University of Virginia

Roboticorp Portfolio

David Bertoglio, Catherine Dworak, Steven Hauser, Jireh Miaw, Laura Morgan Team 19
May 3, 2013 – Laboratory 9 – Delivery

Approval Sheet

All group members whose names are listed below approve of the document and contributed fairly.

Member Names

Morgan, Laura
Miaw, Jireh
Hauser, Steven
Dworak, Catherine
Bertoglio, David

Pledge

On my honor, as a student, I have neither given nor received unauthorized aid on this assignment.

Names

Morgan, Laura
Miaw, Jireh
Hauser, Steven
Dworak, Catherine
Bertoglio, David

Table of Contents

About

Members of Team 19:
David Bertoglio, Catherine Dworak, Steven Hauser, Jireh Miaw, Laura Morgan

Team 19 Robot with light, touch, sound, and ultrasonic sensors and robotic arm.

Overview

Our team was given the task of the developing of the control system for a mobile robot that performs simple motion and has sensors that collect data about the environment. The robot is controlled from a base-station computer that communicates with the robot via Bluetooth to send and receive commands and telemetry data. The hardware is a Lego Mindstorms NXT robot utilizing a programmable microcomputer brick. The goal of this project was to design and create the software for this robot utilizing the software engineering process.

This robot is built out of legos with 3 wheels, 2 motors, and 4 sensors, including light, sound, ultrasonic, and microphone, as well as a swinging arm. It can perform simple movements, such as moving straight forward and backward, moving in an arc forward and backward, left and right, and turn in place left and right. These movements are controlled using the w-a-s-d keys on the keyboard. In addition to these basic movements, the robot has a few special features. 1) The robot has a swinging arm that swings clockwise when the key is pressed. 2) The robot turns exactly 180 degrees when the t key is pressed. 3) As a self-defnese mechanism, the robot moves backward when it detects loud sounds. All of these actions are performed by the interaction of the base station control system and the on-board system.

The base-station system runs on a computer and is operated by a human. This system takes commands from the user and sends messages to the on-board system to perform those commands on the robot. It contains a graphic user interface, consisting of w-a-s-d buttons, text fields to display sensor readings, buttons to refresh the sensor readings, buttons to control the swinging arm and then 180 degree turn, as well as buttons to change the speed of the robot. All of the functionality of the robot can be performed using this GUI. The base-station contains only 2 classes – a baseStation class that performs all the action (creating and sending messages when required), and a GUI class that interacts with the baseStation class when buttons are clicked.

The on-board system software runs on the microcomputer brick physically attached to the robot, and controls the movement of the robot based on commands it receives from the base-station. The on-board software consists of 3 classes – the activator class which establishes Bluetooth connection and sends and receives messages, the messageHandler class which decodes and encodes messages, and the driver class which accesses the NXT motors and sensors to control the movement of the robot and receive telemetry data. This class implements the Lejos API, a package class developed from Java that provides access to NXT motors, sensors, etc.

These two systems send messages back and forth via Bluetooth using a communication protocol developed by the team. The protocol is an agreed upon document that describes the composition of messages. Each message consists of 11 characters. The first two characters determine the type of command, the next 8 characters are used for various parameters for the command, and the last character holds the checksum used for error detection.

Finally, the robot system contains a debugging system. The base-station contains a GUI for the debugger, which can set breakpoints in the code, access state variables, and contains a log of previous messages sent and received. This is used to access the on-board software remotely in case an error occurs in the field.

The development of this robot control system took place over the course of a semester, utilizing 9 team members. These team members were separated into 2 groups, one that developed the on-board system and one that developed the base-station. The interaction and cooperation of every member of these two groups was essential for the completion and success of this robot.

Robot Control Specification

Laboratory # 2: Requirements and Specification

Work Product

The Specification Document describes the behavior of the robot control system. It includes the glossary, mode definition, mode transition table, conditions, input and output data items, and event table.

Document Revision Information

2/10/2013 – Template created
2/15/2013 – Mode definitions and events
2/18/2013 – Input and output data items
2/22/2013 – Glossary and event table
2/24/2013 - Completed

Approval Sheet

All group members whose names are listed below approve of the document and contributed fairly.

Morgan, Laura
Miaw, Jireh
Hauser, Steven
Dworak, Catherine
Bertoglio, David

Specification Document was inspected on 2/24/13
Specification Document was approved on 2/14/13

