ARCHITENTERPRISES

Team 20

Archit Rupakhetee, Raymond Tang, Tyler Lenig, Andrew McMillion

Lab number 9

Final Binder

May 1st, 2013

Version 1.0

By signing below, each group member approves of this document and contributed fairly to its completion.

Raymond Tang, Andrew McMillion, Archit Rupakhetee, Tyler Lenig

On our honors, as students of the University of Virginia, we have neither given nor received unauthorized aid on this assignment.

Raymond Tang, Andrew McMillion, Archit Rupakhetee, Tyler Lenig

ARCHITENTERPRISES

Contents

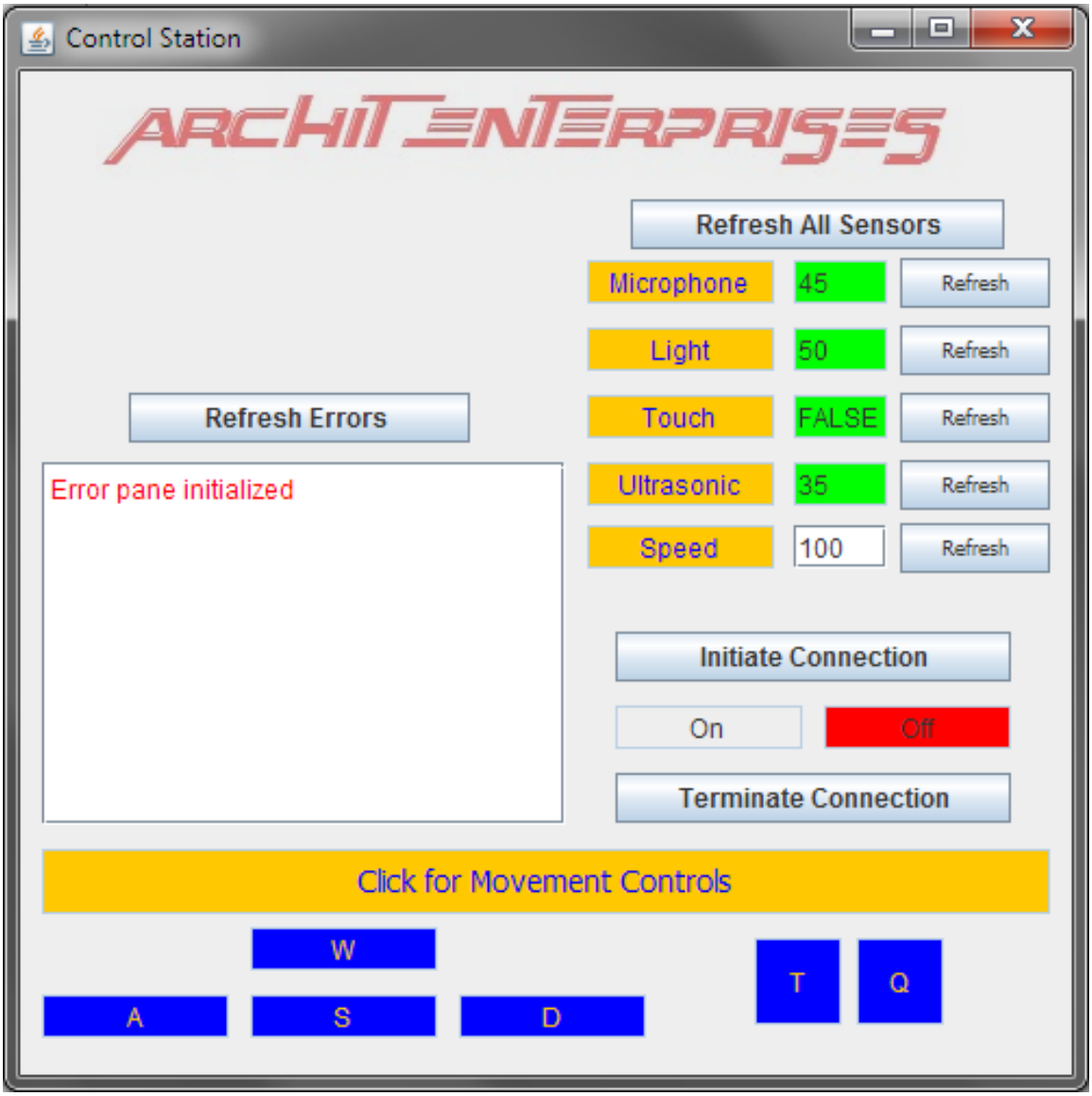
System Overview

The system we were tasked with designing was a robotic exploration system that is capable of exploring hostile environments. Besides the ability to move the robot also has various sensors that are capable of providing data about the environment to the user. The system has two parts to it, the robot on-board software and a base station system that communicates and controls the robot. Furthermore we were also required to create a debugging system that allows the on-board robot software to be debugged from the base station.

The Robot's motion is controlled by two motors which have a wheel attached to them. There is also a third motor attached to the robot which is a swinging arm that could be used to move or hit various objects in the environment. The sensors that are attached to the robot are the following; an ultrasonic sensor, a touch sensor, a light sensor and a microphone. Some of these sensors, provide telemetry data to the base station and also has extra features implemented for them. The ultra-sonic sensor is attached to the front of the robot and searches for an obstacle immediately in front of the robot. If the sensor detects something immediately in the front then the robot stops. There is a similar feature implemented in the back using a touch sensor. If the touch sensor is pushed then the robot tops all backwards movements. When the microphone

detects a very loud sound the robot moves backwards continuously until the sound is below a threshold.

The base station system comes with an interface so that the user may interact with the robot through the keyboard and mouse. The interface looks as shown below:



Before the Robot can be controlled a connection must first be established by clicking the 'Initiate Connection' button. When the button is clicked the base station attempts to establish a Bluetooth connection with the robot whose information is hard-coded into the system. Once connection is established the interface shows this by highlighting the 'On' button instead of the 'Off' as show above. The Terminate connection is used to end the connection once use of the robot is finished. Valid keyboard inputs for the Base station system are the keys W, A, S, D, T and Q. The buttons corresponding to these keys on the interface change from being blue to yellow while they are pressed down. Basic movement of the robot is controlled using the W, A, S and D key. For example while the W key is held down the robot moves forward until the key is released. Our system also features the ability to move in an arc, if two buttons such as the W key and the D key are both pressed then the robot moves in an arc to the right. Pressing T once on the keyboard turns the robot around 180 degrees, this is a macro that allows for easy turn-around. A final key feature is a swinging arm, which "swings" or rotates when the Q key is pressed and held. Some keys not fully implemented, but notable are keys to increment and decrement the motor's speed.

The interface also displays telemetry data for the sensors for each individual sensor. Clicking the refresh all button at the top refreshes the data for

all sensors, while clicking the refresh button for the individual sensor refreshes the data for only that sensor. The ability to select only one sensor could prove useful in situations where you need to retrieve telemetry data faster or save bandwidth. The pane on the right provides error messages, which are received from the robot, and the refresh button simply refreshes those messages.

A key part of this system is communication between the halves. The standard for communication was developed using a communication specification document. This document outline how commands would be sent to the robot. We decided to use an 11-character message for our standard communications package. The first three of the 11-characters specifies a command that can be decoded by the on-board robot software. The next seven characters of the message act as the parameters for the command. The 11th and final character is a checksum character that is used for error-detection.

Finally, the debugger is a separate piece of software that is independent from the base-station software. It is run from the base station and allows deeper level access to the robot's on-board software. The debugger allows for the insertion of breakpoints at places in the on-board software so that isolating errors can be done quicker. The interface of the debugger comes with a few default breakpoints that are typically right before the execution of a key function. The debugger can

also alter values for the sensor, and other variables to further test for error.There is also a text field that allows for typing of debugger commands to insert breakpoints or change sensor values.

Overall this system is designed for exploration of an unknown environment everything in the design has been developed with that in mind. The current product is ready for this task and furthermore we have also left some versatility for the future. There are certain features and commands that have been implemented with the future could be further developed in the future for another purpose.

Debugger Specification (Our Group)

Preamble

The following document is our specification for the NXT robot's onboard debugging system. The system is intended to be used exclusively by the engineering team when there is a problem with the onboard software. It is part of the actual control interface that can be brought up when there is an error or system failure. This system is invoked when the onboard system fails, and it is manipulated by only the engineers, not the operator.

The debugging system allows for the engineering team to have in depth access to the robots variables and state. The main function of this system is to get the robot back up and running, so it is purposefully intended to give the engineers full control. This requires software components to be on the robot itself and in the base station. The base station holds the interface for interacting with the debugging software on the robot. These components in conjunction are used to return the robot to its operational state and allow the operator to continue to use the robot. The debugging systems largest restriction is if communication with the robot is cut off then there is not much the system can do until connection is established.

This document contains a list of modes, conditions, and other variables that the debugger can perform or check with respect to the robot. Furthermore, it also outlines how various errors can occur and how they are displayed to the engineering team. The goal is to provide clear and concise definitions of all elements contained in the document to aid in outlining the capabilities of the debugger.

Assumptions

Our mode table consists of three modes without any data initialization. This is because we are the debug system and in normal mode, our software simply monitors the system. Our event table and mode transition table displays and creates output after which brings the system back to the normal mode.

Since our specification is exclusively for the debugging side, we assume that there are no input data items for our side of the GUI, all input comes from the base control software. The input data items that we have included in this specification are utilized by our software, but the actual implementation and manipulation of the input data items is exclusively controlled by the main control software.

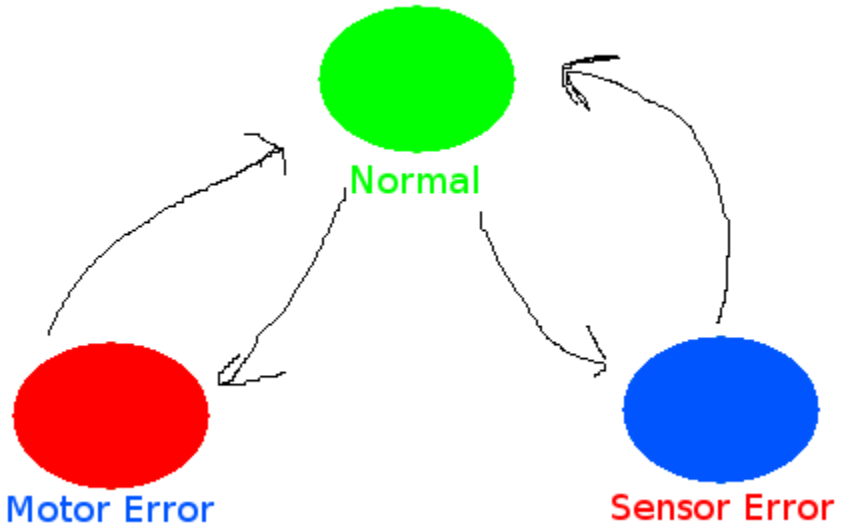
Modes

Mode	Definition
normal	Normal robot operation (i.e. no errors detected)
motor_error	Some error with a motor has been determined, this mode is where it is debugged. (i.e. motor is running too fast in the forward or backward direction)
sensor_error	Some sensor error has been determined, this mode is where it is debugged. (i.e. sensor is not transmitting data)

Mode Transition Table

Mode	*normal*	*motor_error*	*sensor_error*
normal	Robot is operating as expected (no motor or sensor error, i.e. @T(%no_error%))	any event that causes an error with a motor (i.e. @T(%motor_1_overclock_forward%))	any event that causes an error with a sensor (i.e. @F(%sensor_mic_running%))
motor_error	Action that fixes a motor error (i.e. !motor_1_speed! := $max_forward_speed$ //ERROR// shows $motor_1_error$)	-	-
sensor_error	Action that fixes a sensor error (i.e. //ERROR// shows $sensor_mic_error$)	-	-

Mode Transition Diagram



Our mode transition diagram shows the various modes our system can be in and how you get to and from each one. Our main mode is called *normal* and it is the mode in which the robot is operating under conditions that are making an errors. From this mode, our system goes into either *motor error* or *sensor error* depending on the type of error detected (all problems with motors go force the system into *motor error* and all problems with sensors force the system into *sensor error*). After the error is rectified, the system returns to normal mode, waiting on another error to surface.

