AIBOChannel application, Figure 4-9, functions as a translator for the incompatible languages of AIBO and the Timeliner executor. Translation is required in both directions of information flow.

Telemetry

The language translation analogy works quite well in this case. The telemetry packets sent by the AIBO assume that the receiver already knows what data is contained in the packets and the sizes of each piece of data. So, these packets contain only the bytes of data, without delimiters or other extraneous data. On the other hand, the Timeliner executor expects data that is formatted with the address for where in the CVT the piece of data belongs, the size of the piece of data (number of bytes) and the actual data. This allows Timeliner to parse data that is sent piecemeal, so that it is not necessary to send an update for the entire CVT in one packet. In the telemetry direction, the translation consisted of taking apart the AIBO’s packets and formatting them appropriately for Timeliner.

Commands

In the other direction, the AIBO is able to accept UDP and TCP connections, and in fact, some of the commands needed for turning and activating the emergency stop required TCP connections. However, Timeliner is only capable of sending UDP commands at the moment. So, the commands that need to be translated into TCP commands are intercepted and repackaged as TCP packets for the AIBO.

These TCP-only commands sometimes require menu-navigation in the AIBO (see Section 3.1.1), so AIBOChannel turns Timeliner’s single UDP commands into multiple TCP connections involving a sequence of commands.

An additional functionality of AIBOChannel makes use of the CVTMonitor (origi-
The job of the AIBOChannel application is to translate data that goes between the AIBO and the Timeliner executor.

Finally, created by Joe Bondi and Emily Braunstein), which displays the CVT values that are passing through to Timeliner in a simple window. This function was especially useful for debugging tasks, such as quickly checking that Timeliner or the AIBO was receiving data encoded in the correct endian.

**Timeliner’s Cycle Speed**

One of the bounds for Timeliner’s cycle speed is simply the speed at which data can be transferred between Timeliner and the robot. Over a wireless connection with the AIBO, a simple ping takes, on average, 4 milliseconds to complete a round trip, indicating a rough upper bound of 250 Hz in cycle speed. In order to test Timeliner with the AIBO, the original 1 Hz cycle speed was sped up to a 10 Hz speed.

This cycle speed limits how often Timeliner reads incoming telemetry. In order to avoid a buildup of outdated telemetry data, AIBOChannel moderates the 125 Hz flow of telemetry from the AIBO to match Timeliner’s 10 Hz by sending the most recently received packet of data every 100 milliseconds.
Chapter 5

Results and Discussion

5.1 Goal

The ultimate goal of the project was to demonstrate control over the AIBO by directing it around a table in such a way that the dog did not fall over the edge of the table. The subgoals of the project were to direct the AIBO to:

- Stand alone,

- Walk,

- Walk and stop after \( n \) steps,

- Walk and then turn,

- Walk until an event is sensed, then turn,

- Walk in a path intersected by a wall, detect the wall, turn and try avoid hitting it.
5.2 Experimentation

As mentioned in the previous chapter, the Timeliner UIL can be used to develop either one of the SPA or Subsumption programming architectures. Knowing this, however, the control implemented in this project is a simple reactive control, which is simpler than either of the two programming architectures. It does not depend upon a model of the environment, which is similar to the structure of subsumption, but it is only a single layer of reactivity, while subsumption generally indicates many layers building upon previous layers of behavior. This reactive control simply demonstrated closing the loop around the robot’s sensors and Timeliner’s sequencing capability. Each of the subgoals was satisfied by the simple reactive control, but the available sensor data was insufficient for accomplishing the main goal of walking the robot around a table and preventing it from falling off.

The first few subgoals involved simply sending commands from Timeliner to the robot. The scripts for those commands are in Appendix A.4.

The last subgoal, that of detecting a wall in the path of the robot and evading it, involved processing sensor data. The sensor monitoring sequences are kept in the MONITOR bundle, and all of its sequences are set to active by default. The BASIC_ACTIONS bundle contains sequences for basic actions that can be performed, such as walking forward, turning or toggling the emergency stop. Tasks (such as the subgoals) have their own bundles, so for example, the task of walking forward until a wall is detected is contained in the DETECT_WALL bundle.

The action sequences for controlling the Tekkotsu walk (namely TK_WALK) are of note, because they use variables in the CVT as parameters to define the walk command. The parallel sequence ambiguity demonstrated in Section 2.2.1 applies in this case, because multiple sequences in different bundles access and modify the parameter variables, FORWARD, STRAFE and ROTATE.

In general, robotic programming also consists of modeling the state of the world,
for the environment of the robot. For the purposes of this project, the simplest type of intelligence which allowed the AIBO to just walk around was used. Correspondingly, practically no world state was recorded. In order build up a model of the world within Timeliner, details about the AIBO's environment should be stored in the CVT variables, or even as bundle variables in one of the continuously active bundles. The disadvantage with bundle variables is that their scope is limited to a single bundle, so the CVT variables are more convenient for storing information that affects actions in many different bundles. If the need for data storage exceeds the default 1400 bytes that Timeliner sets aside for the CVT, Timeliner itself must be recompiled to allow for a larger CVT. That restriction aside, the CVT can store any type of information the user can create in the attributes and instances GDB files. The attraction of storing a model of the world is that previously identified obstacles or boundaries are recorded and can be taken into account while determining future trajectories. One of the challenges in modeling the robot's environment is determining how to retrieve and represent such data. This project does not address that question, and it may even be preferable to leave such computations to an external application like AIBOChannel.

For achieving the goal of detecting obstacles (and holes) in the robot's path, two abstractions of the goal were achieved. The first was simply the detection of an event (human input) to activate evasion movement, the second was to sense related data (the presence of an obstacle directly ahead of the AIBO) to activate evasion movement. The \texttt{DETECT\_EVENT} and \texttt{DETECT\_WALL} bundles in Appendix A.4 are very similar, which implies that once an algorithm for obstacle detection is implemented, making use of that information should be pretty straightforward.

5.2.1 \textbf{AIBO Does a High Five}

One of the preliminary tests of Timeliner in the control loop with the AIBO involved a high five. Timeliner commanded the AIBO to raise its two front paws. Sequence
MAIN contains a loop to test both buttons on the front paws so that when a button is pressed, the corresponding paw is commanded to descend, indicating a completed high five.