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

Text Macros

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

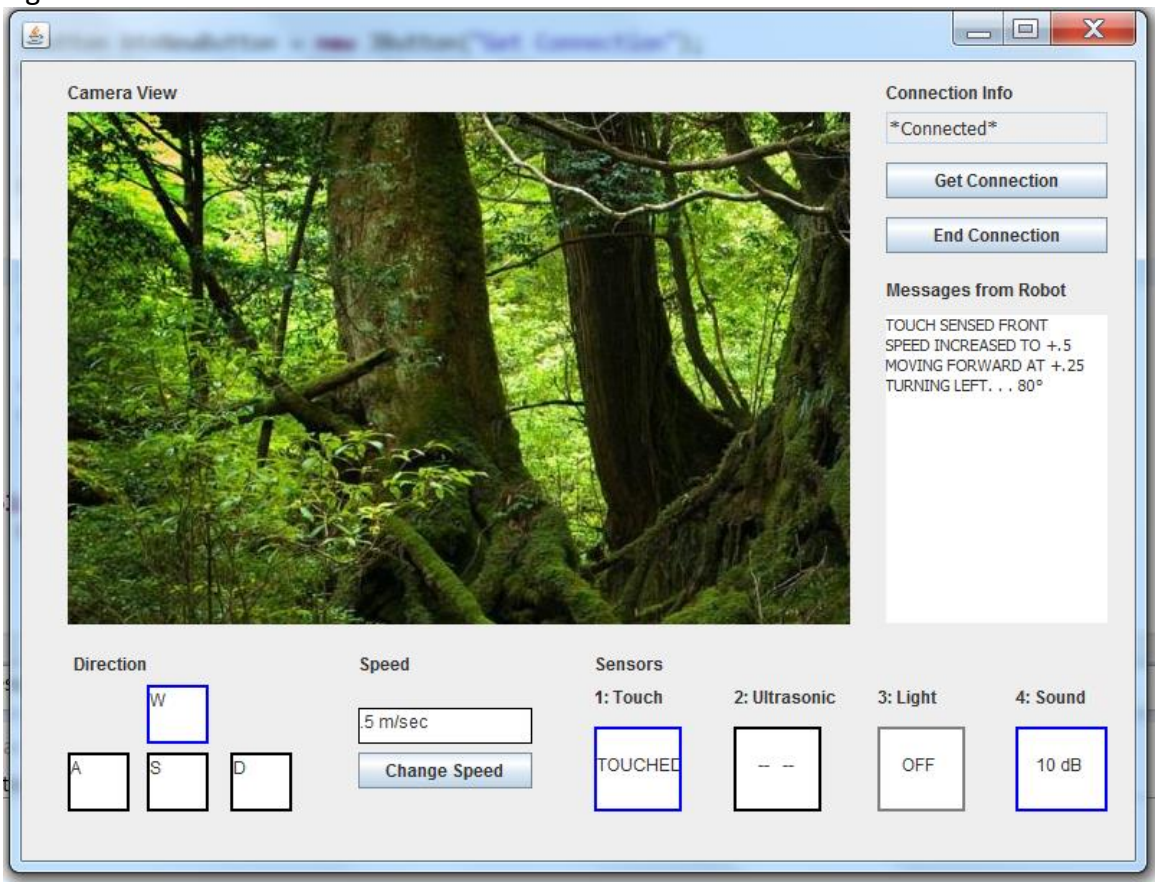
Name	Definition
Normal Operation	%connected%
Awaiting Connection	!connection! = false /button_get_connection/ = $pressed$
No Connection	!connection! = false

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%)	

User Interface

Figure 1. Robot GUI



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

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*

Onboard System and Debugger Specification

Laboratory # 2: Requirements and Specification

Work Product

The Specification Document describes the behavior of the onboard system and debugger. It includes the glossary, mode definition, mode transition table, conditions, input and output data items, and event table.

Document Revision Information

2/4/2013 – Initial shell
2/11/2013 – Rough draft
2/23/2013 – Initial editing

Approval Sheet

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

Tang, Raymond
McMillion, Andrew
Rupakhetee, Archit
Lenig, Tyler

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 a clear and concise definition 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.

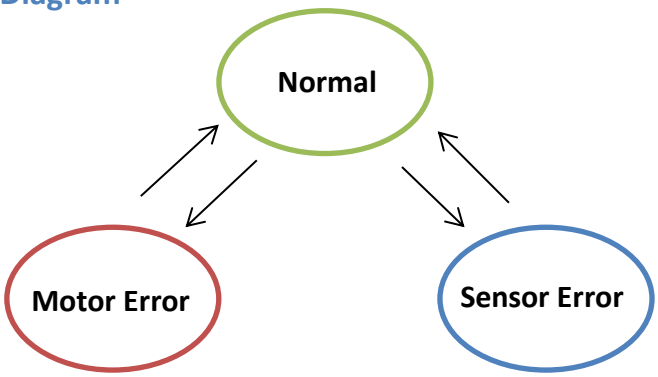
Modes

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	@T(!motor_2_speed! != 0 && //motor_2_speed// = 0)

ACTION	!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.

Conditions

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! >= $max_backward_speed$

Communications Protocol Specification

Laboratory # 4: Development Tools and Communications Protocol

Work Product

This document describes the communication protocol implemented by Teams 19 and 20 for communication between the base station control system, and the robot. This document describes the creation, and decoding process for messages.

Document Revision Information

2/15/2013 – created
2/17/2013 – designed base station to robot messages
2/22/2013 – continued design
2/24/2013 – designed robot to base station messages
3/17/2013 – added commands and error detection

Approval Sheet

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

Morgan, Laura
Miaw, Jireh
Hauser, Steven
Dworak, Catherine
Bertoglio, David

Lenig, Tyler
Tang, Raymond
Rupakhetee, Archit
McMillion, Andrew

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.