User Interface

This is the basic design for the interface of the debugger. The debugger is integrated into the actual interface and the user types in a command and an access code that opens up the debugger. The console demonstrates this as the user is running program and an error is encountered. The command for the debugger is then typed, followed by the access code which starts the debugger. The debugger opens up on a separate window in the same interface with its own console. The debugger has direct access to sensor and motor values and it can further test and check values through the console.

Input Data Items

Input Data	Definition
/user_motor_1_speed/	The desired speed of motor 1 inputed by the user.
/user_motor_2_speed/	The desired speed of motor 2 inputed by the user.
/user_motor_3_speed/	The desired speed of motor 3 inputed by the user.

Input data item: motor 1 speed
Acronym: /user_motor_1_speed/
Class: Integer
Data representation: I/O port 1
Description: This field governs the speed of motor 1, it must be greater than or equal to 0 and less than or equal to $max_forward_speed$ or greater than or equal to $max_backward_speed$ and less than or equal to 0.
Value encoding: user defined integer value.
Timing characteristics: updated once user strikes enter key

Input data item: motor 2 speed
Acronym: /user_motor_2_speed/
Class: Integer
Data representation: I/O port 2
Description: This field governs the speed of motor 2, it must be greater than or equal to 0 and less than or equal to $max_forward_speed$ or greater than or equal to $max_backward_speed$ and less than or equal to 0.
Value encoding: user defined integer value.
Timing characteristics: updated once user strikes enter key

Input data item: motor 3 speed
Acronym: /user_motor_3_speed/
Class: Integer
Data representation: I/O port 3
Description: This field governs the speed of motor 3, it must be greater than or equal to 0 and less than or equal to $max_forward_speed$ or greater than or

equal to $max_backward_speed$ and less than or equal to 0.
Value encoding: user defined integer value.
Timing characteristics: updated once user strikes enter key

Output Data Items

Output Data	Definition
//motor_1_speed//	The angular velocity (degrees per second) of motor 1 (left motor).
//motor_2_speed//	The angular velocity (degrees per second) of motor 2 (right motor).
//motor_3_speed//	The angular velocity (degrees per second) of motor 3 (rear motor).
//ERROR//	Displays error output. This is an actual console displaying these errors.

Output data item: motor 1 speed
Acronym: //motor_1_speed//
Class: Integer
Data representation: I/O Port 1
Description: Displays current speed of motor 1
Value encoding: Some integer x such that:
0<= x <= $max_forward_speed$
` or
$max_backward_speed$ <= x <= 0
Timing characteristics: Update in background every 1 ms, displayed to user in console upon request

Output data item: motor 2 speed
Acronym: //motor_2_speed//
Class: Integer
Data representation: I/O Port 2
Description: Displays current speed of motor 2
Value encoding: Some integer x such that:
0<= x <= $max_forward_speed$
` or
$max_backward_speed$ <= x <= 0
Timing characteristics: Update in background every 1 ms, displayed to user in

console upon request

Output data item: motor 3 speed
Acronym: //motor_3_speed//
Class: Integer
Data representation: I/O Port 3
Description: Displays current speed of motor 3
Value encoding: Some integer x such that:
0<= x <= $max_forward_speed$
` or
$max_backward_speed$ <= x <= 0
Timing characteristics: Update in background every 1 ms, displayed to user in console upon request

Output data item: error message
Acronym: //ERROR//
Class: String
Data representation: String printed out to debugger console
Description: Error message that displays than an error code in execution and where the error occurred (i.e. motor or sensor)
Value encoding: String that comes from a predefined table of basic errors
Timing characteristics: Error message is printed to the console once it arrive in the interface within 1 ms of its arrival.

Event Table

Mode	Definition
normal ACTION	@T(%no_error%) //ERROR// shows no_error
motor_error ACTION	@T(%motor_1_overclock_forward%) !motor_1_speed! := $max_forward_speed$ //ERROR// shows $motor_1_error$
motor_error ACTION	@T(%motor_2_overclock_forward%) !motor_1_speed! := $max_backward_speed$ //ERROR// shows $motor_1_error$
motor_error ACTION	@T(%motor_3_overclock_forward%) !motor_3_speed! := $max_forward_speed$ //ERROR// shows $motor_3_error$
motor_error ACTION	@T(%motor_1_overclock_backward%) !motor_2_speed! := $max_backward_speed$ //ERROR// shows $motor_2_error$
motor_error ACTION	@T(%motor_2_overclock_backward%) !motor_2_speed! := $max_forward_speed$ //ERROR// shows $motor_2_error$
motor_error ACTION	@T(%motor_3_overclock_backward%) !motor_3_speed! := $max_backward_speed$ //ERROR// shows $motor_3_error$

motor_error ACTION	@T(!motor_1_speed! != 0 && //motor_1_speed// = 0) !motor_1_speed!:= 0 //ERROR// shows $motor_1_error$
motor_error ACTION	@T(!motor_2_speed! != 0 && //motor_2_speed// = 0) !motor_2_speed! := 0 //ERROR// shows $motor_2_error$
motor_error ACTION	@T(!motor_3_speed! != 0 && //motor_3_speed// = 0) !motor_3_speed! := 0 //ERROR// shows $motor_3_error$
sensor_error ACTION	@F(%sensor_mic_running%) !sensor_mic! := 0 //ERROR// shows $sensor_mic_error$
sensor_error ACTION	@F(%sensor_ultrasonic_running%) !sensor_ultra! := 0 //ERROR// shows $sensor_ultrasonic_error$
sensor_error ACTION	@F(%sensor_touch_running%) !sensor_touch! := 0 //ERROR// shows $sensor_touch_error$
sensor_error ACTION	@F(%sensor_light_running%) !sensor_light! := 0 //ERROR// shows $sensor_light_error$
sensor_error ACTION	@F(%bluetooth_sensor_running%) !motor_1_speed! := 0 !motor_2_speed! := 0

	!motor_3_speed! := 0 //ERROR// shows $connection_error$
sensor_error ACTION	@F(!time_robot_sent! - !time_base_received! <= $timeout$) !motor_1_speed! := 0 !motor_2_speed! := 0 !motor_3_speed! := 0 //ERROR// shows $connection_error$
sensor_error ACTION	@F(!time_base_sent! - !time_robot_received! <= $timeout$) !motor_1_speed! := 0 !motor_2_speed! := 0 !motor_3_speed! := 0 //ERROR// shows $connection_error$
sensor_error ACTION	@T(!poll_time! > $poll_interval$!motor_1_speed! := 0 !motor_2_speed! := 0 !motor_3_speed! := 0 //ERROR// shows $connection_error$

Glossary

Symbolic Constants

Name	Definition	Value
no_error	String	“Running well”
$motor_1_error$	String	“ERM0000001”
$motor_2_error$	String	“ERM0000002”
$motor_3_error$	String	“ERM0000003”
$sensor_mic_error$	String	“ERS0000001”
$sensor_ultrasonic_error$	String	“ERS0000002”
$sensor_touch_error$	String	“ERS0000003”
$sensor_light_error$	String	“ERS0000004”
$connection_error$	String	“Connection to the robot has been interrupted.”
$poll_interval$	Time (in ms) between base-station polls to determine Bluetooth connectivity.	1000ms
$timeout$	Time in ms that the !message_received! signal has to be received between !message_sent! before *connection_lost* is declared.	5000ms
$max_forward_speed$	Maximum speed at which motor can operate without reaching an !unsafe_speed!	TBD

$max_backward_speed$	Maximum speed at which motor can operate without reaching !unsafe_speed!	TBD

Text Macros

Name	Definition
!motor_1_speed!	Integer to contain motor 1 speed
!motor_2_speed!	Integer to contain motor 2 speed
!motor_3_speed!	Integer to contain motor 3 speed
!sensor_light!	Value to hold output from light sensor
!sensor_mic!	Value to hold output from microphone
!sensor_ultra!	Value to hold output from ultrasonic sensor
!sensor_touch!	Value to hold output from touch sensor
!poll_time!	Amount of time required to poll from base station from robot.

Condition	**Definition**
%Sent_Message%	A message has been sent.
%Receive_Message%	A message has been received.
%Await_Message%	A response is needed.
%Check_Format%	A message is verified to be in the correct format.
%no_error%	Not in an error mode.
%sensor_mic_running%	Microphone sensor is returning data.
%sensor_ultrasonic_running%	Ultrasonic sensor is returning data.
%sensor_touch_running%	Touch sensor is returning data.
%sensor_light_running%	Light sensor is returning data.
%bluetooth_sensor_running%	Bluetooth connection active
%motor_1_overclock_forward%	!motor_1_speed! >= $max_forward_speed$
%motor_2_overclock_forward%	!motor_2_speed! >= $max_forward_speed$
%motor_3_overclock_forward%	!motor_3_speed! >= $max_forward_speed$
%motor_1_overclock_backward%	!motor_1_speed! >= $max_backward_speed$
%motor_2_overclock_backward%	!motor_2_speed! >= $max_backward_speed$
%motor_3_overclock_backward%	!motor_3_speed! >= $max_backward_speed$

Control Specification (Partner Group)

Glossary

Symbolic Constants

Name	Definition	Value
max_speed	max speed of motors	TO BE DETERMINED
$NoOp$	no operation is taken by GUI	Null
$pressed$	button on GUI is pressed down	True
$released$	butoon on GUI was pressed and has been released	True
arc_radius	radius of arc taken by robot when multiple buttons pressed	TO BE DETERMINED

Name	Definition
!connection!	connection between the robot and base station
!error_message_table!	listing of all error messages to error code
!reading!	decoded /input_message/ to be displayed
!response!	message sent from robot to base station

Input Data Items

Name	Definition
/button_backwad/	controls backward movement
/button_forward/	controls forward movment
/button_left/	controls movement left
/button_right/	controls movement right
/button_sensor_light/	displays light sensor information
/button_sensor_unltrasonic/	displays ultrasonic sensor information
/input_speed/	input for new speed
/button_get_connection/	get connection
/button_end_connection/	end connection
/button_change_speed/	changes speed of robot
/input_message/	message received from robot

Output Data Items

Name	Definition
//data_log//	display for messages from robot
//sensor_light//	display for light sensor
//sensor_touch//	display for touch sensor
//sensor_sound//	display for sound sensor
//sensor_ultrasonic//	display for ultrasonic sensor
//output message//	message sent to robot

Conditions

Name	Definition
%connection_received%	Whether a connection is created or not.
%get_connection%	/button_get_connection/ = $released$
%message_recieved%	A message
%time_out%	10 seconds no response
%connected%	!connection! = True
%end_connection%	/button_end_connection/ = $released$
%is_error_message%	Whether /input_message/ is error