BUNDLE HIFIVE

SEQUENCE MAIN
  DECLARE L_TOUCHED BOOLEAN
  DECLARE R_TOUCHED BOOLEAN
  SET L_TOUCHED TO FALSE
  SET R_TOUCHED TO FALSE
  CALL PAWS_UP
  EVERY 1 BEFORE L_TOUCHED AND R_TOUCHED
    IF STATUS.LF_CONTACT = 1 THEN
      SET L_TOUCHED TO TRUE
      CALL LEFTDOWN
    END IF
    IF STATUS.RF_CONTACT = 1 THEN
      SET R_TOUCHED TO TRUE
      CALL RIGHTDOWN
    END IF
  END EVERY
  MESSAGE "TASK FINISHED"
CLOSE SEQUENCE

SUBSEQUENCE PAWS_UP
  COMMAND LEGLINEAR, FR_SEQ_NUM => 0,
    ROBOT_ID => 0,
    LEGS_USED => 12,
    FOOT_X => 0.01/0.0001,
    FOOT_Y => -0.05/0.0001,
    FOOT_Z => -0.04/0.0001,
    SPEED => 0.1/0.0001,
    FORCE_THRESH => 120
CLOSE SUBSEQUENCE

SUBSEQUENCE LEFTDOWN
  COMMAND LEGLINEAR, FR_SEQ_NUM => 0,
    ROBOT_ID => 0,
    LEGS_USED => 8,
    FOOT_X => 0.01/0.0001,
    FOOT_Y => -0.02/0.0001,
CLOSE SUBSEQUENCE

SUBSEQUENCE RIGHTDOWN
	COMMAND LEGLINEAR, FR_SEQ_NUM => 0,
	ROBOT_ID => 0,
	LEGS_USED => 4,
	FOOT_X => 0.01/0.0001,
	FOOT_Y => -0.02/0.0001,
	FOOT_Z => 0.12/0.0001,
	SPEED => 0.1/0.0001,
	FORCE_THRESH => 120

CLOSE SUBSEQUENCE

CLOSE BUNDLE

The subsequences PAWS_UP, LEFTDOWN and RIGHTDOWN use the Draper LEGLINEAR command to move the AIBO’s paws to the \((x, y, z)\) position indicated by the command parameters. Note that the values for FOOT_X, FOOT_Y, FOOT_Z and SPEED are divided values.\(^1\)

During this entire test, the AIBO rested on a stand which allowed the robot to move its paws without the danger of falling forward onto its face.

Appendix A.4 contains the scripts for more complicated tests with the AIBO.

5.3 Results

The resulting Timeliner-AIBO system accomplishes all of the above subgoals, but not the main goal. Instead, Timeliner was demonstrated to work for some obstacle avoidance using the IR sensors to trigger avoidance maneuvers or human-input to trigger action on the robot. It should, however, be possible to design a more complete obstacle detection and avoidance scheme by combining the vision (camera) and IR

---

\(^1\)The reason for this is that the commands on the AIBO have been written to understand those parameters in units of 0.0001 meters. The value 0.01/0.0001 then represents 0.01 m or 1 cm.
data in an obstacle detection algorithm (Fasola, Rybski and Veloso [7] offer one option that was developed using an older version of the AIBO with only a single IR distance detector) to determine where the AIBO should not go. Even with such an algorithm, however, the possibility of failure requires that a backup method, such as checking pressure and positions of joints, will always be necessary.

Although the current system's 'obstacle detection' could be improved (especially for holes in the ground!), its rudimentary obstacle detection and reaction is functional. It is important to note that Timeliner was made for responding to data, and not for running complicated algorithms to analyze data. So, instead of implementing a complicated obstacle detection algorithm in the Timeliner UIL, it is perhaps better to wrap Timeliner in an application like AIBOChannel that will perform the analysis and send Timeliner the results. The definition of supervisory control given by Sheridan (Section 4.1.1) mentions that the intelligence of the subordinate affects how much work the supervisor has to do. In this case, either the subordinate or the supervisor can be 'enhanced' by AIBOChannel to accomplish complicated data analysis, but having Timeliner itself run computations better suited for Matlab is probably not the best idea.

The AIBO is also a remarkably flexible robot, which adds difficulty to the task of programming for it. For example, preprogramming a map of the terrain into Timeliner seems a waste of potential for a robot equipped with both IR distance sensors and a camera. Instead, the use of those sensors and camera allows for travel on a changing terrain, or at least saves the programmer from hardcoding a map in Timeliner. Using Timeliner with simpler robots and for simpler tasks will be easier than commanding the AIBO to detect obstacles or potential falls and to avoid them.
5.3.1 Advantages and Disadvantages of Timeliner

Timeliner is built with some advantages that make it an attractive robotic control system.

- The Timeliner scripting language is easy to understand and automatically handles parallel sequences or tasks. In fact, the language itself encourages modularized programming through separation of different tasks into bundles and sequences.

- The command and telemetry setup is simple to configure, it is functional, and it works. The user or operator does not need to program lower level details, such as which sockets lead where and package what data, and can instead focus on the tasks he would like the robot to accomplish.

- The Timeliner system is built to include the human operator in the control of the system by using messages to notify its users. Such interaction automatically makes the system perfect as a robot-to-human interface.

The disadvantages of using Timeliner result from the fact that the very same structure that provides some advantages is somewhat inflexible. The command and telemetry functions, which are so simple to use because they are already set up, are limited to UDP. If the user needs to send Timeliner commands to multiple UDP ports, Timeliner has to be modified to provide this flexibility, because it currently sends commands to only one location.

5.3.2 Cycle Speed

Timeliner functioned reliably at 10 Hz. Given the supervisory role of Timeliner, matching its speed to keep up with the AIBO’s telemetry output speed of 31 Hz is probably unnecessary. In fact, there seemed to be a bottleneck within AIBOChannel
(probably due to the multiple connections it had to make with the robot) that caused
data to transmit at an amazing top speed of 5 Hz.

5.3.3 System Synchronization

Related to the topic of cycle speed is the one which arises when actions triggered by
commands sent by Timeliner take a long time to finish. Since the commands sent by
Timeliner are only related to the performance of the command by the transmission of
a single UDP packet, a successful completion of the command statement or even the
finished state of the sequence gives no indication as to whether the robot has finished
performing the requested action.

For example, the Draper walk command on the AIBO may be configured to tell
the AIBO to walk twenty steps with a single command. If a sequence contains only
this command to walk twenty steps, the sequence will send the command and finish
before the command finishes on the robot.

This synchronization is a common issue between two communicating systems
which should be addressed with an appropriate interface that gives each system ac-
access to the status of the other system. For Timeliner to test if the robot has finished
performing commands, instead of testing the state of the commanding sequence, it is
necessary to either test the state of the robot using telemetry, or have the robot send
data back indicating what commands it has finished. In its current setup, the robot
does not send this sort of data directly, but it should.

5.3.4 Robot programming made simpler?

An interesting attempt was made to render robot programming more accessible to
people without programming backgrounds in PROGO [5]. Doty and Janz mention
that they had to make tradeoffs between the functionality of the language (C-based)
and usability.
Lozano-Pérez [14] notes that functionality should not be sacrificed to make the system easier to use for less-experienced operators. Instead, ideally, “robot programming languages should support the functional requirements of its most sophisticated users. The sophisticated users can implement special-purpose interfaces, in the language itself, for the less experienced users.” Using Timeliner allows the sophisticated user to program a simple interface that matches Timeliner’s predetermined interface, and allows the less experienced users to then script actions on the AIBO using the Timeliner UIL.

In the end, Timeliner functions as that special-purpose interface that can be used by less experienced users. Timeliner builds upon the “more sophisticated” robot programming that uses Sony’s Open-R SDK. There are other projects that try to make Open-R more accessible. Tekkotsu is one such project, as well as another called URBI [18]. Timeliner can function well as an interface that hides some of the machinery from the user who simply wants to be able to tell the AIBO to walk around, take pictures and store them in the system.

5.3.5 TIDE

The Timeliner Integrated Development Environment integrates the components of Timeliner, normally used as separate executables, into a single application. The GDB compilation, script writing, compilation and eventual execution of bundles and sequences are all tied together in TIDE. The application itself is an Eclipse [6] plug-in, so it takes advantage of many features of the Eclipse IDE.