Base Station to Robot Messages

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

Error Messages

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.

Design Document

Laboratory # 5: Design

Work Product

Description of the design of the robot on-board software, including high level description, UML class and sequence diagrams, state diagram, concurrent structure, and class interfaces in Java.

Document Revision Information

3/22/2013 – Design Document Created

Approval Sheet

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

Morgan, Laura
Miaw, Jireh
Hauser, Steven
Dworak, Catherine
Bertoglio, David

High-level system architecture

The robot on-board software will be object-oriented. It will consist of 3 classes, Activator, Driver, and MessageHandler. The Activator will contain instances of Driver and MessageHandler. Driver and MessageHandler will not be able to access each other's fields and methods directly; any interaction between Driver and MessageHandler must go through the Activator class.

Activator

The Activator class contains the main method. This class is the only one that deals with the Bluetooth connection. It will contain fields and methods to create the connection and check if the connection is there. It creates 3 threads: timer, read, and output. The timer thread is used to determine how much time has elapsed between sending the last message from the on-board system and receiving an acknowledgment from the base computer. The input and output threads are the channels to send and receive messages from the base computer.

The activator receives messages from the base computer, then sends them to the MessageHandler class for decoding, then channels the usable message to the Driver class to implement the required action.

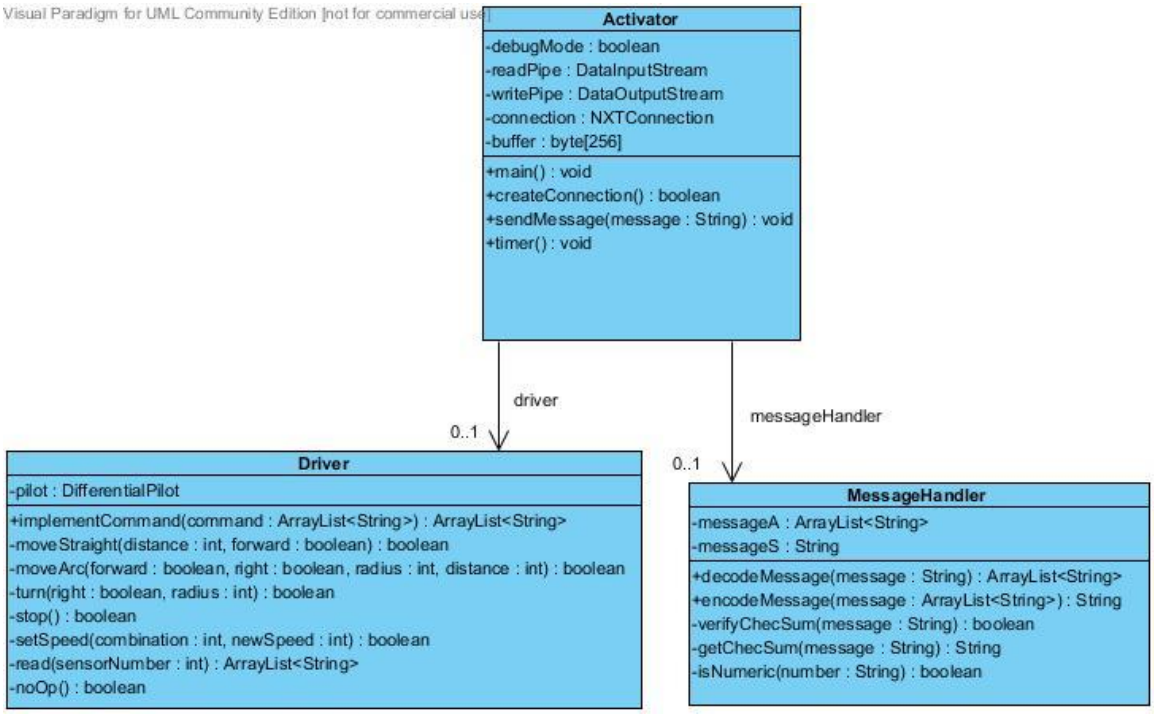
MessageHandler

The MessageHandler class has one purpose: to deal with messages. It will be capable of decoding a message from the base station, validating the checksum, endcoding a new message to send to the base station, and creating a checksum for the new message. It will take messages in the format designated by the Communications Protocol and transform them into a format that the Driver can use to perform actions. On the reverse, it will take messages (acknowledgments or sensor data), and put them into the communications protocol format, so they can be sent over the Bluetooth channel from the Activator class. All encoded and decoded messages are passed back to the Activator class, and from there are sent to their final destination.