Set of Modes

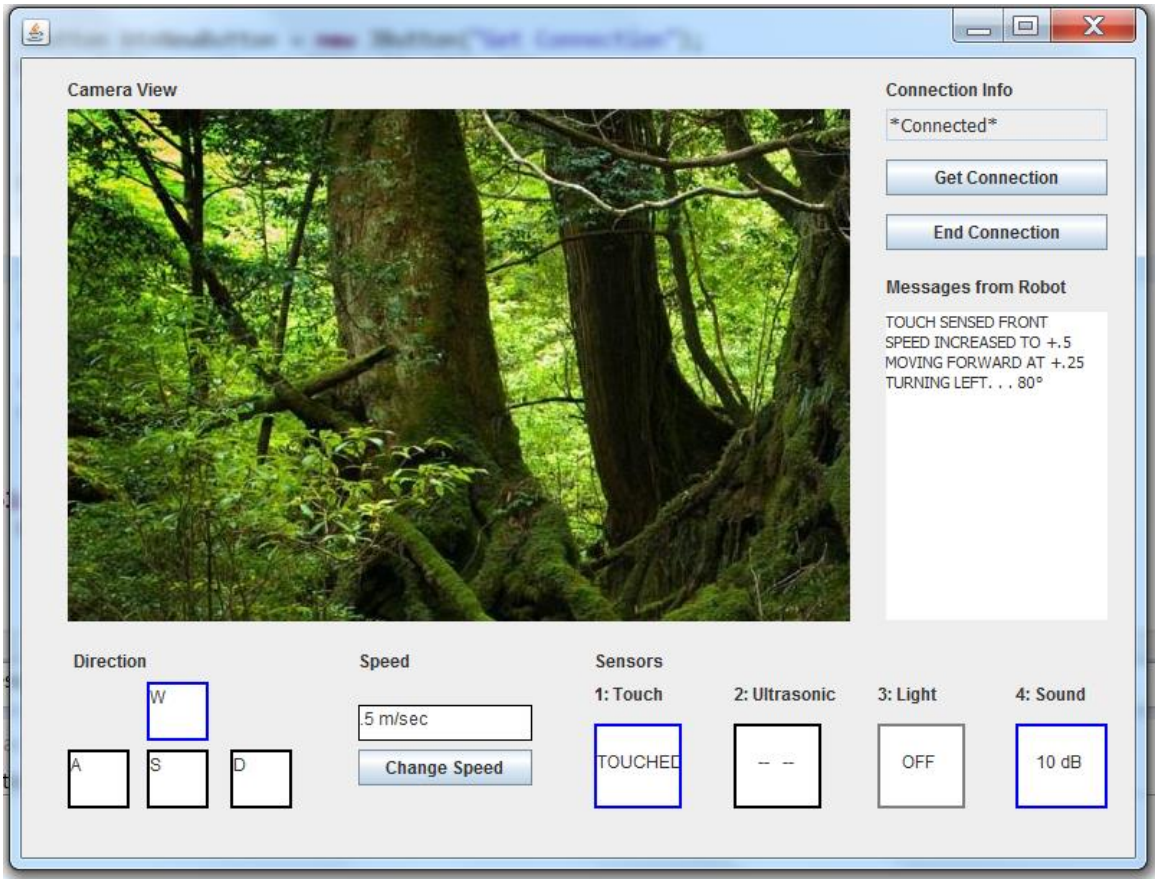
Name	Definition
Normal Operation	%connected%
Awaiting Connection	!connection! = false /button_get_connection/ = $pressed$
No Connection	!connection! = false

Mode Transition Table

	Normal Operation	*Awaiting Connection*	*No Connection*
Normal Operation		@T(%connection_received%)	@T(%end_connection%)
Awaiting Connection	@T(%connection_received)		@T(%time_out%)
No Connection		@T(%get_connection%)	

Define User Interface

Figure 1. Robot GUI



Inputs and Outputs

Inputs

Input data item: Forward push button
Acronym: /button_forward/
Hardware: Switch, normally open
Description: /button_forward/
- controls forward movement
- while pressed move forward, when released stop

Input data item: Backward push button
Acronym: /button_backward/
Hardware: Switch, normally open
Description: /button_backward/
- controls backwards movement
- while pressed move backward, when released stop

Input data item: Right push button
Acronym: /button_right/
Hardware: Switch, normally open
Description: /button_right/
- controls movement right

Input data item: Left push button
Acronym: /button_left/
Hardware: Switch, normally open
Description: /button_left/
- controls movement left

Input data item: Light sensor button
Acronym: /button_sensor_light/
Hardware: Switch, normally open

Description: /button_sensor_light/

-data from light sensor will be displayed in light sensor display

Input data item: Ultrasonic sensor button
Acronym: /button_sensor_ultrasonic/
Hardware: Switch, normally open
Description: /button_sensor_ultrasonic/

-data from ultrasonic sensor will be displayed in ultrasonic sensor display

Input data item: Speed input
Acronym: /input_speed/
Hardware: Switch, normally open
Description: /input_speed/

-receive keyboard input of numbers to change speed

Input data item: Change speed button
Acronym: /button_change_speed/
Hardware: Momentary switch, normally open
Description: /button_change_speed/

-change robot speed to speed currently in /input_speed/

Data Representation:

Byte 3 is Motor/Motor combinations

Bytes 4-9 is the new speed

Input data item: Get connection button
Acronym: /button_get_connection/
Hardware: Momentary switch, normally open
Description: /button_get_connection/

-transitions from *No Connection* to *Awaiting Connection*

Input data item: End connection button
Acronym: /button_end_connection/
Hardware: Momentary switch, normally open

Description: /button_end_connection/
-transitions from *Normal Operation* to *No Connection*

Input data item: Message received from robot
Acronym: /input_message/
Hardware: Communications link (bluetooth)
Description: /input_message/
-message sent from the robot

Outputs

Output data item: Message sent to the robot
Acronym: //output_message//
Hardware: Communications link (bluetooth)
Description: //output_message//
- encodes commands for the robot to complete based on user input
Characteristic of values: encoded based on communication specification; 10 character message

Output data item: Light sensor output
Acronym: //sensor_light//
Hardware: LCD monitor
Description: //sensor_light//
- display most recently read value from light sensor
Characteristic of values: Strings

Output data item: Sound sensor output
Acronym: //sensor_sound//
Hardware: LCD monitor
Description: //sensor_sound//
- display most recently read value from sound sensor
Characteristic of values: Strings

Output data item: Touch sensor output

Acronym: //sensor_touch//
Hardware: LCD monitor
Description: //sensor_touch//
- display most recently read value from touch sensor

Characteristic of values: Strings

Output data item: Ultrasonic sensor output
Acronym: //sensor_ultrasonic//
Hardware: LCD monitor
Description: //sensor_ultasonic//
- display most recently read value from ultrasonic sensor

Characteristic of values: Strings

Output data item: Display for messages from robot
Acronym: //data_log//
Hardware: LCD monitor
Description: //data_log//
- displays messages from robot
- displays error message from robot

Characteristic of values: Strings/sentences in textbox

Define Set of Events

Mode	Event	Action
Normal Operation	@T(/button_forward/ = $pressed$)	//output_message// = “MSF0000000” is sent
	@T(/button_backward/ = $pressed$)	//output_message// = “MSB0000000”
	@T(/button_left/ = $pressed$)	//output_message// = “TNL0000000”
	@T(/button_right/ = $pressed$)	//output_message// = “TNR0000000”
	@T(/button_left/ = $pressed$ AND /button_forward/ = $pressed$)	//output_message// = “MAFL000000”
	@T(/button_right/ = $pressed$ AND /button_forward/ = $pressed$)	//output_message// = “MAFR000000”
	@T(/button_left/ = $pressed$ AND /button_backward/ = $pressed$)	//output_message// = “MABL000000”

	@T(/button_right/ = $pressed$ AND /button_backward/ = $pressed$)	//output_message// = “MABR000000”
	@T(/button_right/ = $pressed$ AND /button_left/ = $pressed$)	$NoOp$
	@T(/button_forward/ = $pressed$ AND /button_backward/ = $pressed$)	$NoOp$
	@T(/button_forward/ = $released$)	//output_message// = “ST00000000”
	@T(/button_backward/ = $released$)	//output_message// = “ST00000000”
	@T(/button_left/ = $released$)	//output_message// = “ST00000000”
	@T(/button_right/ = $released$)	//output_message// = “ST00000000”
	@T(/button_left/ = $released$ AND /button_forward/ = $released$)	//output_message// = “ST00000000”
	@T(/button_right/ = $released$ AND /button_forward/ = $released$)	//output_message// = “ST00000000”
	@T(/button_left/ = $released$ AND /button_backward/ = $released$)	//output_message// = “ST00000000”

	@T(/button_right/ = $released$ AND /button_backward/ = $released$)	//output_message// = “ST00000000”
	@T(/button_right/ = $released$ AND /button_left/ = $released$)	$NoOp$
	@T(/button_forward/ = $released$ AND /button_backward/ = $released$)	$NoOp$
	@T(/button_change_speed/ = $released$)	Send //output_message// based on /input_speed/
	@T(/button_sensor_ultrasonic/ = $released$)	//output_message// = “RS30000000”
	@T(/button_sensor_light/ = $released$)	//output_message// = “RS40000000”
	@T(%get_connection%)	$NoOp$
	@T(%end_connection%)	Go to *No Connection*
	@T(/input_message/ = “RS1~”)	//sensor_touch// = !reading!
	@T(/input_message/ = “RS2~”)	//sensor_sound// = !reading!
	@T(/input_message/ = “RS3~””)	//sensor_ultrasonic// = !reading!
	@T(/input_message/ = “RS4~”)	//sensor_light// = !reading!

	@T(%is_error_message%)	lookup error in !error_message_table! and display message on //data_log//
Awaiting Connection	@T(%connection_received%)	Go to *Normal Operation*
	@T(%time_out%)	Go to *No Connection*
No Connection	@T(%get_connection%)	Go to *Awaiting Connection*

Design

Overview

BaseStation

The BaseStation class will be the main driver for the entire software. All the other classes will be ran through this class. This class will contain the main logic and decision making as well as the logic for controlling the robot's motors. Therefore, this class will interact with all the other classes. The BaseStation class will receive information from hardware buttons, display the information onto the GUI via an overloaded handleEvent() in the GUI class. It will also send that information to the robot through the Bluetooth class, which will be explained in more detail later. The BaseStation class will also receive information from the robot's on-board sensors, via classes that implement SensorInterface, and relay it to the GUI. This allows the BaseStation to hide how our entire system interacts and relays information to the GUI which enables greater modularity, lower coupling and increase cohesion.

GUI

The GUI class is driven by the base station class. It only interacts with the BaseStation class. It will receive information to display from the base station, as well as present information to the BaseStation class if given any information in the GUI from the user. The GUI class will include functionality to accept input

from the user via text fields, as well as display information given by the sensors and errors an embedded textbox.

Bluetooth

The Bluetooth class will be responsible for encoding and decoding the 11 character string as specified by our communications protocol. It will send and receive information from the robot via the built-in Lejos Bluetooth connection. The Bluetooth class will also be responsible for sending information to the correct places based on the type of the decoded information. It will send the information to the BaseStation class if the information is an error message or an acknowledgement that a command was received. It will also send information to the correct sensor if the command in question holds telemetry data. The Bluetooth class will therefore interact with both the BaseStation class and any class that implements SensorInterface.