TIDE proved to be a very useful tool, especially because it streamlined some aspects of working with Timeliner by integrating the many different components. For those familiar with Timeliner, TIDE is especially easy to pick up and understand. It is, however, less intuitive for a new user with no background in Timeliner. When developing for the robot, changes were often made to the GDB definitions, which
required rerunning **gdb.exe** on a regular basis. Editing the GDB files and compiling the database is not as well integrated into the flow of the application as the other components of Timeliner are. A tool that would be very useful in TIDE would be a GDB Editor that generated the GDB definition files in the appropriate formats for the system. This tool would allow the user to avoid editing those files by hand.

### 5.4 Lessons Learned

This section relates some aspects of the project or features of Timeliner that in hindsight might have made the project better. Since Timeliner was not developed with robotic control in mind, there are some desired features which, while not requiring major changes in the Timeliner structure itself, indicate the need for some added flexibility in Timeliner.

#### 5.4.1 Programming directly on the AIBO

Realistically, the robot’s interface should be modified to function as seamlessly with Timeliner as possible. Although it was possible to control some actions on the AIBO without AIBOChannel, the translator application was still necessary in order for Timeliner to understand the AIBO’s telemetry packets. The introduction of a translation application in between the AIBO and Timeliner added time and complications to each data transmission, which could have been avoided if modifications were made directly to the programming on the AIBO. Ideally, the AIBO would be programmed to send telemetry in the format expected by Timeliner, as well as to a pre-determined port. The AIBO would also accept and carry out UDP commands sent to a single port.
5.4.2 Cycle speed

Currently, Timeliner’s cycle speed is 1 Hz. In the robotic world, response speed on this time scale is too slow, especially if the robot is in a situation that will do damage to the robot unless immediate action is taken. For the project, Timeliner was recompiled to run at 10 Hz. If Timeliner were to include the capability of changing the cycle speed of the executor without requiring recompilation, such flexibility may come in useful for adapting Timeliner to systems that require faster or slower response times.

The cycle speed also limits the speed at which sequences may be started from within the same bundle. This is due to the smaller size of the internal bundle command buffer versus the external bundle command buffer. As a result, sequences are expected to be started more often externally (from a different bundle) than internally. This expectation may not be reasonable, though. For consistency, Timeliner’s bundle command buffers could be the same size, or the internal and external buffers could be shared.

5.4.3 UDP vs. TCP

UDP and TCP are two protocols designed for different types of communication, and it would be useful to be able to use both protocols depending on the type of communication desired. In general, UDP is used for time sensitive data such as video or radio feeds for which the receipt of every packet is not necessary. Timeliner is already capable of sending and receiving UDP packets. TCP is a more structured protocol, in which there is a client and a server. All packets sent are guaranteed to be received eventually, as long as both sides of the connection remain open. This protocol would be useful for commanding robots, because sending three commands to a robot which only receives the last could cause unnecessary complications.
5.4.4 Robot Simulations

In hindsight, it would have been nice to experiment with an AIBO simulation first, to save the actual robot from any damage, and perhaps save time on little things such as setting up or starting up the robot. It turns out that an AIBO Simulation is available by Webots [21], which incorporates the URBI API [18]. However, no matter the accuracy, current simulations cannot account for all of the real-time or real-life problems that occur, such as locked joints or slight instabilities. Simulations are useful for preliminary tests, but are by no means a substitution for real-world experimentation.
Chapter 6

Future Work

6.1 The Robot as a Component

One of the most compelling reasons for following through with this project is that it takes on the challenge of interfacing a robot with a much larger system. Take, for example, a smart room which employs a robot to clean and pick up objects. If the room is equipped with cameras as well, a system may use those cameras in addition to the visual feedback from the robot in order to plan out the best path for the robot to follow. Such a system is interesting because it allows the robot to simultaneously function as an independent unit and as part of a larger, smarter system.

Although the scope of the project did not include testing the control and use of the robot within a larger Timeliner system, this would be a very interesting experiment for large, existing Timeliner systems.

6.2 Minimizing Timeliner

Another interesting project is that of compiling Timeliner to run directly on the robot. The AIBO itself has a 576MHz MIPS R7000 processor and 64 MB RAM. This should be sufficient memory to run Timeliner, but the Timeliner executables,
which are compiled in Ada, would have to be recompiled to function with the possibly limited instructions of the processor. The Timeliner executor, newly compiled to run at speeds on the order of MHz, would then run on top of the Open-R SDK or replace it entirely, and the operator would upload scripts wirelessly.

6.3 Handling Sensor Malfunction

This project did not address the issue of sensor malfunction, but it is an important consideration when dealing with sensors. The most important part of handling sensor malfunction is detecting it, then perhaps making the programs and scripts robust enough to work despite some faults. One commonly used method for detecting sensor malfunction is a disagreement in consensus of multiple sensors. Such a method could be designed to work in another robot, but for the AIBO, each sensor is quite different from the others, so testing consensus is difficult. Ensuring that sensor malfunction is handled is an important aspect of robot programming, though, because it is a problem that is bound to occur at some point. Luckily, Timeliner monitoring scripts can model the possible variations of a sensor and message the user if the sensor is acting oddly, but this type of monitoring does not cover the full range of sensor malfunction. When designing any Timeliner robotic control system, this will be an important aspect to cover.
Appendix A

Timeliner System Details

A.1 Thorough Timeliner Overview

Figure A-1 show how information is transmitted and stored throughout Timeliner. The rest of this paragraph explains Figure A-1 in detail. *gdb_input.script* is a configuration file which tells *gdb.exe* where to find the ground database definition, composed of *commands.txt*, *attributes.txt* and *instances.txt*. *gdb_pui_address_file.txt* simply details what information is located where in the binary DB once the GDB has been compiled. Each command and each instance from *instances.txt* has its own pui address. *.*.tix* is a Timeliner executable output by the Timeliner compiler (*tlcomp.exe*). The Timeliner executor then runs the executables, which can send commands to the Target System and process data sent from the Target System to Timeliner.

A.2 GDB Definition Documents

The Ground Database used in Timeliner contains some information about the system controlled by Timeliner. The database is defined by three configuration files, *commands.txt*, *attributes.txt* and *instances.txt*, which are described in the fol-
Figure A-1: A diagram depicting Timeliner data flow throughout its components and the target system.
A.2.1 Commands

The commands.txt file in the ground database allows the user to configure commands byte by byte to be sent to the receiver (the robot). The commands are hard-coded to be 128 bytes long, so if your commands must be longer or shorter, the Timeliner code itself must be changed. However, if the robot can be made to ignore extra zeros in incoming commands, there is no need to change Timeliner. This is the commands.txt actually used for working with the AIBO.