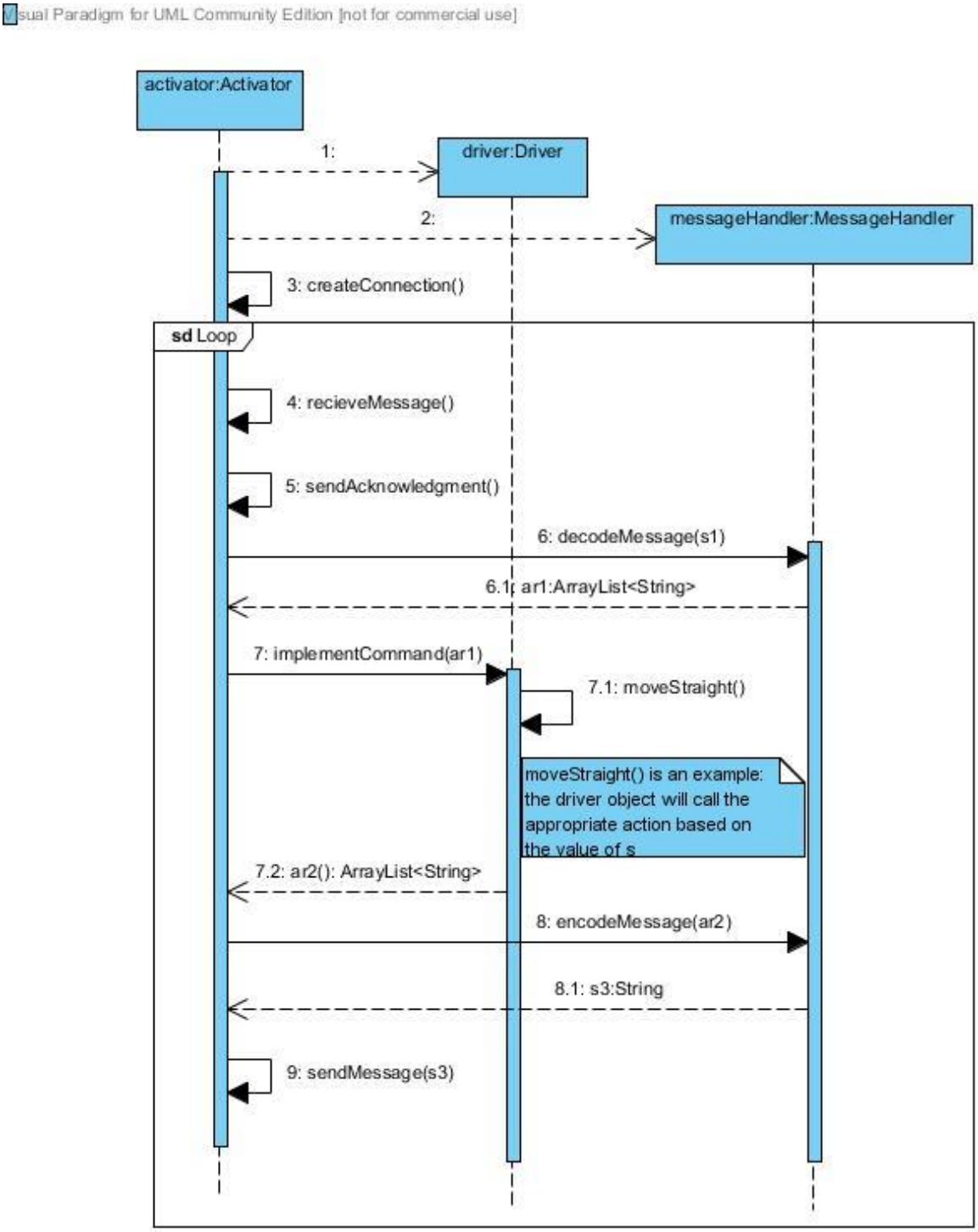
Driver

The Driver class is in charge of performing robot actions. It will contain an instance of the Differential Pilot Object from Lejos, which contains classes that control robot movement, such as setting the speed and rotating. The Driver class will contain a method for each action the robot should be able to perform: moveStraight, moveArc, turn, stop, setSpeed, read, and noOp. Additionally, it will have a method called implementAction, which will take in a decoded message and call the correct method to perform the required action.