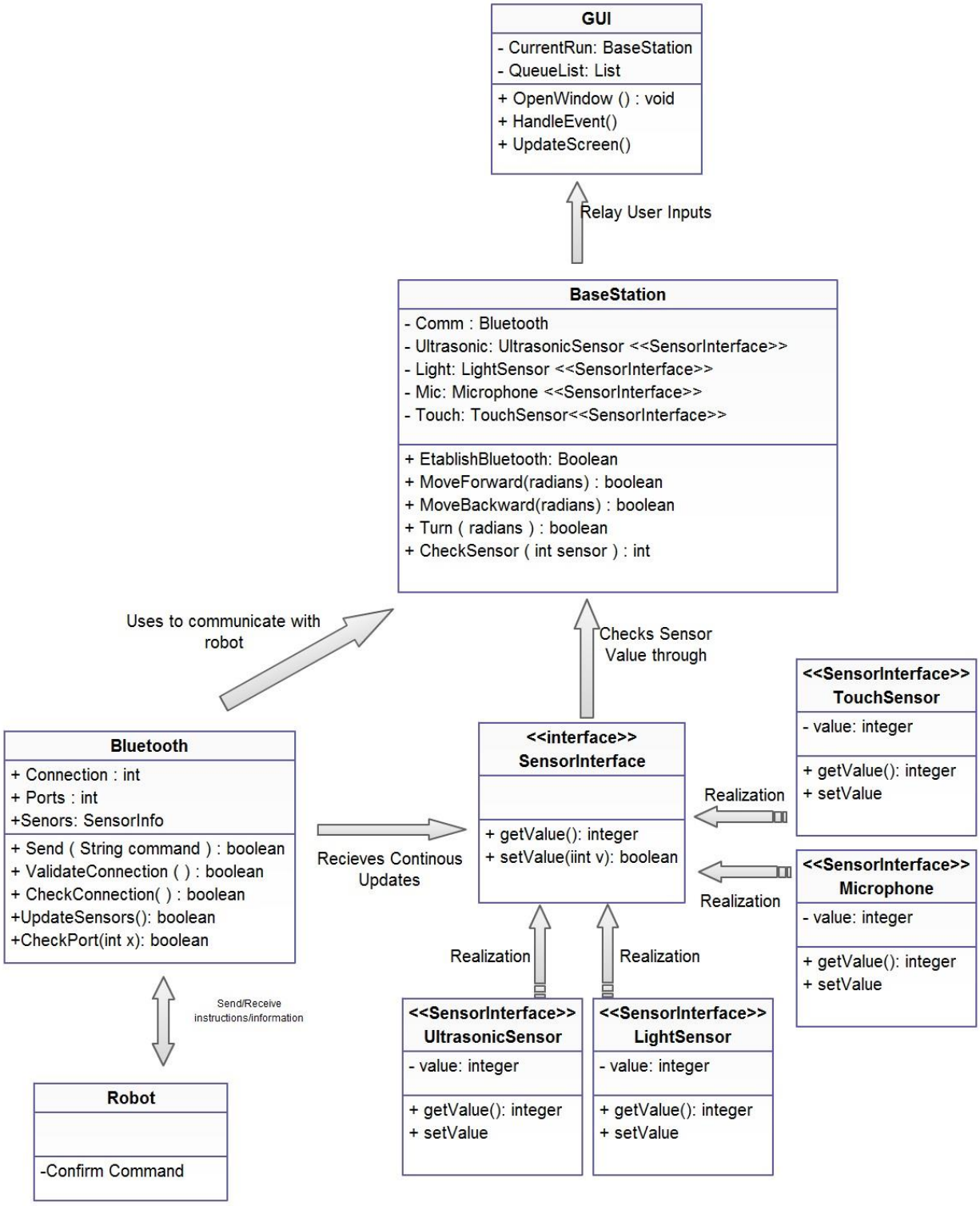
SensorInterface

This is an interface that outlines the basic methods that each sensor will implement. It will interact with the Bluetooth and BaseStation classes in order to send information to the GUI and receive telemetry information from the robot. Each sensor, Microphone, UltrasonicSensor, TouchSensor and LightSensor, will

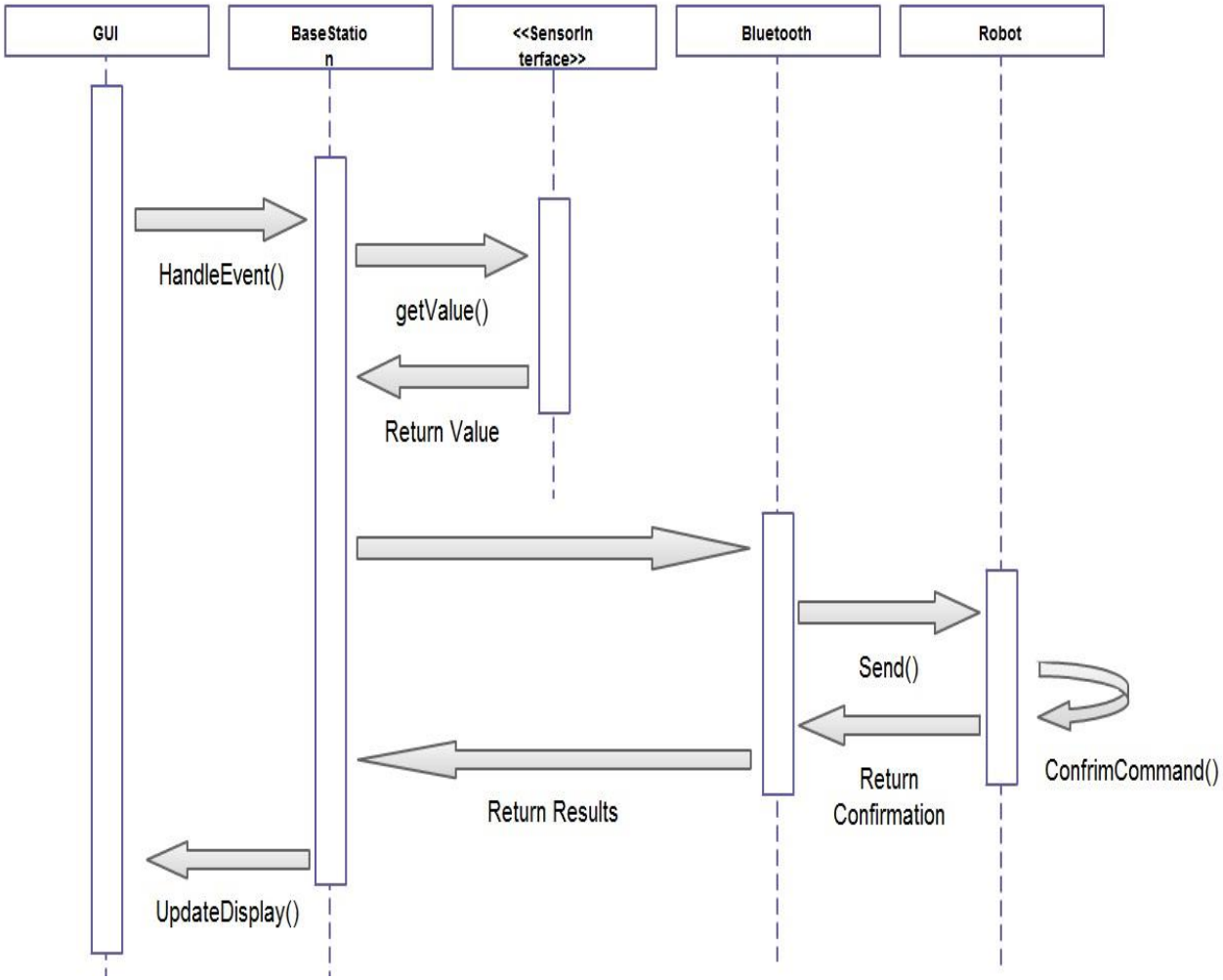
implement this interface and also include other methods specific to the given type of sensor.

UML Diagrams

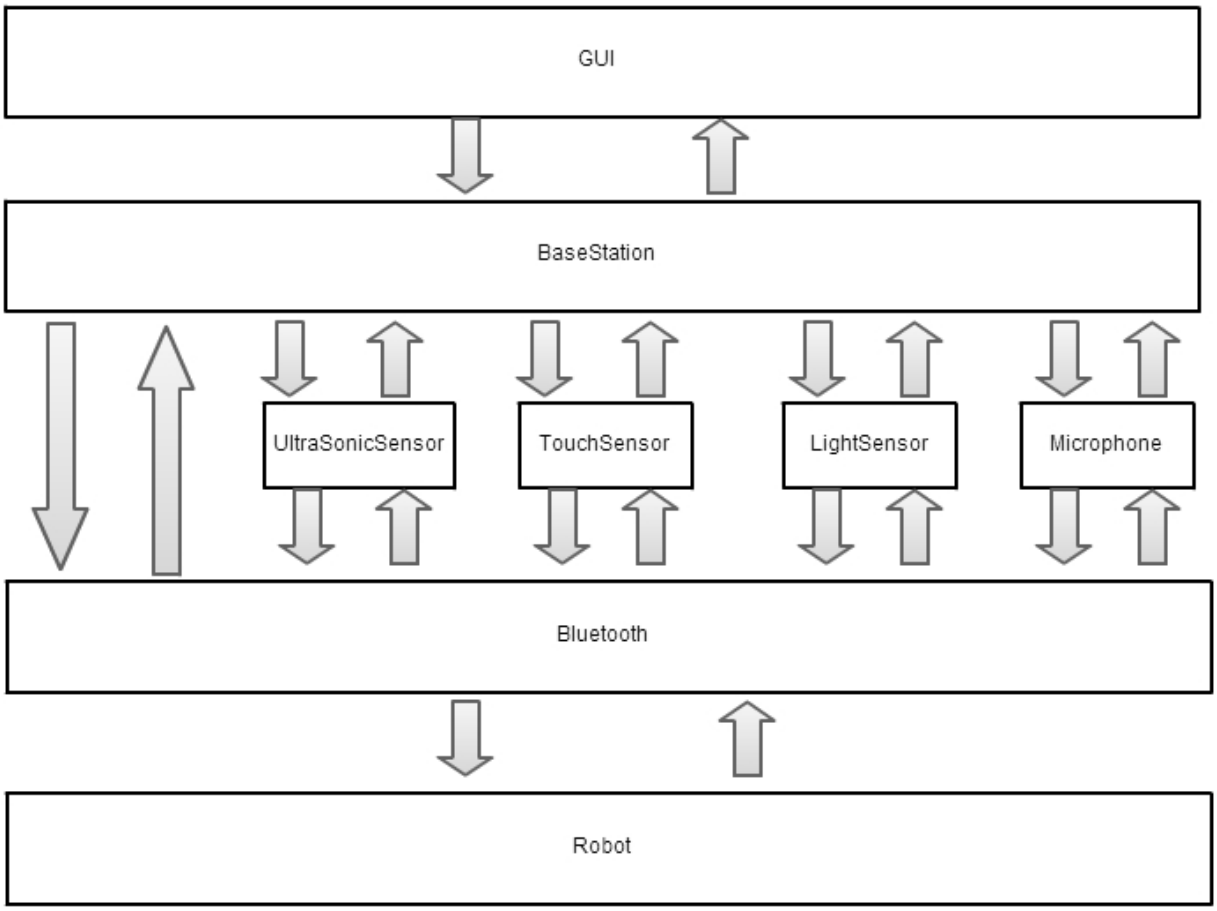
Class Diagram



Sequence Diagram



Concurrency Diagram



Class Interfaces

Our class interfaces are located on our website along with our other Lab 5 deliverables. They are also accessible at the following link to our Git:

https://github.com/Raytray/Archit-Enterprises/tree/master/Archit_Code/ClassImplementation/src

Anticipated Changes and Risks

Since this document is a living document, it is going to be changed as we further investigate our design. As we analyze our design from a usability perspective, we will more than likely change our GUI to make it as usable as possible. We will make changes to our communications protocol after extensive testing and evaluation of our design to ensure that our communications protocol and design will function harmoniously. Our classes will be supplemented with more functionality as we see fit throughout our design and implementation process to validate our design.

We also have various risks that we have already identified as negatively affecting our design. We anticipate having issues communicating with our robot through our system which will inhibit the system overall. Another risk we are facing is that our system might be too large to fit within the constraints of the hardware. Our schedule also presents a risk to us in the manner that we might not be able to complete all of the milestones we have set before the given deadline. In order to rectify these risks and other risks that may arise, we might have to implement the aforementioned changes and other changes to the system.