Note: Mind the bytes offset - for command parameters, the description switches to 0-indexed instead of 1-indexed, because it is an offset, not an address.

```
[COMMAND: RESTART_WALK]
[MACHINE_TYPE: SUN]
[SHELL]

<table>
<thead>
<tr>
<th>Byte Range</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT1-8</td>
<td>0OH, 0DH, OOH, OOH, OOH, OOH, OOH, AOH;</td>
</tr>
<tr>
<td>BT9-16</td>
<td>02H, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT17-24</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT25-32</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT33-40</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT41-48</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT49-56</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT57-64</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT65-72</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT73-80</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT81-88</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT89-96</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT97-104</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT105-112</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT113-120</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
<tr>
<td>BT121-128</td>
<td>0OH, 0OH, OOH, OOH, OOH, OOH, OOH, OOH;</td>
</tr>
</tbody>
</table>
```

[PARAMETERS]
-- Name, Type, Arrayness, Byte Offset, Bit Location

69
[COMMAND: CANCEL_WALK]
[MACHINE_TYPE: SUN]
[SHELL]

BT1-8  => OOH, ODH, OOH, OOH, OOH, OOH, AOH;
BT9-16 => O1H, OOH, OOH, OOH, OOH, OOH, OOH;
BT17-24 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT25-32 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT33-40 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT41-48 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT49-56 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT57-64 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT65-72 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT73-80 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT81-88 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT89-96 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT97-104 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT105-112 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT113-120 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT121-128 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;

[PARAMETERS]
--Name, Type, Arrayness, Byte Offset, Bit Location

[COMMAND: ESTOP_TOGGLE]
[MACHINE_TYPE: SUN]
[SHELL]

BT1-8  => OOH, OOH, OOH, OOH, OOH, OOH, OOH, AOH;
BT9-16 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT17-24 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT25-32 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT33-40 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT41-48 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT49-56 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT57-64 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT65-72 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT73-80 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT81-88 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT89-96 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT97-104 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT105-112 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT113-120 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT121-128 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT57-64 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT65-72 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT73-80 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT81-88 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT89-96 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT97-104 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT105-112 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT113-120 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT121-128 => OOH, OOH, OOH, OOH, OOH, OOH, OOH;

--- Name, Type, Arrayness, Byte Offset, Bit Location

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Arrayness</th>
<th>Byte Offset</th>
<th>Bit Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_SEQNUM</td>
<td>INTEGER</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ROBOT_ID</td>
<td>INTEGER</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

[COMMAND: ROTATE]
[MACHINE_TYPE: SUN]

[SHELL]
TASK_VALUE, FLOAT, 1, 9, 0;
----------------------------------

[COMMAND: FORWARD]
[MACHINE_TYPE: SUN]
[SHELL]

BT1-8 => OOH, ODH, OOH, OOH, OOH, OOH, OOH, AOH;
BT9-16 => 66H, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT17-24 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT25-32 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT33-40 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT41-48 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT49-56 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT57-64 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT65-72 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT73-80 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT81-88 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT89-96 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT97-104 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT105-112 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT113-120 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT121-128 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;

[PARAMETERS]

<table>
<thead>
<tr>
<th>Name,</th>
<th>Type,</th>
<th>Arrayness, Byte Offset, Bit Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- Name,</td>
<td>INTEGER,</td>
<td>1, 2, 0;</td>
</tr>
<tr>
<td>ROBOT_ID,</td>
<td>INTEGER,</td>
<td>1, 4, 0;</td>
</tr>
<tr>
<td>TASK_VALUE,</td>
<td>FLOAT,</td>
<td>1, 9, 0;</td>
</tr>
</tbody>
</table>

[COMMAND: STRAFE]
[MACHINE_TYPE: SUN]
[SHELL]

BT1-8 => OOH, ODH, OOH, OOH, OOH, OOH, OOH, AOH;
BT9-16 => 73H, OOH, OOH, OOH, OOH, OOH, OOH, OOH;

72
BT17-24 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT25-32 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT33-40 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT41-48 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT49-56 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT57-64 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT65-72 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT73-80 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT81-88 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT89-96 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT97-104 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT105-112 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT113-120 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT121-128 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;

[PARAMETERS]

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Arrayness</th>
<th>Byte Offset</th>
<th>Bit Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_SEQ_NUM</td>
<td>INTEGER</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ROBOT_ID</td>
<td>INTEGER</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>TASK_VALUE</td>
<td>FLOAT</td>
<td>1</td>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>

[COMMAND: LEGSTOP]
[MACHINE_TYPE: SUN]
[SHELL]

BT1-8 => 00H, OCH, OOH, OOH, OOH, OOH, 00H, 70H;
BT9-16 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT17-24 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, 00H;
BT25-32 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, OOH;
BT33-40 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, OOH;
BT41-48 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, OOH;
BT49-56 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, OOH;
BT57-64 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, OOH;
BT65-72 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, OOH;
BT73-80 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, OOH;
BT81-88 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, OOH;
BT89-96 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, OOH;
BT97-104 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, OOH;
BT105-112 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, OOH;
BT113-120 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, OOH;
BT121-128 => 00H, OOH, OOH, OOH, OOH, OOH, 00H, OOH;
<table>
<thead>
<tr>
<th>Command</th>
<th>Type</th>
<th>Arrayness</th>
<th>Byte Offset</th>
<th>Bit Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_SEQ_NUM</td>
<td>INTEGER</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ROBOT_ID</td>
<td>INTEGER</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>LEGS_USED</td>
<td>INTEGER</td>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**COMMAND: LEGARC**

**MACHINE_TYPE: SUN**

**SHELL**

---

<table>
<thead>
<tr>
<th>Arrayness</th>
<th>Byte Offset</th>
<th>Bit Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

---

**PARAMETERS**

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Arrayness</th>
<th>Byte Offset</th>
<th>Bit Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_SEQ_NUM</td>
<td>INTEGER</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ROBOT_ID</td>
<td>INTEGER</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>LEGS_USED</td>
<td>INTEGER</td>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>ROTATION_AXIS</td>
<td>INTEGER</td>
<td>1</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>ARC_X</td>
<td>INTEGER</td>
<td>1</td>
<td>14</td>
<td>0</td>
</tr>
</tbody>
</table>
ARC_Y, INTEGER, 1, 16, 0;
ARC_Z, INTEGER, 1, 18, 0;
ARC_ANGLE, INTEGER, 1, 20, 0;
TURN_RATE, INTEGER, 1, 22, 0;
FORCE_THRESH, INTEGER, 1, 24, 0;

-- NOTE: ARC_X AND ON ARE STILL TBD IN SIMULATION LOP

[COMMAND: LEGQUADRATIC]
[MACHINE_TYPE: SUN]
[SHELL]

BT1-8 => OOH, 18H, OOH, OOH, OOH, OOH, OOH, 70H;
BT9-16 => OOH, 02H, OOH, OOH, OOH, OOH, OOH, OOH;
BT17-24 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT25-32 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT33-40 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT41-48 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT49-56 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT57-64 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT65-72 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT73-80 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT81-88 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT89-96 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT97-104 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT105-112 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT113-120 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;
BT121-128 => OOH, OOH, OOH, OOH, OOH, OOH, OOH, OOH;

[PARAMETERS]

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Arrayness</th>
<th>Byte Offset</th>
<th>Bit Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_SEQ_NUM</td>
<td>INTEGER</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ROBOT_ID</td>
<td>INTEGER</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>LEGS_USED</td>
<td>INTEGER</td>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>FOOT_X</td>
<td>INTEGER</td>
<td>1</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>FOOT_Y</td>
<td>INTEGER</td>
<td>1</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>FOOT_Z</td>
<td>INTEGER</td>
<td>1</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>LIFT_HEIGHT</td>
<td>INTEGER</td>
<td>1</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>SPEED</td>
<td>NATURAL</td>
<td>1</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>
FORCE_THRESH, INTEGER, 1, 22, 0;