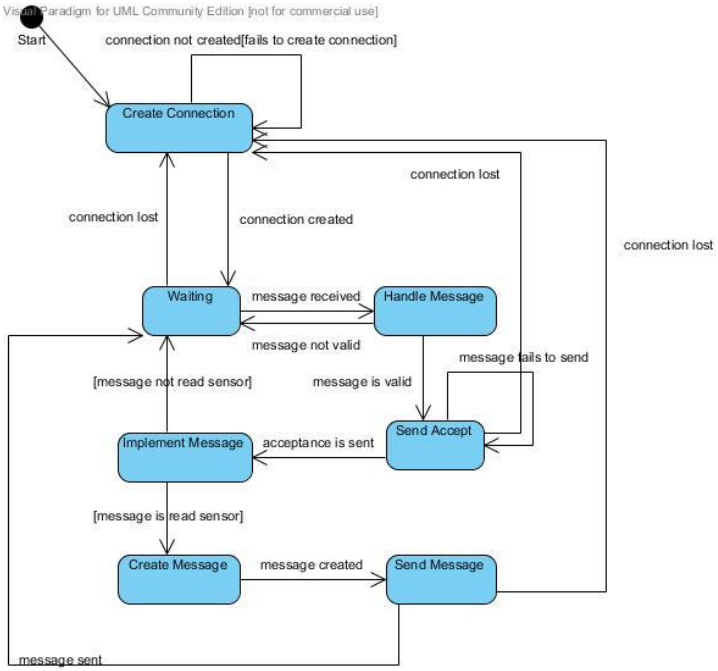
UML Class Diagram



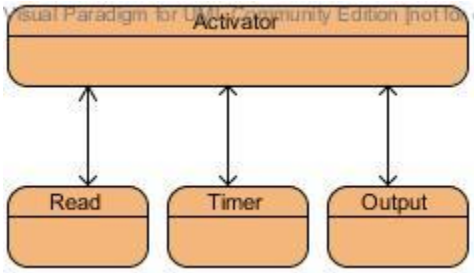
UML Sequence Diagram



Finite State Diagram



Concurrent Structure



Class interfaces

Driver

```
/*This class hides the design decisions behind how to control the
*actual functionality of the robot.
*/

public Class Driver{
   private DifferentialPilot pilot;

   //creates the DifferentialPilot
   public Driver();

   /*
   * public method that implements commands
   * command an array that breaks down each parameter available
   * for any command type
   */
   public String[] implementCommand(String[] command);

   /*
   * private method that hides how movement in a straight
   * direction works
   * boolean forward move robot forward when true, backwards when
   * false
   * distance is the distance for the robot to move
   */
   private boolean moveStraight(boolean forward, int distance);

   /*
   * private method that hides how movement in an arc works
   * boolean forward moves robot forward in arc when true, and
   * backwards when false
   * boolean right arcs the robot to the right when true, left
   * when false
   * distance determines the distance for the robot to move
   * radius determines the radius to move along
   */
   private boolean moveArc(boolean forward, boolean right, int
   distance, int radius);

   /*
   * private method that hides how turning works
   * boolean right turns the robot right when true, and left when
   * false
   * radius determines what radius in degrees to turn
   */
```

```
private boolean turn(boolean right, int radius);

/*
* private method stop abstracts how stopping works
*/
private boolean stop();

/*
* private method that hides how setting speed works
* int combination determines which motor or motor combination
* to set speed for
* newSpeed determines the new speed to set to
*/
private boolean setSpeed(int combination, int newSpeed);

/*
* private method read controls reading a sensor
* int sensor number determines the sensor to read from
*/
private ArrayList<String> read(int sensorNumber);

/*
* Does nothing, no operation
*/
private boolean noOp();
}
```

Activator

```
/*
* This class is designed to handle the connection and activating
* both driving of the robot hardware and message handling.
*/
public class Activator {
  //Driver that controls the hardware side of robot
   private Driver driver;

   //MessageHandler that creates, encodes, and decodes messages
   to be sent
   private MessageHandler messageHandler;

   //boolean used to determine whether to allow debugCommands or
   not
   private boolean debugMode;

   //Pipes for reading and writing messages to and from the base
   station
   private DataInputStream readPipe;
   private DataOutputStream writePipe;

   /*
   * NXTConnection that acts as the bluetooth connection between
   * base station
   * and robot
   */
   private NXTConnection connection;

   //buffer used for reading from the stream
   private byte[256] buffer;

   /*
   * main method that controls the creation of connection and
   * actual running
   * of the robot system
   */
   public static void main(String[] args);

   /*
   * creates the connection between robot and base station
   * allows for multiple connections to be made
   */
   public boolean createConnection();

   /*
   * method that sends message created by messageHandler to base
   * station
   * message is a message created by messageHandler
   */
```

```
    public void sendMessage(String message);

    /*
    * Method that creates the timer for checking timeouts on
    * messages
    */
    public void timer();
}
```

MessageHandler

```
/*
* This class abstracts away the implementation of the communications
protocol
* This class contains methods that are required to decode and encode
various
* messages that the robot needs to send to the base station.
*/

public Class MessageHandler{

   /*
   * decodeMessage takes a message and decodes into parameters
   * for
   * the Driver to use.
   * Parameter message is the message to be decoded
   */
   public ArrayList<String> decodeMessage(String message);

   /*
   * encodeMessage uses parameters from the Driver to create a
   * message
   * to be sent to the base station.
   * Parameter message is ArrayList of Strings to be used to
   * crease message
   */
   public String encodeMessage(ArrayList<String> message);

   /*
   * Verify checksum verifies if the calculated checksum is
   * equivalent
   * to the checksum sent in the message
   * Parameter message is String on which to check checksum
   */
   private boolean verifyChecksum(String message);

   /*
   * Calculates the checksum of the provided message
   * Parameter message is the message on which to get the
   * checksum
   */
```

```
    private String getChecksum(String message);

    /*
    * Checks to see if number is of a numeric type (i.e. it can be
    * converted to number)
    * Parameter number is the String to check whether the number
    * is a boolean
   */
    private boolean isNumeric(String number);
}
```

Software Source Code

Work Product
Listing of software source code.