Source Code Listing

A list of the files in our delivered source code is below:

Archit_Logo.png BaseStation.java GUI.java

To view the source code, please follow the link below to our Git:

https://github.com/Raytray/Archit-Enterprises/tree/master/Archit_Code/Enhanced%20Prototype/src

Inspection

Schedule

We decided to spread out the entirety of our inspection activity across the two week period. This schedule is meant to show the main milestones we had for our prototype and when each was completed by.

April 15th

- Generate a checklist for each phase in the inspection
- Generate a schedule for completing the inspection
- Ensure which group members will complete each part of the inspection
- Ensure that Phase 1 of the inspection is completed

April 17th

- Swap results from Phase 1 with the other team
- Begin to complete the rework of our code from the Phase 2 results
- Begin to complete Phase 2 of the inspection

April 21st

- Swap results from Phase 2 with the other team
- Begin to complete the rework of our code from the Phase 2 results

April 25th

- Complete the Phase 3 inspection of our partner group's code
- Exchange the results of the last phase with our group

Group Member Assignments

Each group member played a crucial role in the completion of this inspection. Throughout the inspection we used both collaboration and solo work to complete it. Below are the lists of what each member completed individually but we all worked together to compile the inspection checklists, the schedule and the Phase 3 inspection.

Tyler Lenig completed the Phase 1 inspection and the compilation of all of the required portions of the inspection report.

Archit Rupakhetee completed the rework of our code after the results of the Phase 3 inspection were provided to us by our partner team.

Raymond Tang completed the Phase 2 inspection.

Andrew McMillion completed the rework of our code after the results of the Phase 1 and Phase 2 inspections were provided to us by our partner team.

Inspection Checklist

Phase 1

Internal documentation

- Comments
  - Used sparingly
  - Clarify complicated aspects of code
- Systematically convey information
  - File information at the top, such as authors, date, affiliations
- Only required libraries are imported
  - Each method uses camel case and describes an action
  - Variables declared in groups; no lone variable declarations

Layout

- All methods with similar functions are grouped together
  - Getter/Setters near each other
- Coherent code - code is in sensible order
- Spacing is consistent throughout
  - No extraneous whitespace
- Indentation is consistent

Phase 2

Coding practices (single inspector)

Each class is capitalized

Each line of code completes one cohesive action and is not overly long

All mathematical operations use parentheses for clarity

Comments

- Document source code
- Systematically convey information
  - File information at the top, such as authors, date, affiliations

Magic numbers

- Use symbolic constants
  - Includes variables and methods
  - No numbers that are unexplained

Meaningful identifiers

- No use of one letter identifiers
- Identifiers should help self-comment the code
  - Use LONG descriptive names
  - Camelcase for methods
  - Underscore as separator

Coding techniques

- All field declared private

Thorough getters and setters to access private fields

No use of dangerous coding techniques

Each class in a separate file

Phase 3

Meets specification document.

Implements required modes, text macros, symbolic constants

Contains every error message

Relational operators correct.

Correct use of dynamic storage/outside files

Appropriate class names/design aspects

Appropriate method names

Implements communication protocol fully

Sends acks, sensor data and error message to the GUI

Receives and processes properly formed messages

Handles checksum generation and validation correctly according to communications protocol.

Has operation capabilities when disconnected from the Base Station

Has basic movement capabilities

Turn, stop, move forward and move backward on key press, stopping on depress.

Interface handles and fixes run-time errors

Inclusion of breakpoints in code

Ability to monitor variable values at run-time

Group Member Efforts

Each group member expended the proper amount of effort when completing their required portion of the inspection.

Tyler Lenig utilized his time effectively throughout the lab weeks and completed his portion to the best of his abilities in the appropriate time frame. He spent approximate 5-8 hours on completion his portion.

Archit Rupakhetee utilized his time effectively throughout the lab weeks and completed his portion to the best of his abilities in the appropriate time frame. He spent approximate 3-5 hours on completion his portion.

Raymond Tang utilized his time effectively throughout the lab weeks and completed his portion to the best of his abilities in the appropriate time frame. He spent approximate 3-5 hours on completion his portion.

Andrew McMillion utilized his time effectively throughout the lab weeks and completed his portion to the best of his abilities in the appropriate time frame. He spent approximate 3-5 hours on completion his portion.

Even though each group member put in time on their own, we also spent approximately 5-8 hours working together to complete Phase 3 and other aspects of the inspection.

Implementation Questions

Below is the list of questions provided to us by our partner group.

1) Why does decode message and encode message and implement command use a string ArrayList?

2) Why do we include getPadding() function in Message Handler?

3) What are the arguments passed in the differential pilot for?

4) Why is the differential pilot passed Motor.B and Motor.C but not Motor.A?

5) Why is the setRotateSpeed of the Pilot originally set to 90?

6) What is COMMAND_TYPE_INDEX and why is it initialized to 0?

7) What is DEFAULT_RADIUS and why is it initialized to 90?

Answers to Implementation Questions

Below are the answers to the questions provided by our partner group.

1) It is because of readSensor that allows the sensor to arbitrarily add values. When commands are being encoded, decoded and implemented, there could be any number of commands in the queue. Having the data structure to store them be an ArrayList ensures that it can dynamically resize to accommodate any amount of commands.
2) To ensure the message is the correct length by adding 0s.
3) They represent the wheel diameter, the track width, the left motor and the right motor.
4) Because B and C are left and right motors, motor A will be used as an arm for the robot.
5) It is originally set to rotate at 90 because that is a very central and neutral rotational speed. If the user requires a lower or a higher rotational speed, they can set the speed manually.
6) It is the index used to locate what type of command is needed by finding the value at the index that is

COMMAND_TYPE_INDEX. It is initially set to 0 to ensure that the default noOp() case is the first case.

7) It is the value used by the move arc commands to ensure that the move arc commands only move in that severe of an arc. It is initially set to 90 so the arc will not exist unless specified to move in an arc.

Inspection Results

Phase 1

Internal documentation

Comments

Comments are used sparingly. One consideration would be to add a few comments to explain what each class is used for and what information it hides.

Some extraneous comments occur in the Debugger and DebuggerShell classes, consider revision or deletion.

Systematically convey information

Each file conveys their information in the appropriate fashion with fields at the top and methods below.

Only required libraries are imported

Only libraries utilized by the file are imported.

Each method uses camel case and describes an action.

This is completed to the fullest extent.

Variables declared in groups; no lone variable declarations.

All variables are declared near their instantiation and use.

Layout

All methods with similar functions are grouped together

For MessageHandler, consider grouping all of the decoding methods together. For Debugger, consider grouping all command related methods together. All other classes are complete.

Coherent code - code is in sensible order

Apart from grouping similar methods together, all of the code is in a sensible, coherent order.

Spacing is consistent throughout

No extraneous whitespace is evident. Spacing is consistent throughout, even from file to file.

Indentation is consistent

Indentation is consistent and appropriate throughout.

Phase 2

All files:

Include header information such as Author, Date, License.

Utilize spaces instead of tabs for code indentation.

Driver:

Line 14-24: Magic numbers should be replaced with symbolic constants indicating

that these values are the default values. Otherwise, the identifiers should have updated names indicating the symbolic constant they represent and that they are the default values.

Line 34: DifferentialPilot constructor has magic numbers that should be represented by symbolic constants.

Line 35: setRotateSpeed has magic number that should be represented by symbolic constant.

Line 61: Playing sound has numbers that too should be represented by symbolic constants.

Line 73: New lines should be inserted between logical blocks.

Line 131: boolean travel is unclear what it does, perhaps rename travel to is_traveling

line 131: Return statements are unclear as to their effect. Perhaps return SpeedSet = True or Error = False.

Line 178-212: Use of new line should be inserted between the outer if else statement.

line 238: private boolean stop() {} -- Good practice to only set boolean flag after stop command is issued to reduce errors in context switching or early termination causing incorrect flags.

MessagingHandler:

Line 84: Utilize symbolic constant for 256. -- Repeat for it's usage in other class declaration.

Line 94: Utilize MESSAGE_LENGTH symbolic constant declared at top.

Phase 3

Meets specification document.

Implements required SCR aspects (modes, text macros, etc)

Contains every error message

Every error message and appropriate SCR aspects are included.

Relational operators correct.

All relational operators (multiple instances) are correct.

Correct use of dynamic storage/outside files

No outside files are used and are not required.

ArrayLists are used to ensure that storage of run-time variables is possible (dynamic storage).

Appropriate class names/design aspects

Appropriate method names

All methods and class names are appropriate and describe clear aspects of the system.

Implements communication protocol fully

Sends acks, sensor data and error message to the GUI

Returns acks, error messages and sensor data to the GUI when they occur. Consider adding the ability to send all sensors at one time instead of a separate message for each sensor (when polled for all sensors).

Receives and processes properly formed messages

Properly handles well-formed messages and executes their desired function.

Handles checksum generation and validation correctly according to communications protocol.

Generates the appropriate checksum value and appends to the end of the message to validated messages.

Has operation capabilities when disconnected from the Base Station (autonomous control)

No autonomous control has been implemented. The current assumption is that the robot stops and initiate reconnection procedure when it is disconnected. Consider allowing the robot to operate based on sensor input alone until the connection has been reestablished.

Has basic movement capabilities

Line 40 utilizes motor C. Is this intentional? Line 42 sets Motor.C as well, and swing calls motor C. Is the pilot supposed to be Motor A and B by this?

Turn, stop, move forward and move backward on key press, stopping on depress.

Robot has basic movement capabilities and is able to move forward, backward, turn and stop.

Interface handles and fixes run-time errors

Inclusion of breakpoints in code

Breakpoints were not added yet. Consider adding them, perhaps at the beginning of every method or wherever you see appropriate.

Ability to monitor variable values at run-time

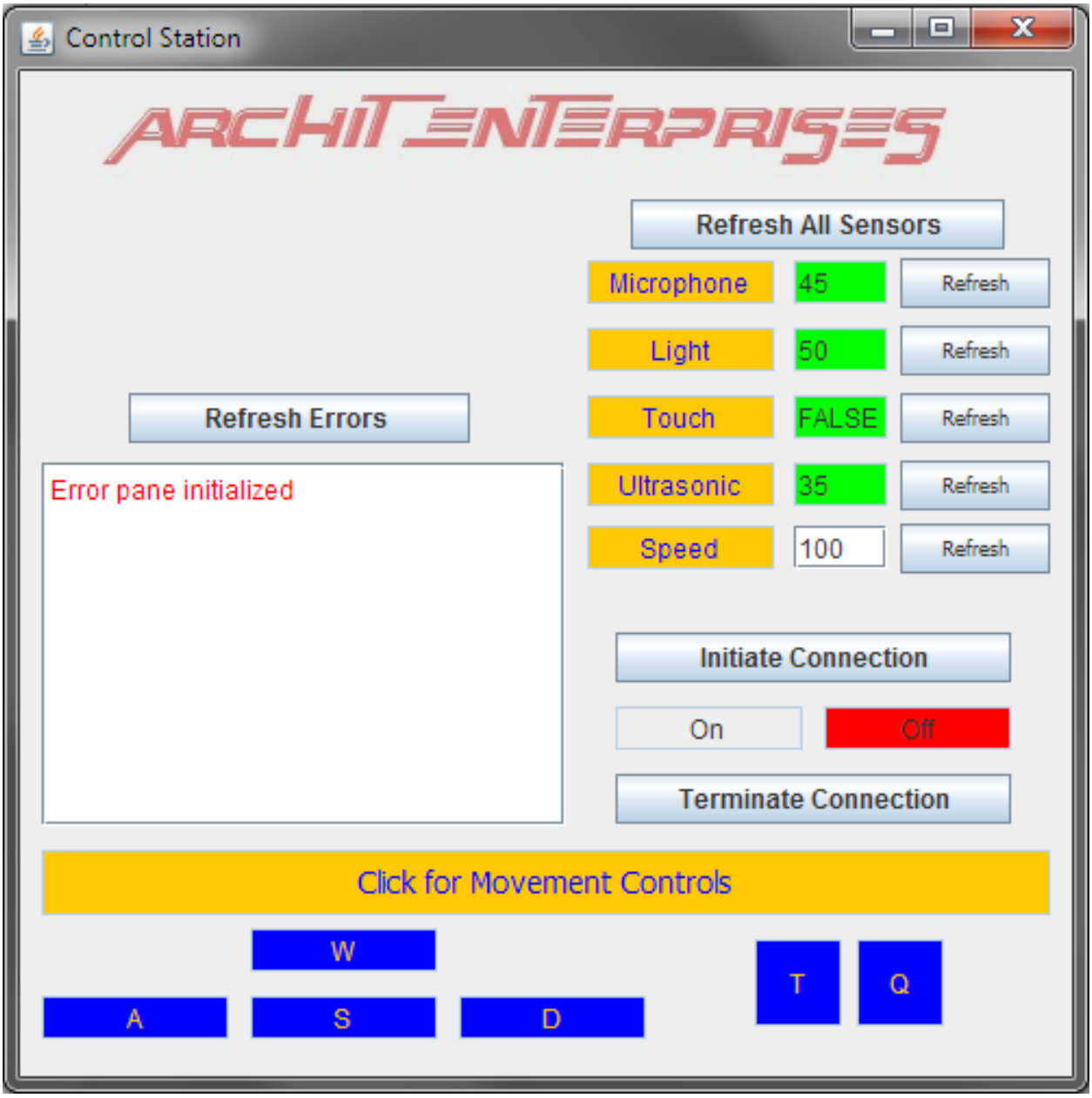
Unable to monitor variable values at runtime.