[COMMAND: LEGLINEAR]
[MACHINE_TYPE: SUN]

BT1-8  => 00H, 16H,  OOH,  OOH,  OOH,  OOH,  OOH,  70H;
BT9-16 => 00H, 03H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT17-24 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT25-32 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT33-40 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT41-48 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT49-56 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT57-64 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT65-72 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT73-80 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT81-88 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT89-96 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT97-104 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT105-112 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT113-120 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;
BT121-128 => 00H, 00H,  OOH,  OOH,  OOH,  OOH,  OOH,  OOH;

[PARAMETERS]
-- Name,        Type,    Arrayness, Byte Offset, Bit Location
              --
FR_SEQ_NUM,   INTEGER, 1,        2,   0;
ROBOT_ID,     INTEGER, 1,        4,   0;
LEGS_USED,    INTEGER, 1,        10,  0;
FOOT_X,       INTEGER, 1,        12,  0;
FOOT_Y,       INTEGER, 1,        14,  0;
FOOT_Z,       INTEGER, 1,        16,  0;
SPEED,        INTEGER, 1,        18,  0;
FORCE_THRESH, INTEGER, 1,        20,  0;

[COMMAND: WALK]
[MACHINE_TYPE: SUN]
[SHELL]

BT1-8  =>  00H, 24H, 00H, 00H, 00H, 00H, 00H, 00H;
BT9-16 =>  00H, 01H, 00H, 00H, 00H, 00H, 00H, 00H;
BT17-24 =>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;
BT25-32 =>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;
BT33-40 =>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;
BT41-48 =>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;
BT49-56 =>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;
BT57-64 =>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;
BT65-72 =>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;
BT73-80 =>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;
BT81-88 =>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;
BT89-96 =>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;
BT97-104=>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;
BT105-112=>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;
BT113-120=>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;
BT121-128=>  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H;

[PARAMETERS]

--Name, Type, Arrayness, Byte Offset, Bit Location

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Arrayness</th>
<th>Byte Offset</th>
<th>Bit Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_SEQ_NUM</td>
<td>INTEGER</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ROBOT_ID</td>
<td>INTEGER</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>LEGS_USED</td>
<td>INTEGER</td>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>WALK_HEIGHT</td>
<td>INTEGER</td>
<td>1</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>SPEED</td>
<td>INTEGER</td>
<td>1</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>NUM_STEPS</td>
<td>INTEGER</td>
<td>1</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>LEG_LIFT_HEIGHT</td>
<td>INTEGER</td>
<td>1</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>FFOOT</td>
<td>INTEGER</td>
<td>1</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>AFOOT</td>
<td>INTEGER</td>
<td>1</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>WALK_WIDTH</td>
<td>INTEGER</td>
<td>1</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>GAIT_TYPE</td>
<td>INTEGER</td>
<td>1</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>TURN_RATE</td>
<td>INTEGER</td>
<td>1</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td>FORCE_THRESH</td>
<td>INTEGER</td>
<td>1</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>CRAB_ANGLE</td>
<td>INTEGER</td>
<td>1</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>COMPLETION_THRESH</td>
<td>INTEGER</td>
<td>1</td>
<td>34</td>
<td>0</td>
</tr>
</tbody>
</table>
A.2.2 Attributes

The attributes.txt file in the ground database describes object-like constructions that may be used in the CVT of the Timeliner Executor. These constructions are similar to structs in C, consisting of single elements or arrays of the basic Timeliner datatypes.

The constructs defined in attributes.txt are referenced by the instances.txt file(s).

[CLASS: AIBO_REAL_TYPE]
[ATTRIBUTES]
--LONG_INTEGER MEANS INT32
--FLOAT MEANS SINGLE

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Arrayness</th>
<th>Read/Write</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMESTAMP</td>
<td>LONG_INTEGER</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>NUM_PID_JOINTS</td>
<td>LONG_INTEGER</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>LF_POS_ROTATOR</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>LF_POS_ELEVATOR</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>LF_POS_KNEE</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>RF_POS_ROTATOR</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>RF_POS_ELEVATOR</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>RF_POS_KNEE</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>LB_POS_ROTATOR</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>LB_POS_ELEVATOR</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>LB_POS_KNEE</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>RB_POS_ROTATOR</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>RB_POS_ELEVATOR</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>HEAD_NOD</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>TAIL_TILT</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>TAIL_PAN</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>HEAD_MOUTH</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
</tbody>
</table>

-- NUM_SENSORS SHOULD BE 11

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Arrayness</th>
<th>Read/Write</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_SENSORS</td>
<td>LONG_INTEGER</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
<tr>
<td>IR_NEAR</td>
<td>FLOAT</td>
<td>1</td>
<td>READ_WRITE</td>
<td>INTEL</td>
</tr>
</tbody>
</table>
IR_FAR,  FLOAT,  1,    READ_WRITE, INTEL;
IR_CHEST, FLOAT,  1,    READ_WRITE, INTEL;
ACCEL_BACK, FLOAT,  1,    READ_WRITE, INTEL;
ACCEL_LEFT, FLOAT,  1,    READ_WRITE, INTEL;
ACCEL_DOWN, FLOAT,  1,    READ_WRITE, INTEL;
POWER_REMAINING, FLOAT,  1,    READ_WRITE, INTEL;
POWER_TEMP, FLOAT,  1,    READ_WRITE, INTEL;
POWER_CAPACITY, FLOAT,  1,    READ_WRITE, INTEL;
POWER_VOLTAGE, FLOAT,  1,    READ_WRITE, INTEL;
POWER_CURRENT, FLOAT,  1,    READ_WRITE, INTEL;
-- NUM_BUTTONS SHOULD BE 10
NUM_BUTTONS, LONG_INTEGER, 1, READ_WRITE, INTEL;
LF_CONTACT, FLOAT,  1,    READ_WRITE, INTEL;
RF_CONTACT, FLOAT,  1,    READ_WRITE, INTEL;
LB_CONTACT, FLOAT,  1,    READ_WRITE, INTEL;
RB_CONTACT, FLOAT,  1,    READ_WRITE, INTEL;
CHIN_BUTTON, FLOAT,  1,    READ_WRITE, INTEL;
HEAD_BUTTON, FLOAT,  1,    READ_WRITE, INTEL;
FB_BUTTON, FLOAT,  1,    READ_WRITE, INTEL;
MB_BUTTON, FLOAT,  1,    READ_WRITE, INTEL;
BB_BUTTON, FLOAT,  1,    READ_WRITE, INTEL;
LAN_ENABLE_SWITCH, FLOAT,  1,    READ_WRITE, INTEL;
LF_DUTY_ROTATOR, FLOAT,  1,    READ_WRITE, INTEL;
LF_DUTY_ELEVATOR, FLOAT,  1,    READ_WRITE, INTEL;
LF_DUTY_KNEE, FLOAT,  1,    READ_WRITE, INTEL;
RF_DUTY_ROTATOR, FLOAT,  1,    READ_WRITE, INTEL;
RF_DUTY_ELEVATOR, FLOAT,  1,    READ_WRITE, INTEL;
RF_DUTY_KNEE, FLOAT,  1,    READ_WRITE, INTEL;
LB_DUTY_ROTATOR, FLOAT,  1,    READ_WRITE, INTEL;
LB_DUTY_ELEVATOR, FLOAT,  1,    READ_WRITE, INTEL;
LB_DUTY_KNEE, FLOAT,  1,    READ_WRITE, INTEL;
RB_DUTY_ROTATOR, FLOAT,  1,    READ_WRITE, INTEL;
RB_DUTY_ELEVATOR, FLOAT,  1,    READ_WRITE, INTEL;
RB_DUTY_KNEE, FLOAT,  1,    READ_WRITE, INTEL;
HEAD_TILT_DUTY, FLOAT,  1,    READ_WRITE, INTEL;
HEAD_PAN_DUTY, FLOAT,  1,    READ_WRITE, INTEL;
HEAD_NOD_DUTY, FLOAT,  1,    READ_WRITE, INTEL;
TAIL_TILT_DUTY, FLOAT,  1,    READ_WRITE, INTEL;
TAIL_PAN_DUTY, FLOAT,  1,    READ_WRITE, INTEL;
HEAD_MOUTH_DUTY, FLOAT,  1,    READ_WRITE, INTEL;