Document Revision Information
4/19/13 – Phase 1 Rework
4/21/13 – Phase 2 Rework
4/26/13 – Phase 3 Rework

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

Morgan, Laura
Miaw, Jireh
Hauser, Steven
Dworak, Catherine
Bertoglio, David

Driver

```
import java.util.ArrayList;
import lejos.nxt.Button;
import lejos.nxt.LightSensor;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.SensorPortListener;
import lejos.nxt.Sound;
import lejos.nxt.SoundSensor;
import lejos.nxt.TouchSensor;
import lejos.nxt.UltrasonicSensor;
import lejos.robotics.navigation.DifferentialPilot;

public class Driver {
	private DifferentialPilot pilot;
	private final int DEFAULT_RADIUS = 90;
	private final int FORWARD_ANGLE = 1;
	private final int BACKWARD_ANGLE = -1;
	private final int COMMAND_TYPE_INDEX = 0;
	private final int PARAMETER1_INDEX = 1;
	private final int PARAMETER2_INDEX = 2;
	private final int PARAMETER3_INDEX = 3;
	private final int PARAMETER4_INDEX = 4;
	private final int SAFEDISTANCE = 50;
	private final int SOUNDTHRESHOLD = 85;
	private final int INITIAL_ROTATE_SPEED = 90;
	private final int SWING_FORWARD_ANGLE = 90;
	private final int SWING_BACKWARD_ANGLE = -90;
	private final double WHEEL_DIAMETER = 2.25f;
	private final double TRACK_WIDTH = 5.5f;
	private boolean DRIVING = false;
	private boolean hasStopped = false;
	private boolean moveBreakpoint = false;
	private boolean moveForwardBreakpoint = false;
	private boolean moveBackwardBreakpoint = false;
	private boolean arcBreakpoint = false;
	private boolean arcForwardBreakpoint = false;
	private boolean arcBackwardBreakpoint = false;
	private boolean arcForwardRightBreakpoint = false;
	private boolean arcForwardLeftBreakpoint = false;
	private boolean arcBackwardRightBreakpoint = false;
	private boolean arcBackwardLeftBreakpoint = false;
	private boolean setspeedBreakpoint = false;
	private boolean readSensorBreakpoint = false;
	private boolean readSensorTouchBreakpoint = false;
	private boolean readSensorUltrasonicBreakpoint = false;
	private boolean readSensorLightBreakpoint = false;
	private boolean readSensorMicrophoneBreakpoint = false;
	private boolean turnBreakpoint = false;
	private boolean turnRightBreakpoint = false;
	private boolean turnLeftBreakpoint = false;
```

```
private boolean swingBreakpoint = false;

private TouchSensor touchSensor;
private UltrasonicSensor ultrasonicSensor;
private LightSensor lightSensor;
private SoundSensor soundSensor;

public Driver() {
	pilot = new DifferentialPilot(WHEEL_DIAMETER, TRACK_WIDTH, Motor.B,
			Motor.C);
	pilot.setRotateSpeed(INITIAL_ROTATE_SPEED);
	Motor.A.setSpeed(INITIAL_ROTATE_SPEED);

	touchSensor = new TouchSensor(SensorPort.S1);
	ultrasonicSensor = new UltrasonicSensor(SensorPort.S2);
	lightSensor = new LightSensor(SensorPort.S3);
	soundSensor = new SoundSensor(SensorPort.S4);

}

public ArrayList<String> safetySense() {
	if (hasStopped) {
		if (ultrasonicSensor.getDistance() > SAFEDISTANCE)
			hasStopped = false;
	} else if (!hasStopped && DRIVING
			&& (ultrasonicSensor.getDistance() < SAFEDISTANCE)) {
		Sound.twoBeeps();
		hasStopped = true;
		stop();
		return readSensor("ultrasonic");
	}
	if (DRIVING && (touchSensor.isPressed())) {
		Sound.beepSequence();
		stop();
		return readSensor("touch");
	}
	if (DRIVING && (soundSensor.readValue() > SOUNDTHRESHOLD)) {
		Sound.playNote(Sound.FLUTE, 130, 500);
		stop();
		return readSensor("sound");
	} else if (!DRIVING && (soundSensor.readValue() > SOUNDTHRESHOLD)) {
		moveStraight(false, 0);
		return readSensor("sound");
	}
	return new ArrayList<String>();
}

public ArrayList<String> implementCommand(ArrayList<String> command) {
	boolean forward, right, setTravelSpeed;
	int radius, distance, speed;
	String sensor, motor;
	switch (command.get(COMMAND_TYPE_INDEX)) {
	case "straight":
		if (command.get(PARAMETER1_INDEX).equalsIgnoreCase("forward")) {
			forward = true;
```

```
		} else {
			forward = false;
		}
		distance = Integer.parseInt(command.get(PARAMETER2_INDEX));
		moveStraight(forward, distance);
		return new ArrayList<String>();
	case "turn":
		if (command.get(PARAMETER1_INDEX).equalsIgnoreCase("right")) {
			right = true;
		} else {
			right = false;
		}
		radius = Integer.parseInt(command.get(PARAMETER2_INDEX));
		turn(right, radius);
		return new ArrayList<String>();
	case "stop":
		stop();
		return new ArrayList<String>();
	case "arc":
		if (command.get(PARAMETER1_INDEX).equalsIgnoreCase("forward")) {
			forward = true;
		} else {
			forward = false;
		}
		if (command.get(PARAMETER2_INDEX).equalsIgnoreCase("right")) {
			right = true;
		} else {
			right = false;
		}
		distance = Integer.parseInt(command.get(PARAMETER3_INDEX));
		radius = Integer.parseInt(command.get(PARAMETER4_INDEX));
		moveArc(forward, right, distance, radius);
		return new ArrayList<String>();
	case "readsensor":
		sensor = command.get(PARAMETER1_INDEX);
		return readSensor(sensor);
	case "setspeed":
		motor = command.get(PARAMETER1_INDEX);
		if (command.get(PARAMETER2_INDEX).equalsIgnoreCase("travel"))
			setTravelSpeed = true;
		else
			setTravelSpeed = false;
		speed = Integer.parseInt(command.get(PARAMETER2_INDEX));
		setSpeed(motor, setTravelSpeed, speed);
		return new ArrayList<String>();