Rework Results

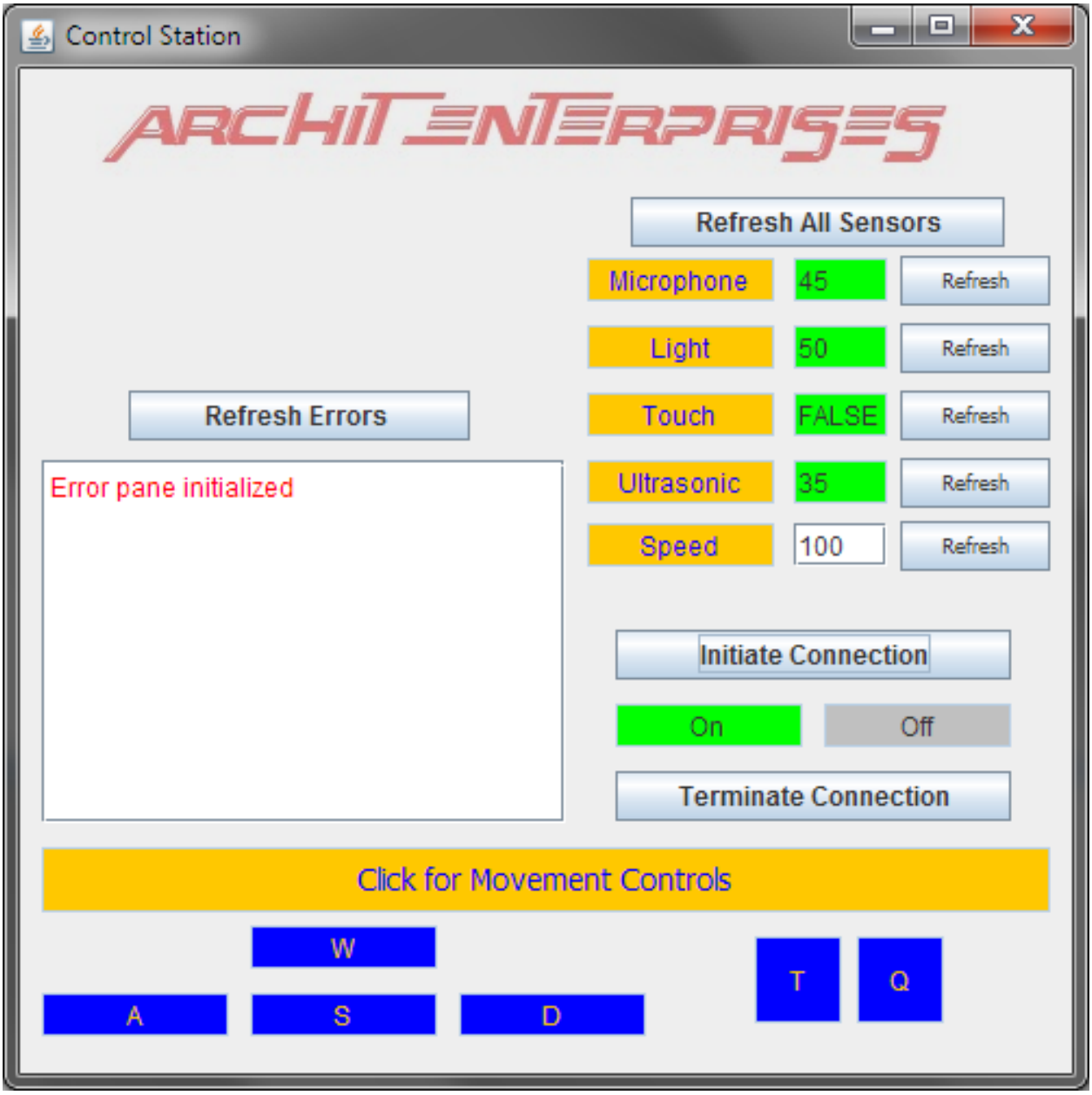
After receiving ample feedback from our partner group, we spent a majority of the week reworking our code between phases and afterward to ensure that our code is as complete as possible. Although we did split up the reworking of code amongst different group members, specifically Andrew McMillion and Archit Rupakhetee, we all played an integral role in reviewing and ensuring the code was as professional as possible.

In terms of lines of code added, we added approximately 50 lines of code. We also had to modify approximately 200 lines of code.

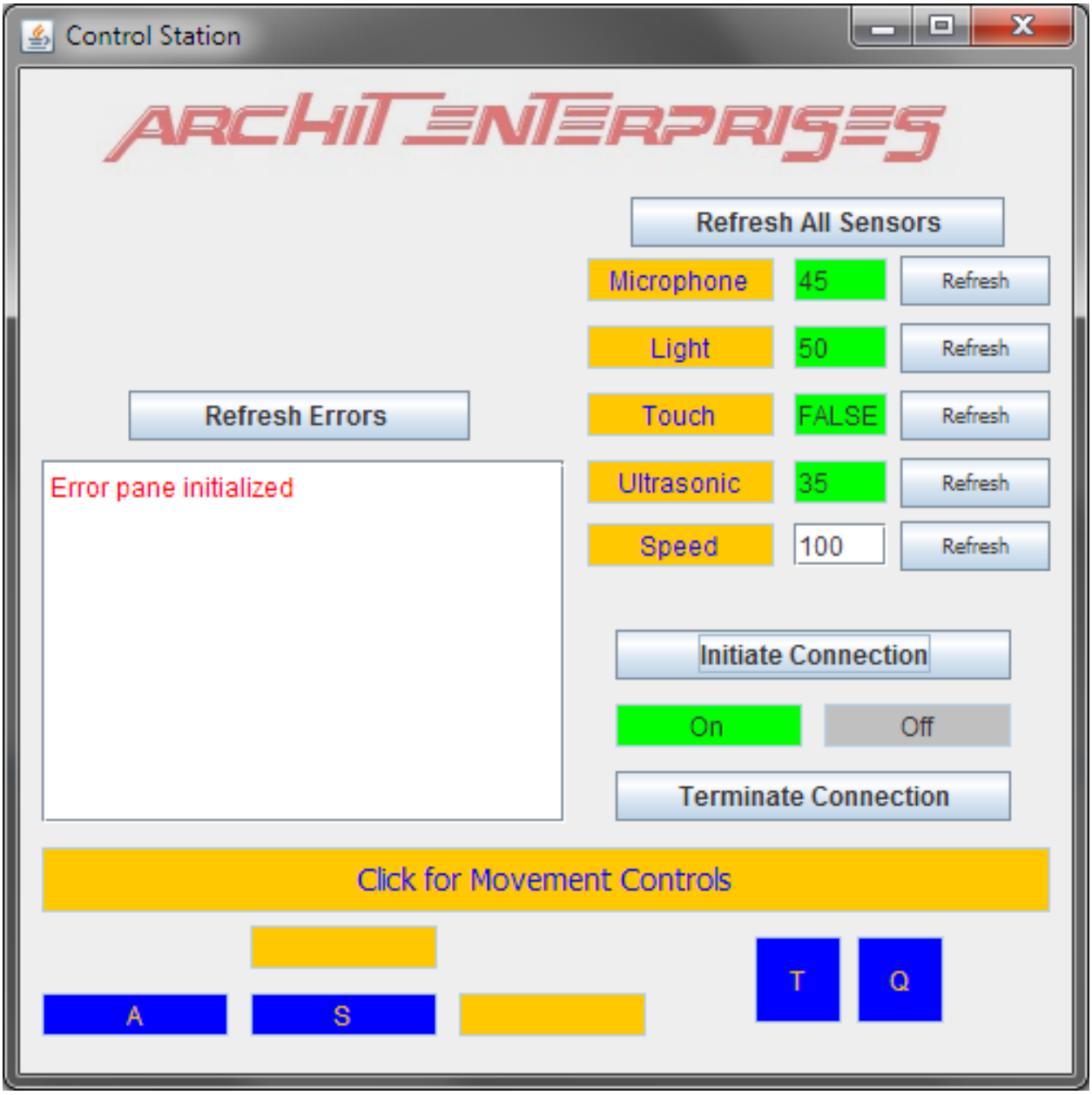
User Interface



The GUI immediately after initialization. The connection pane indicates that a connection with the robot has not been established. Mock sensor values are displayed, and the error pane is initialized with a confirmation message.



The GUI immediately after successful connection with the robot. The connection pane indicates that a connection with the robot has been successfully established. The mock sensor values do not change until the first sensor poll after connection is established, at which point the values displayed represent real telemetry from the robot's on-board sensor array.



The GUI during keyboard control input. The connection pane indicates that a connection with the robot has been successfully established. The key indicators light up to indicate corresponding key presses, and the robot will move in the desired direction for as long as the commands are held (or until autonomous control dictates otherwise, like if a hazard is detected nearby). The GUI does not

allow conflicting movement commands, such as ‘W’ and ‘S’ pressed simultaneously (commanding the robot to move both forward and backward concurrently).

Management Reports

Management Report Overview

Below are a few examples of our management reports. Each management report is made up of a variety of sections including management responsibilities, group member contributions, weekly meeting report, upcoming schedule and unresolved issues. Over time, we were able to improve our management reports by adding information about our advancement in the Capability Maturity Model and by developing a consistent meeting schedule.

Management Report for Lab 2

Management Responsibilities

Scheduling and task assignment and was undertaken by Tyler Lenig.

Configuration management and file system control was undertaken by Archit Rupakhetee, Andrew McMillion and Raymond Tang.

Document preparation was undertaken by every member of our group.

Presentation preparation was undertaken by every member of our group.

Web site development was undertaken by Raymond Tang.

The main point of contact with the other team was Tyler Lenig.

Group Member Contributions

Throughout the completion of this lab, each group member contributed equally to the following:

- Documentation of the inspection of the debugging system software
- Documentation of the communications protocol
- Presentation for Lab 3
- The following components of the SCR documentation for the debugging system
  - Event Table
  - Text Macros
  - Symbolic Constants

With regards to the SCR documentation of the debugging system, each member did the following:

- Archit Rupakhetee completed the graphic mock-up of our user interface and the preamble
- Raymond Tang completed the mode transition diagram and the mode transition table
- Andrew McMillion completed the condition table and the mode definitions
- Tyler Lenig completed the input and output data items.