[CLASS: AIBO_MONITOR]
[ATTRIBUTES]
--- Name,  Type,  Arrayness, Read/Write,  Machine
----------------------------------------------------------------------------------------
CONNECTED,  SHORT_INTEGER,  1,  READ_WRITE,  INTEL;
ESTOP_ON,   SHORT_INTEGER,  1,  READ_WRITE,  INTEL;
----------------------------------------------------------------------------------------

CLASS:  SEQ_PARAMETERS
ATTRIBUTES
----------------------------------------------------------------------------------------
--- Name,  Type,  Arrayness, Read/Write,  Machine
----------------------------------------------------------------------------------------
FORWARD,   FLOAT,  1,  READ_WRITE,  INTEL;
STRAFE,    FLOAT,  1,  READ_WRITE,  INTEL;
ROTATE,    FLOAT,  1,  READ_WRITE,  INTEL;
----------------------------------------------------------------------------------------

A.2.3 Instances

The instances.txt file in the ground database defines what actually exists in the Timeliner Executor's CVT. It references the constructs in the attributes.txt file, and can create multiple instances of each construct. The CVT is limited to 1400 bytes, but this limit can be changed by changing the Timeliner code itself.

Note: There are actually two instances.txt files required to define the ground database: one for the Test Environment (instances_test.txt) and one for the Real Environment (instances_real.txt). This was useful, for example, for having one CVT work with the AIBO simulation, and the other work with the actual robot. Since I ended up developing without a simulation, I used instances_real.txt.

[INSTANCES]
----------------------------------------------------------------------------------------
--- Name,  Class,  Location,  Bit Offset
----------------------------------------------------------------------------------------
(STATUS,  AIBO_REAL_TYPE,  00000001H,  0  )
(MONITOR, AIBO_MONITOR,  +,  0  )
(PARAMS,  SEQ_PARAMETERS,  +,  0  )
----------------------------------------------------------------------------------------
A.3 Defining Ports

The second line of the configuration file `tinet-ccsds-config.txt` (found in the Timeliner executables folder) tells Timeliner the destination IP address and port number to send its commands to. If it was only necessary to send UDP commands to the AIBO, we could set the outgoing address to the AIBO’s IP (192.168.0.101 in this case) and command port (20051) and bypass the translation application entirely.

A.4 Timeliner Scripts

Some of the subgoals of the project are accomplished in the following file, `positions.tls`, which contains sequences that command the robot to stand still or to walk using the Draper algorithm.

A.4.1 positions.tls

BUNDLE POSITIONS

SEQUENCE STAND
   COMMAND LEGLINEAR, FR_SEQ_NUM => 0,
   ROBOT_ID => 0,
   LEGS_USED => 12,
   FOOT_X => 0.01/0.0001,
   FOOT_Y => -0.02/0.0001,
   FOOT_Z => 0.12/0.0001,
   SPEED => 0.1/0.0001,
   FORCE_THRESH => 120

COMMAND LEGLINEAR, FR_SEQ_NUM => 0,
   ROBOT_ID => 0,
   LEGS_USED => 3,
   FOOT_X => 0.01/0.0001,
   FOOT_Y => 0.02/0.0001,
   FOOT_Z => 0.115/0.0001,
   SPEED => 0.1/0.0001,
   FORCE_THRESH => 120

CLOSE SEQUENCE
SEQUENCE DRAPER_WALK
COMMAND WALK, FR_SEQ_NUM => 0,
   ROBOT_ID => 0,
   LEGS_USED => 15,
   WALK_HEIGHT => 0.13/0.001,
   SPEED => 0.1/0.0001,
   NUM_STEPS => 4,
   LEG_LIFT_HEIGHT => 0.05/0.001,
   FFoot => -0.05/0.002,
   AFoot => 0.01/0.002,
   WALK_WIDTH => 0.00/0.001,
   GAIT_TYPE => 0,
   TURN_RATE => 0.0/0.001,
   FORCE_THRESH => 120/0.01,
   CRAB_ANGLE => 0.0,
   COMPLETION_THRESH => 0.01/0.0001
CLOSE SEQUENCE

SEQUENCE TK_WALK
COMMAND FORWARD, FR_SEQ_NUM => 0,
   ROBOT_ID => 0,
   TASK_VALUE => 1.0
COMMAND STRAFE, FR_SEQ_NUM => 0,
   ROBOT_ID => 0,
   TASK_VALUE => 0
COMMAND ROTATE, FR_SEQ_NUM => 0,
   ROBOT_ID => 0,
   TASK_VALUE => 0
CLOSE SEQUENCE

SEQUENCE TK_WALK_BACK
COMMAND FORWARD, FR_SEQ_NUM => 0,
   ROBOT_ID => 0,
   TASK_VALUE => -1.0
CLOSE SEQUENCE

SEQUENCE ROTATE
EVERY 1 WITHIN 5
COMMAND FORWARD, FR_SEQ_NUM => 0,
   ROBOT_ID => 0,
   TASK_VALUE => 0
COMMAND STRAFE, FR_SEQ_NUM => 0,
   ROBOT_ID => 0,
   TASK_VALUE => 0
COMMAND ROTATE, FR_SEQ_NUM => 0,
   ROBOT_ID => 0,
   TASK_VALUE => 1.0

END EVERY
CLOSE SEQUENCE

SEQUENCE FORWARD
   EVERY 1 WITHIN 5
      COMMAND FORWARD, FR_SEQ_NUM => 0,
         ROBOT_ID => 0,
         TASK_VALUE => 1.0
      COMMAND STRAFE, FR_SEQ_NUM => 0,
         ROBOT_ID => 0,
         TASK_VALUE => 0
      COMMAND ROTATE, FR_SEQ_NUM => 0,
         ROBOT_ID => 0,
         TASK_VALUE => 0