	case "swing":
		swing();
		return new ArrayList<String>();
	case "breakpoint":
		implementBreakpoint(command);
		return new ArrayList<String>();
	case "variable":
		return implementVariable(command);
	default:
		noOp();
```

```
			return new ArrayList<String>();
		}

	}

	private boolean setSpeed(String motor, boolean setTravelSpeed, int speed) {
		if (setspeedBreakpoint) {
			System.out.println("Hit Breakpoint");
			Button.ENTER.waitForPressAndRelease();
		}
		switch (motor) {
		case "motora":
			Motor.A.setSpeed(speed);
			return true;
		case "motorb":
			Motor.B.setSpeed(speed);
			return true;
		case "motorc":
			Motor.C.setSpeed(speed);
			return true;
		case "drivemotors":
			if (setTravelSpeed)
				pilot.setTravelSpeed(speed);
			else
				pilot.setRotateSpeed(speed);
			System.out.println("speed set to " + speed);
			return true;
		default:
			return false;
		}
	}

	private boolean swing() {
		if (swingBreakpoint) {
			System.out.println("Hit Breakpoint");
			Button.ENTER.waitForPressAndRelease();
		}
		Motor.A.rotateTo(SWING_FORWARD_ANGLE);
		Motor.A.rotateTo(SWING_BACKWARD_ANGLE);
		return true;
	}

	private boolean moveStraight(boolean forward, int distance) {
		DRIVING = true;
		if (moveBreakpoint) {
			System.out.println("Hit Breakpoint");
			Button.ENTER.waitForPressAndRelease();
		}
		if (forward) {
			if (moveForwardBreakpoint) {
				System.out.println("Hit Breakpoint");
				Button.ENTER.waitForPressAndRelease();
			}
			if (distance == 0) {
				pilot.forward();
```

```
                        return true;
                } else {
                        pilot.travel(distance);
                        return true;
                }
        } else {
                if (moveBackwardBreakpoint) {
                        System.out.println("Hit Breakpoint");
                        Button.ENTER.waitForPressAndRelease();
                }
                if (distance == 0) {
                        pilot.backward();
                        return true;
                } else {
                        pilot.travel(-distance);
                        return true;
                }
        }
}

private boolean moveArc(boolean forward, boolean right, int distance,
                int radius) {
        if (arcBreakpoint) {
                System.out.println("Hit Breakpoint");
                Button.ENTER.waitForPressAndRelease();
        }
        DRIVING = true;
        if (forward) {
                if (arcForwardBreakpoint) {
                        System.out.println("Hit Breakpoint");
                        Button.ENTER.waitForPressAndRelease();
                }
                if (right) {
                        if (arcForwardRightBreakpoint) {
                                System.out.println("Hit Breakpoint");
                                Button.ENTER.waitForPressAndRelease();
                        }
                        if (distance == 0 && radius == 0) {
                                pilot.arcForward(-DEFAULT_RADIUS);
                                return true;
                        } else {
                                pilot.arc(-radius, FORWARD_ANGLE);
                                return true;
                        }
                } else {
                        if (arcForwardLeftBreakpoint) {
                                System.out.println("Hit Breakpoint");
                                Button.ENTER.waitForPressAndRelease();
                        }
                        if (distance == 0 && radius == 0) {
                                pilot.arcForward(DEFAULT_RADIUS);
                                return true;
                        } else {
                                pilot.arc(radius, FORWARD_ANGLE);
                                return true;
```

```
                }
            }
        } else {
            if (arcBackwardBreakpoint) {
                System.out.println("Hit Breakpoint");
                Button.ENTER.waitForPressAndRelease();
            }
            if (right) {
                if (arcBackwardRightBreakpoint) {
                    System.out.println("Hit Breakpoint");
                    Button.ENTER.waitForPressAndRelease();
                }
                if (distance == 0 && radius == 0) {
                    pilot.arcBackward(-DEFAULT_RADIUS);
                    return true;
                } else {
                    pilot.arc(-radius, BACKWARD_ANGLE);
                    return true;
                }
            } else {
                if (arcBackwardLeftBreakpoint) {
                    System.out.println("Hit Breakpoint");
                    Button.ENTER.waitForPressAndRelease();
                }
                if (distance == 0 && radius == 0) {
                    pilot.arcBackward(DEFAULT_RADIUS);
                    return true;
                } else {
                    pilot.arc(radius, BACKWARD_ANGLE);
                    return true;
                }
            }
        }
    }

    private boolean turn(boolean right, int radius) {
        if (turnBreakpoint) {
            System.out.println("Hit Breakpoint");
            Button.ENTER.waitForPressAndRelease();
        }
        DRIVING = true;
        if (right) {