Meeting #1

Attendees:

Raymond Tang, Tyler Lenig, Andrew McMillion, Archit Rupakhetee

Location:

Wilsdorf Hall on February 11th at 4:45 PM

Agenda:

Get familiar with SCR documentation and the debugging system.

Results:

This meeting was our first of the week and allowed us to generate a list of cases that we are going to use to complete our specification. The initial list we created was not in SCR documentation, but simply a list of potential features our system should have. This initial list served as our second version of our debugging system specification. We also made contact with our partner group to set up a time to discuss the communications protocol.

Meeting #2

Attendees:

Raymond Tang, Andrew McMillion, Archit Rupakhetee, Tyler Lenig and Group 20

Location:

Rice Hall on February 15th at 1 PM

Agenda:

Discuss the basic features the communication protocol should have.

Results:

This meeting was our second of the week and served as the initial meeting with our partner group that we were supposed to have last week (we were unable to meet due to a variety of scheduling conflicts). We were able to discuss the key aspects our communications protocol should have. We also set up a Google document to allow for collaboration on the completion of our communications protocol.

Meeting #3

Attendees:

Archit Rupakhetee, Raymond Tang, Andrew McMillion, Tyler Lenig

Location:

Rice Hall on February 18th at 3:30 PM

Agenda:

Continue compiling our debugging system specification

Results:

Our third meeting of the week allowed us to truly begin working on creating our specification in SCR documentation. As advised in class to do, we began by creating a list of modes that we felt were appropriate for our system and a glossary of other components. This served as our third version of our document. We also set up a time to meet later in the week to complete our specification.

Meeting #4

Attendees:

Andrew McMillion, Tyler Lenig, Archit Rupakhetee, Raymond Tang

Location:

Rice Hall on February 23rd at 3 PM

Agenda:

Continue completing our debugging system specification

Results:

We used our fourth meeting to continue to complete our specification. In this meeting, we discovered a few components that we omitted from our prior versions of our document as well as began to format our document into a presentable product. We also used this meeting to finalize a mock-up of our interface. We decided that we needed to meet the following day to complete all of the tasks still needing to be completed.

Meeting #5

Attendees:

Raymond Tang, Archit Rupakhetee, Tyler Lenig, Andrew McMillion

Location:

Rice Hall on February 24th at 3:30 PM

Agenda:

Finalize our debugging system specification, management report and inspection document

Create a presentation for lab 3

Results:

This was our final meeting of the week. We used this meeting to complete all of the required documents for the post-lab as well as create a presentation for the next studio laboratory. We also set up the schedule for next week.

Ensure all documents are completed with proper formatting and key sections included

This task will be completed by each group member by February 23rd.

Ensure all documents are added to the Github repository.

This task will be completed by Tyler Lenig by February 24th.

Ensure all documents are placed in an easily accessible way on the website.

This task will be completed by Raymond Tang by February 25th.

Along with these concrete tasks to complete, we will also hold our regular meeting on Monday, February 25th, and begin to analyze our partner group's specification. We will also begin to implement some of the process refinement strategies that we fleshed out in our presentation for lab 3.

Unresolved Issues

This week did not result in many more unresolved issues than we had from last week. The main issue we had this week was communication with our partner group. We had a slight lapse in communication that resulted in a slight delay when completing the communications protocol but we were able to overcome it in the end.

Some issues that we have to overcome for next lab are as follows:

Learning to use our Gantt chart to its fullest potential

Learning to use the CoCoMo effort estimate to its fullest potential

Management Report for Lab 6

Management Responsibilities

Scheduling and task assignment and was undertaken by Tyler Lenig.

Configuration management and file system control was undertaken by Archit Rupakhetee, Andrew McMillion and Raymond Tang.

Document preparation was undertaken by every member of our group.

Presentation preparation was undertaken by every member of our group.

Web site development was undertaken by Raymond Tang.

The main point of contact with the other team was Tyler Lenig.

Group Member Contributions

With regards to the deliverables for post-lab 6 and the source code for pre-lab 7, each member contributed in the following fashions:

Archit Rupakhetee and Raymond Tang compiled the milestones, schedule and prototype goals.

Archit Rupakhetee ensured that the proper commands were sent when a given key was pressed.

Raymond Tang compiled the checksum function for the communication commands.

Tyler Lenig compiled the management report, ensured that the connection to the robot was possible and worked on compiling every section of our code into one cohesive unit.

Andrew McMillion work extensively on creating and refining the GUI

Meeting #1

Attendees:

Raymond Tang, Tyler Lenig, Andrew McMillion, Archit Rupakhetee

Location:

Rice Hall on April 1st at 4:45 PM

Agenda:

Compile a schedule for the upcoming week

Complete the deliverables for post-lab 6

Results:

This meeting was our first of the week and is becoming our regular scheduling meeting. In addition to developing a schedule for the week, we used this meeting to complete the design documentation, including an overview of the functionality of our prototype and the schedule/milestones that go along with it. We also refined our code for our prototype.

Meeting #2

Attendees:

Raymond Tang, Andrew McMillion, Archit Rupakhetee, Tyler Lenig and Group 19

Location:

Rice Hall on April 5th at 3:30 PM

Agenda:

Complete Integration Test with Partner Team

Discuss various aspects of the system to incorporate

Results:

This meeting was our second of the week and served as the joint meeting with our partner group to complete the integration testing of our system. We were able to successfully test basic movement (i.e. moving the robot forward continuously) and advanced movement (i.e. being able to move the robot when a key is pressed and stop it when it is released). We were also able to discuss various functionality of our system including breakpoints for the debugging system.

Meeting #3

Attendees:

Archit Rupakhetee, Raymond Tang, Andrew McMillion, Tyler Lenig and Group 19

Location:

Rice Hall on April 7th at 4 PM

Agenda:

Complete the required program code for pre-lab 7

Results:

This last meeting of the week was the most labor intensive for our group. We used this meeting to ensure that our system will achieve the desired in-lab goals. We collaborated on every aspect of the system and did extensive testing using our test client to ensure that the desired goals were met.

Ensure all documents are completed with proper formatting and key sections included

This task will be completed by each group member by April 7th.

Ensure all documents are added to the Github repository.

This task will be completed by Tyler Lenig by April 7th.

Ensure all documents are placed in an easily accessible way on the website.

This task will be completed by Raymond Tang by April 8th.

Along with these concrete tasks to complete, we will also hold our regular meeting on Monday, April 8th, and begin to understand the requirements for post-lab 7 as well as set up a schedule that works for everyone for the upcoming week. We will continue to refine our code for delivery throughout the week to ensure that the final product pleases the customer. We will also continue to refine our process to make our way to higher levels in the CMM.

Unresolved Issues

The main issues we are having right now have to do with our enhanced prototype. This lab is one of the most important of the semester and requires a great deal of care when completing it. We had a variety of issues these last two weeks, mainly with integration testing. Integrating our system with our partner group's system is a very challenging task. In order to overcome this issue, we will continue to conduct integration tests to make sure that our system will work as specified.

We also were unable to fully test our telemetry data before the lab this week which was quite nerve-racking. We will complete this test this week to ensure that our telemetry data works as specified.

Cool features were another area that we struggled with. We need to hold an outside session to brainstorm ideas to give our design interesting and innovative features that will separate ours from other groups.

Management Report for Lab 8

Management Responsibilities

Scheduling and task assignment and was undertaken by Tyler Lenig.

Configuration management and file system control was undertaken by Archit Rupakhetee, Andrew McMillion and Raymond Tang.

Document preparation was undertaken by every member of our group.

Presentation preparation was undertaken by every member of our group.

Web site development was undertaken by Raymond Tang.

The main point of contact with the other team was Andrew McMillion.

Group Member Contributions

With regards to the inspection report for post-lab 8 and the delivered code for lab 9, each member contributed in the following fashions:

All members contributed equally to developing the pre-lab 8 inspection checklists and the Phase 3 inspection of our partner group's code.

Tyler Lenig conducted the Phase 1 inspection of the partner group's code and completed the management report.

Archit Rupakhetee ensured our code was up to date after receiving the results of the Phase 3 inspection of our code.

Raymond Tang conducted the Phase 2 inspection of our partner group's code, developed the questions for our partner group to answer about our code and answered the questions posed to us by our partner group.

Andrew McMillion ensured our code was up to date after receiving the results of the Phase 1 and 2 inspections of our code.

Meeting #1

Attendees:

Raymond Tang, Tyler Lenig, Andrew McMillion, Archit Rupakhetee

Location:

MEC 213 at 3:30 PM

Agenda:

Ensure the inspection checklists are complete and cohesive

Develop a schedule for the upcoming two weeks

Designate tasks for the week

Results:

This meeting was our first of the week and occurred during the scheduled laboratory period. Since we did not have a studio laboratory this week, we were able to treat this one as our preliminary one and we did the normal routine we do every week. We were able to synthesize schedules and develop a game plan for the upcoming weeks.

Meeting #2

Attendees:

Raymond Tang, Andrew McMillion, Archit Rupakhetee, Tyler Lenig

Location:

Rice Hall on April 22nd at 3:30 PM

Agenda:

Review components required for the delivered code

Discuss requirements for inspection

Results:

This was our second meeting of the two week period and we used it mainly to brainstorm ideas for our delivered code. We discussed a variety of cool features that we thought were plausible for our code and came up with a list that we felt was appropriate. We also used this time to discuss the inspection and what we still needed to complete for it.

Meeting #3

Attendees:

Raymond Tang, Andrew McMillion, Archit Rupakhetee, Tyler Lenig

Location:

Rice Hall on April 25th at 7:00 PM

Agenda:

Conduct the Phase 3 inspection

Ensure that the group is available to complete the code over the weekend

Results:

This was our third meeting of the period and we used it to complete the Phase 3 inspection. We initially had each member complete the inspection by themselves and then we came together as a group and synthesized our results. We also used it to make sure we all had an open schedule for the weekend to complete the required deliverable code.

Meeting #4

Attendees:

Raymond Tang, Andrew McMillion, Archit Rupakhetee, Tyler Lenig and Group 19

Location:

Rice Hall on April 28th at 3:00 PM

Agenda:

Finalize the delivered code

Results:

This was our last meeting of the week and we used it to ensure that our code and our partner group's system were up to date and fully functional. We used this time to make sure that our cool and innovative features were implemented and functional and that all of the regular features continued to work.

Ensure all documents are completed with proper formatting and key sections included

This task will be completed by each group member by April 28th.

Ensure all documents are added to the Github repository.

This task will be completed by Tyler Lenig by April 28th.

Ensure all documents are placed in an easily accessible way on the website.

This task will be completed by Raymond Tang by April 29th.

Since our class is coming to an end after the laboratory session on Monday, April 29th, the only further scheduling required is to compile our laboratory binder. We will have at least one meeting between April 29th and May 2nd to ensure that our binder is as complete as possible before it is delivered.

Throughout the semester we have been able to resolve our CMM to Level 4: Managed. We are now able to control and effectively manage the software development lifecycle through the utilization of each group member's strengths. With more time, we would be able to optimize our process and be able to innovate our process.

Unresolved Issues

After completing our system, the only unresolved issue remaining is the construction of our binder. This issues should be easily and quickly resolved because we have all of the required pieces, all we have to do is put them together in a professional way.

Other Documents

Risk Analysis

Overview

The following document represents some risks we were having with our development process in its early stages and the way we were able to resolve each risk. We represented the data in a tabular fashion to ensure quick access and analysis of our decisions.