END EVERY
CLOSE SEQUENCE

SEQUENCE STRAFE
   EVERY 1 WITHIN 5
      COMMAND FORWARD, FR_SEQ_NUM => 0,
         ROBOT_ID => 0,
         TASK_VALUE => 0
      COMMAND STRAFE, FR_SEQ_NUM => 0,
         ROBOT_ID => 0,
         TASK_VALUE => 1.0
      COMMAND ROTATE, FR_SEQ_NUM => 0,
         ROBOT_ID => 0,
         TASK_VALUE => 0

END EVERY
CLOSE SEQUENCE

SEQUENCE STOP_MOVEMENT
   STOP POSITIONS.FORWARD_5
   STOP POSITIONS.STRAFE_5
   STOP POSITIONS.ROTATE_5
      COMMAND FORWARD, FR_SEQ_NUM => 0,
         ROBOT_ID => 0,
         TASK_VALUE => 0.0
      COMMAND STRAFE, FR_SEQ_NUM => 0,
         ROBOT_ID => 0,
         TASK_VALUE => 0.0
      COMMAND ROTATE, FR_SEQ_NUM => 0,
CLOSE SEQUENCE

A note about the parameters in some of the Draper commands such as WALK: some of the units of the parameters were different from 1, like FOOT_X, whose unit was 0.0001. This difference in units meant that although the desired value was 0.01 cm, it was necessary to divide 0.01 by 0.0001 to get the appropriate parameter value for the command.

A.4.2 monitor.tls

The monitoring sequences:

BUNDLE MONITOR

SEQUENCE PANIC ACTIVE
  WHENEVER STATUS.LB_CONTACT +
    STATUS.LF_CONTACT +
    STATUS.RB_CONTACT +
    STATUS.RF_CONTACT < 2 THEN
    IF MONITOR.ESTOP_ON = 0 THEN
      COMMAND ESTOP_TOGGLE
    END IF
    MESSAGE "TURNED ON EMERGENCY STOP, UNSTABLE"
  END WHENEVER
CLOSE SEQUENCE

SEQUENCE ESTOP_STATUS ACTIVE
  WHENEVER MONITOR.ESTOP_ON = 1 THEN
    MESSAGE "ESTOP IS ACTIVATED, SO NO COMMANDS WILL BE HEARD"
  END WHENEVER
CLOSE SEQUENCE

SEQUENCE CONNECTION ACTIVE
  --- MONITOR CONNECTION, IF CONNECTION IS LOST
  WHENEVER MONITOR.CONNECTED = 0 THEN
MESSAGE "CONNECTION WITH AIBO LOST! PLEASE RESTORE CONNECTION"
END WHENEVER
CLOSE SEQUENCE

SEQUENCE OBSTACLE
WHENEVER STATUS.IR_NEAR < 350 THEN
  MESSAGE "OBSTACLE DETECTED"
  START DETECT_WALL.OBSTACLE
END WHENEVER
CLOSE SEQUENCE

SEQUENCE EVENT
WHENEVER STATUS.FB_BUTTON > 0 AND STATUS.BB_BUTTON > 0 THEN
  MESSAGE "EVENT DETECTED"
  START DETECT_EVENT.EVENT
END WHENEVER
CLOSE SEQUENCE

SEQUENCE DETECT_TOO_MUCH_PRESSURE
  DEFINE PRESSURE_THRESHOLD AS 0.8
  WHENEVER STATUS.LB_DUTY_ELEVATOR > PRESSURE_THRESHOLD OR
     STATUS.LB_DUTY_KNEE > PRESSURE_THRESHOLD OR
     STATUS.LB_DUTY_ROTATOR > PRESSURE_THRESHOLD OR
     STATUS.LF_DUTY_ELEVATOR > PRESSURE_THRESHOLD OR
     STATUS.LF_DUTY_KNEE > PRESSURE_THRESHOLD OR
     STATUS.LF_DUTY_ROTATOR > PRESSURE_THRESHOLD OR
     STATUS.RB_DUTY_ELEVATOR > PRESSURE_THRESHOLD OR
     STATUS.RB_DUTY_KNEE > PRESSURE_THRESHOLD OR
     STATUS.RB_DUTY_ROTATOR > PRESSURE_THRESHOLD OR
     STATUS.RF_DUTY_ELEVATOR > PRESSURE_THRESHOLD OR
     STATUS.RF_DUTY_KNEE > PRESSURE_THRESHOLD OR
     STATUS.RF_DUTY_ROTATOR > PRESSURE_THRESHOLD THEN
    MESSAGE "TOO MUCH PRESSURE ON JOINTS"
    START BASIC_ACTIONS.TOGGLE_ESTOP
END WHENEVER
CLOSE SEQUENCE

CLOSE BUNDLE

A.4.3 basic_actions.tls

Basic actions on the robot:
BUNDLE BASIC_ACTIONS
DEFINE CYCLE AS 0.1

SEQUENCE TK_WALK ACTIVE
    EVERY 1
        COMMAND FORWARD, FR_SEQ_NUM => 0,
            ROBOT_ID => 0,
            TASK_VALUE => PARAMS.FORWARD
    COMMAND STRAFE, FR_SEQ_NUM => 0,
        ROBOT_ID => 0,
        TASK_VALUE => PARAMS.STRAFE
    COMMAND ROTATE, FR_SEQ_NUM => 0,
        ROBOT_ID => 0,
        TASK_VALUE => PARAMS.ROTATE
    END EVERY
CLOSE SEQUENCE

SUBSEQUENCE STOP_TK_WALK
    STOP TK_WALK
    SET PARAMS.FORWARD TO 0.0
    SET PARAMS.STRAFE TO 0.0
    SET PARAMS.ROTATE TO 0.0
    WAIT CYCLE
    RESUME TK_WALK
CLOSE SUBSEQUENCE

SUBSEQUENCE START_TK_WALK
    STOP TK_WALK
    SET PARAMS.FORWARD TO 0.8
    SET PARAMS.STRAFE TO 0
    SET PARAMS.ROTATE TO 0
    WAIT CYCLE
    RESUME TK_WALK
CLOSE SUBSEQUENCE

SUBSEQUENCE TK_TURN
    STOP TK_WALK
    SET PARAMS.FORWARD TO 0
    SET PARAMS.STRAFE TO 0
    SET PARAMS.ROTATE TO 0.8
    WAIT CYCLE
    RESUME TK_WALK
    WAIT 0.7
    CALL STOPTKWALK
CLOSE SUBSEQUENCE
SEQUENCE TOGGLE_ESTOP
   COMMAND ESTOP_TOGGLE
CLOSE SEQUENCE

-- EVADE AN OBSTACLE
SEQUENCE EVADE
   CALL STOP_TK_WALK
   CALL TK_TURN
   CALL START_TK_WALK
CLOSE SEQUENCE

CLOSE BUNDLE

A.4.4 detect_event.tls

Detecting an event:

BUNDLE DETECT_EVENT

SEQUENCE MAIN
   IF BASIC_ACTIONS.TKWALK.SEQSTAT = SEQ_STOPPEDBYCMD THEN
      -- SHOULD CHECK TO MAKE SURE THESE VARIABLES AREN’T BEING MODIFIED
      SET PARAMS.FORWARD TO 0.8
      SET PARAMS.STRAFE TO 0
      SET PARAMS.ROTATE TO 0
      START BASIC_ACTIONS.TKWALK
   END IF
   START MONITOR.EVENT
CLOSE SEQUENCE