Problem	Details	Resolution
Motors being calibrated to different settings	Each of the motors likely have different sensitivity and rotate at different speeds even when the inputs are the exact same. Making it difficult to have the robot travel in a straight path.	API allows you to individually set motor speed so we can find the difference in each motors speed and account for the discrepancies.
Wheels become easily uneven	The wheels for the robot tend to easily become misaligned, causing the robot to overturn or under turn as well as create difficulties traveling in a straight path.	Before each run with the robot check to make sure the wheels are even.

Problem	Details	Resolution
Robot may sometimes not receive signals	Signals send to the robot are not confirmed so there is a possibility that the robot may not properly receive commands.	Create a confirmation system where the robot is able to indicate or return a signal that confirms that the previous command was properly received. (Note: Bluetooth may already account for this)

Problem	Details	Resolution
Physical memory limit	The NXT brick has a limit to how much program we can save in the actual robot.	Unfortunately there is no way for us to increase the actual size in the NXT brick. The best we can do is write optimal code. This includes things such as reusing code as much as possible and writing algorithms in their shortest form. We can also save space by having only basic commands and codes in the NXT and signaling complex commands from the base station.

Data Transmission Speed and Latency — can the link cope with communication protocol's requirements?

Problem	Details	Resolution
Robots response speed to our commands	We need to know how fast the robot reaction time to our commands, how fast the NXT brick is able to process code and how fast we can respond or give consecutive commands.	Testing the robot for each of the conditions and record findings for future references.

Problem	Details	Resolution
Microphone picking up unnecessary noise from the room	The microphone may pick up normal noise around the room as an instruction and cause error in our programs.	The API allows for selecting a range for the sensitivity of the microphone. We will test and search for the appropriate threshold. If we are unable to find a proper threshold we may resort to muffling the microphone with a cover.

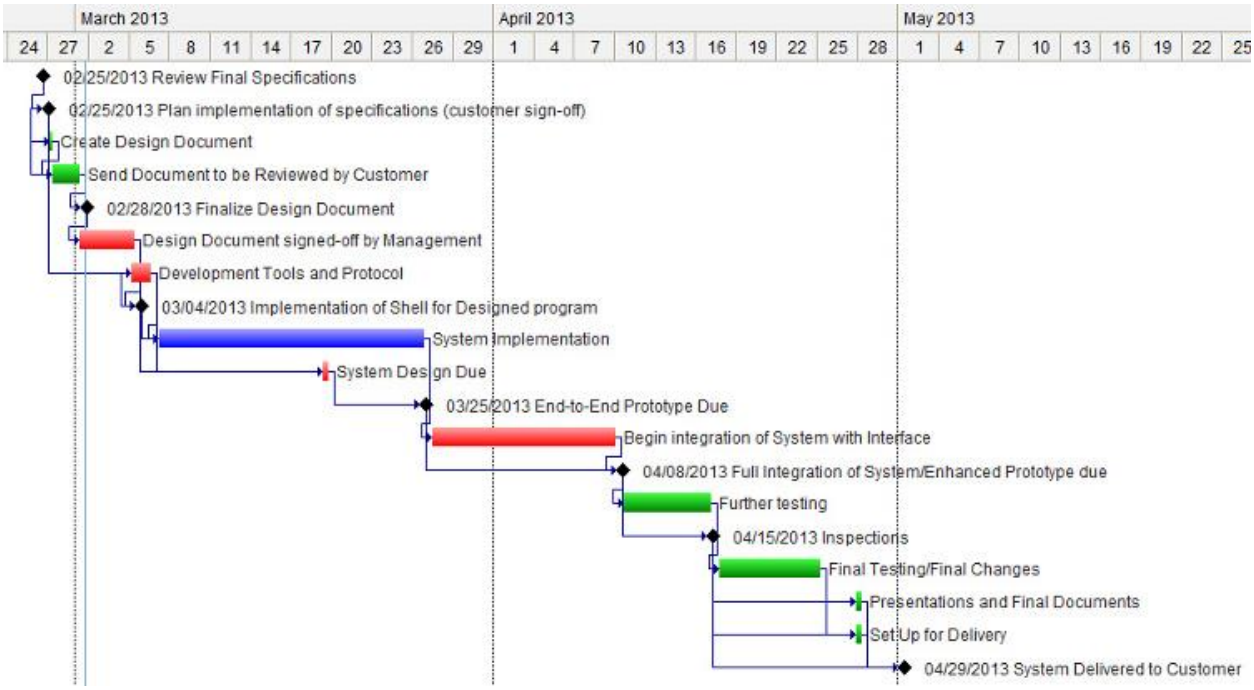
Gantt Chart and CoCoMo Estimation

Overview

The following information represent a few tools we used to organize our process and help us estimate the time commitment needed to complete the project. We used a Gantt chart to help us ensure that we would complete the project on time. We used a COCOMO estimation algorithm to generate an appropriate time estimate to complete the project.

Gantt Chart

Below is a screenshot of our Gantt chart.



Effort Multipliers

As we completed our COCOMO estimation, we decided which effort multipliers mattered to our project and which ones did not. If the effort multiplier is not listed below, it was determined to be nominal and carry a value of one which would not affect our COCOMO estimation.

Required software reliability: high (1.3)

We decided that our software reliability must be high because if the on board software of the robot is not highly reliable, the robot will not be able to function so reliability is paramount.

Database size: very low (0.9)

Since our program does not make use of any databases, its influence in our estimation is minimal.

Execution time constraint: very high (1.3)

The robot must be able to execute commands in real time to allow the user full control over the robot so this effort multiplier is very important.

Main storage constraint: high (1.25)

Since the robot has a small amount of memory, the main storage constraint is high for our system.

Programmer capability: high capability (0.93)

For this project, we believe that our programming prowess gives us an advantage with this system and makes the project easier to complete.

PL experience: high (0.92)

Each of us has a large amount of experience with Java, which is what this project is written in, which gives us an advantage with completing this project.

Estimation

Using the intermediate organic COCOMO estimation model and the effort multipliers above, we determined that:

$$EFFORT = \Pi_i EM_i * a * (KDL)^b$$

$$KDL = 1.1, a = 3.2, b = 1.05$$

$$EFFORT = 5.74 \text{ man-months}$$

After we compiled this number, we then calculated the total development time by computing

$$TDEV = 2.5 * (EFFORT)^{0.38}$$

$$TDEV = 4.86 \text{ months}$$

Communication Specification

Introduction

About

This document describes the protocol used to communicate between robot and base station system. This protocol allows the base station to control the robot, and allows the robot to send messages including errors to the base station.

Protocol Description

This protocol uses 11-character messages to communicate between the robot and the base station. The first 10-character of messages encode both commands from the base station to the robot and messages from the robot to the base station. The messages are structured such that the first two characters determine the type of command or message. The remaining characters are used for various parameters that are documented below. The 11th character holds the checksum used for error detection.

Command Structure

Commands are 10-character messages, where the first two characters are the command type. The remaining characters represent parameters to the command, used by the robot to determine how to execute the command.

No-Op

Message: 0000000000

Description: This command is the no operation command can be used to test if messages are being sent. This message is a “null” message.

Move Straight

Command Type: MS

Parameters: Forward/Backwards, and distance.

Character 2 is forward or backwards (F/B)

Characters 3-9 are distance (#), can be null (0s)

Description: This command moves the robot in a straight line. The forward/backward parameter control the direction the robot will move in. The distance allows for the robot to move a specified distance, this parameter can be null. If distance is null, the robot will continually move

Example Commands:

MSF0000000 will move the robot forward continuously.

MSB0001000 will move the robot backwards 1000 units.

Move Arc

Command Type: MA

Parameters: Forward/Backwards, left/right, radius, distance

Character 2 is forward or backwards (F/B)

Character 3 is left or right (L/R)

Characters 4-6 are radius (# degrees)

Characters 7-9 are distance (#), can be null (0s)

Description: This command moves the robot in an arc. The forward/backward parameter control the direction the robot will move along the arc. Left/Right will control the direction the robot arcs to. Radius is the absolute value of the number of degrees to move. The distance allows for the robot to move a specified distance, this parameter can be null. If distance is null, the robot will continually move until stopped.

Example Commands:

MAFL090000 will move the robot forward to the left along a 90 degree curve continuously

MABR030100 will move the robot backwards along a 30 degree curve for 100 units.

Turn

Command Type: TN

Parameters: Left/Right, and radius

Character 2 is left or right (L/R)

Characters 3-9 are radius (# degrees), can be null (0s)

Description: This command turns the robot when stationary. The Left/Right parameter determines the direction the robot turns. The Radius parameter is an absolute value that determines how far the robot turns. If the radius is null, the robot continually turns until stopped.

Example Commands:

TNR0000090 will turn the robot right 90 degrees

TNL0000000 will turn the robot left continuously

Stop

Message: ST00000000

Description: This command stops any actions that the robot is currently doing. This will end any movement actions.

Read Sensor

Command Type: RS

Parameters: Sensor Port

Character 2 is sensor type (U for Ultrasonic, T for touch, M for sound, L for light)

Characters 3-9 are 0

Description: This command will read a specified sensor. The Sensor Port parameter will determine which sensor to read the value of.

Example Commands:

RSU0000000 will cause the robot to read the value of the sensor, and send the data to the base station.

Set Speed

Command Type: SS

Parameters: Motor/Motor Combination, and new speed.

Character 2 is Motor/Motor combination (A for Motor A, B for Motor B, C for Motor C, D for Drive Motors)

Character 3 is set travel or rotate speed (T/R)

Characters 4-9 are the new speed

Description: This command will change the speed of the motors. The combination will determine which motors or combinations of motors to change the speed for.

Example Commands:

Read All Sensors

Command Message: RA00000000

Description: This command tells the robot to read all sensors and send the data. Each sensor's data will be sent to the base station in a separate message.

End Connection

Command Message: EC0000000

Description: This command instructs the robot to end connection with the base station.

Robot to Base Station Messages

Acknowledgment

Description: This message is sent to the base station as acknowledgment of receiving a command.

Message: AK00000000

Sensor Error Messages

Message Type: ERS

Parameters: Message number

Characters 3-9 are message number

Description: This message will tell the base station that an error with a sensor has occurred. The message number maps to a more specific description, that the base station will have stored locally for reference. Available messages can be seen in a table below, which will have additions added as required. Errors for sensors is

only if the sensor is disconnected. For the bluetooth sensor, it is if the connection is disconnected from the base station.

Message Number	Description
0000001	Error with sensor in port 1
0000002	Error with sensor in port 2
0000003	Error with sensor in port 3
0000004	Error with sensor in port 4

Motor Error Messages

Message Type: ERM

Parameters: Message Number

Characters 3-9 are message number

Description: These messages will tell the base station that an error with a motor has occurred. The message number correlates to a specific description, which the base station has stored locally. Available messages can be seen in a table below, which will have additions added as required. Errors for the motor includes if the motors are not connected, if the speeds are faster than set limit, and if the motors are stuck.

Message Number	Description
0000001	Error with motor in port A
0000002	Error with motor in port B

0000003	Error with motor in port C

Sensor Data Messages

Message Type: SD

Parameters: Sensor Type, and Data

Character 2 is sensor type (U for Ultrasonic, T for Touch, M for Sound, or L for Light)

Characters 3-9 sensor data

Description: These messages allow for the robot to send data to the base station based on the values of the sensor.

Error Detection

Introduction

The Error detection this protocol utilizes is a checksum for detecting errors in packets, and a timeout on acknowledgments. The checksum is calculated using only the first 10 characters of the message, then checked against the character that is sent in the packet. The timeouts will be 10 seconds before the sender will assume the packet was lost and needs to be retransmitted.

Checksum Function

The function for calculating the checksum is $(\text{checksum} = \sum_{i}^{10} (\text{byte})\text{message}[i]) \bmod 256$. This function allows for no matter the value of the checksum it will fit in a one-byte character. The function is the sum of the byte value of each character in the message modulo 256.