SEQUENCE EVENT
   START BASIC_ACTIONS.EVADE
CLOSE SEQUENCE

CLOSE BUNDLE

A.4.5 detect_wall.tls

Detecting a wall:
BUNDLE DETECT_WALL

SEQUENCE MAIN
   IF BASIC_ACTIONS.TK_WALK.SEQSTAT = SEQ_STOPPEDBYCMD THEN
      -- SHOULD CHECK TO MAKE SURE THESE VARIABLES AREN’T BEING MODIFIED
      SET PARAMS.FORWARD TO 0.8
      SET PARAMS.STRAFE TO 0
      SET PARAMS.ROTATE TO 0
      START BASIC_ACTIONS.TK_WALK
   END IF
   START MONITOR.OBSTACLE
CLOSE SEQUENCE

SEQUENCE OBSTACLE
   START BASIC_ACTIONS.EVADE
CLOSE SEQUENCE

CLOSE BUNDLE
Appendix B

AIBOChannel Specifications

The AIBOChannel application was developed specifically for this project, but for anyone who wishes to integrate Timeliner with another robotic platform, the inside details on how AIBOChannel works may be helpful. AIBOChannel is a translator between Timeliner and the AIBO.

B.1 Communication

B.1.1 From AIBO to AIBOChannel (Sensor Data)

The data transmitted by the AIBO was 244-byte packets of sensor data. (See AIBO_REALTYPE in Appendix A.2.2) In order to initiate the stream of data, the AIBO had to receive the string “connection request” at its port 10031. It would then start sending data at the rate of 4 packets every 32 ms to the IP address and port of AIBOChannel that sent the connection request.

B.1.2 From AIBOChannel to Timeliner (Telemetry)

In order to packetize the data sent by the AIBO, AIBOChannel takes as input a file that describes the data, the GDB_TESTER_FILE.txt. This file is actually output by
gdb.exe, and contains the addresses and sizes of each element in the Timeliner CVT. Using this file, AIBOChannel pads each ‘element’ of the data sent by the AIBO with its address in the CVT and the number of bytes in the element. The resulting array of bytes contains, for every element of data:

- address (4 bytes)
- length of data in bytes (2 bytes)
- data (variable number of bytes)

For this project, all of the data sent by the AIBO is eventually stored in the CVT, which makes it simple to tell AIBOChannel what data to send to Timeliner. (Just pad and send all of it!) It would probably be more space-efficient for the CVT, however, to have AIBOChannel pick what parts of the data sent by the AIBO is useful to Timeliner and selectively send those data, padded accordingly.

Note that an address of 0 indicates to Timeliner the end of the data.

### B.1.3 From Timeliner to AIBOChannel (UDP Commands)

Since Timeliner sends only UDP commands, AIBOChannel only needs to be able to receive UDP packets from Timeliner. For the Draper commands that need to be UDP anyway, AIBOChannel simply transmits the command along without modifying it. For the other commands, though, a special format was devised to indicate to AIBOChannel that the command needed to be translated into TCP. If the received UDP command had 0xA0 as its eighth byte, it needed to be translated. AIBOChannel was then programmed to send the TCP commands that corresponded to the indication in the ninth byte. For example, to toggle the emergency stop, the ninth byte was set to 0x00.
B.1.4 From AIBOChannel to AIBO (Commands)

The Draper UDP commands were straightforward, and simply needed to be forwarded as they were received to port 20051 on the AIBO. The TCP commands were more complicated because they required that AIBOChannel ensure that the appropriate port was open on the robot for receiving the desired command. This meant that AIBOChannel had to send multiple messages to traverse the Tekkotsu menu interface of the AIBO to ensure the receipt and activation of its commands. (A basic explanation of how to navigate through the menus can be found at http://www.cs.cmu.edu/~tekkotsu/TekkotsuMon.html.) In AIBOChannel, different functions such as walking or the emergency stop were designated to different threads for communication with their assigned ports. Table 3.1 outlines some of the ports associated with different functions.

B.2 SimpleMonitor

The AIBOChannel additionally uses the SimpleMonitor application, which is a dumbed down version of the CVTMonitor developed by Joseph Bondi and Emily Braunstein. It intercepts telemetry data being sent to the Timeliner executor and displays the most recent values in a GUI for the human operator. This is especially useful for debugging, but the application may have introduced some lag into the system.
Appendix C

AIBO Specifics

The following specifications for the AIBO ERS-7 can also be found at http://www.cs.cmu.edu/~tekkotsu/AiboInfo.html [17].

Computer Operation

- 576MHz MIPS R7000 processor
- 64 MB RAM
- 802.11b wireless ethernet (standard)
- MemoryStick reader/writer

Mechanical Joints and Sensors

- 18 PID joints, each with force sensing
  - 4 legs
    - 3 joints each (elevate, rotate, knee)
    - 1 paw button each
  - 3 joints on neck (tilt, pan, nod)
  - 2 joints on tail (tilt, pan)
  - 1 joint on mouth
- 2 ears, 1 boolean joint each (flick up or down)
- 26 independent LEDs
- Video camera
  - 56.9° wide and 45.2° high
- Resolutions: 208x160, 104x80, 52x40 pixels
- 30 frames per second
- Stereo microphones
- 3 IR distance sensors
- X, Y, and Z accelerometers

Buttons
- 4 pressure sensitive buttons (one on head, three on back)
- 1 boolean button under mouth

Sensor updates
- Sensor updates every 32 ms, with 4 samples per update.

The Aibos run a special operating system developed by Sony, called Aperios. Sony has released a software development kit, the OPEN-R SDK (written in C++). Programming for the Aibo is for the most part very similar to a UNIX environment, except for process control and inter-process communication.

The development environment uses a slightly patched version of the 3.2 GCC compiler, and can be run on almost any UNIX based platform, including Mac OS X and cygwin under Windows.

C.1 Draper Leg Definitions

The legs of the AIBO are referenced by the 1’s in a four bit number, which makes it simple to reference multiple legs at once. For example, the front left leg is referenced by 1000 or the number 8. To reference the front left leg and the back left leg at the same time, use 1010 or 10.
Figure C-1: 4-bit representation of the dog’s legs for the Draper algorithms. Leg 0 is represented by the highest order bit and leg 3 by the lowest order bit.
Appendix D

Useful References

Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIBO</td>
<td>Artificial Intelligence roBOt, or <em>companion</em></td>
</tr>
<tr>
<td>CDH</td>
<td>Command and Data Handling</td>
</tr>
<tr>
<td>CVT</td>
<td>Current Value Table</td>
</tr>
<tr>
<td>GDB</td>
<td>Ground Database</td>
</tr>
<tr>
<td>ISS</td>
<td>International Space Station</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TL</td>
<td>TimeLiner</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UIL</td>
<td>User Interface Language</td>
</tr>
</tbody>
</table>

Table D.1: Commonly Used Abbreviations

Reading Material


UIL Language Specification If the Programmer’s Reference does not provide enough
detail for you, the language specification should do the trick. Can also be found at http://timeliner.draper.com/user.docs.html. [10]
Bibliography