            if (turnRightBreakpoint) {
                System.out.println("Hit Breakpoint");
                Button.ENTER.waitForPressAndRelease();
            }
            if (radius == 0) {
                pilot.rotateRight();
                return true;
            } else {
                pilot.rotate(radius);
                return true;
            }
        } else {
            if (turnLeftBreakpoint) {
```

```
                        System.out.println("Hit Breakpoint");
                        Button.ENTER.waitForPressAndRelease();
                }
                if (radius == 0) {
                        pilot.rotateLeft();
                        return true;
                } else {
                        pilot.rotate(-radius);
                        return true;
                }
        }
}

private boolean stop() {
        pilot.stop();
        DRIVING = false;
        return true;
}

private boolean noOp() {
        return true;
}

private ArrayList<String> readSensor(String sensorType) {
        if (readSensorBreakpoint) {
                System.out.println("Hit Breakpoint");
                Button.ENTER.waitForPressAndRelease();
        }
        ArrayList<String> returnList = new ArrayList<String>();
        returnList.add(sensorType);
        switch (sensorType) {
        case "touch":
                if (readSensorTouchBreakpoint) {
                        System.out.println("Hit Breakpoint");
                        Button.ENTER.waitForPressAndRelease();
                }
                if (touchSensor.isPressed())
                        returnList.add("1");
                else
                        returnList.add("0");
                break;
        case "ultrasonic":
                if (readSensorUltrasonicBreakpoint) {
                        System.out.println("Hit Breakpoint");
                        Button.ENTER.waitForPressAndRelease();
                }
                returnList.add(Integer.toString(ultrasonicSensor.getDistance()));
                break;
        case "sound":
                if (readSensorMicrophoneBreakpoint) {
                        System.out.println("Hit Breakpoint");
                        Button.ENTER.waitForPressAndRelease();
                }
                returnList.add(Integer.toString(soundSensor.readValue()));
                break;
```

```
		case "light":
			if (readSensorLightBreakpoint) {
				System.out.println("Hit Breakpoint");
				Button.ENTER.waitForPressAndRelease();
			}
			returnList.add(Integer.toString(lightSensor
					.getNormalizedLightValue()));
			break;
		default:
			return new ArrayList<String>();
		}
		return returnList;
	}

	private void implementBreakpoint(ArrayList<String> command) {
		boolean breakpointValue;
		int lastIndex = command.size() - 1;
		switch (command.get(lastIndex)) {
		case "true":
			breakpointValue = true;
			break;
		default:
			breakpointValue = false;
			break;
		}
		if (command.size() > 2) {
			switch (command.get(PARAMETER1_INDEX)) {
			case "move":
				if (command.size() > 3) {
					switch (command.get(PARAMETER2_INDEX)) {
					case "forward":
						moveForwardBreakpoint = breakpointValue;
						break;
					case "backward":
						moveBackwardBreakpoint = breakpointValue;
						break;
					default:
						break;
					}
					break;
				}
				moveBreakpoint = breakpointValue;
				break;
			case "arc":
				if (command.size() > 3) {
					switch (command.get(PARAMETER2_INDEX)) {
					case "forward":
						if (command.size() > 4) {
							switch (command.get(PARAMETER3_INDEX))
							{
							case "right":
								arcForwardRightBreakpoint =
								breakpointValue;
								break;
							case "left":
```

```
                                        arcForwardLeftBreakpoint =
                                        breakpointValue;
                                        break;
                                default:
                                        break;
                                }
                                break;
                        }
                        arcForwardBreakpoint = breakpointValue;
                        break;
                case "backward":
                        if (command.size() > 4) {
                                switch (command.get(PARAMETER3_INDEX))
                                {
                                case "right":
                                        arcBackwardRightBreakpoint =
                                        breakpointValue;
                                        break;
                                case "left":
                                        arcBackwardLeftBreakpoint =
                                        breakpointValue;
                                        break;
                                default:
                                        break;
                                }
                                break;
                        }
                        arcBackwardBreakpoint = breakpointValue;
                        break;
                default:
                        break;
                }
        }
case "turn":
        if (command.size() > 3) {
                switch (command.get(PARAMETER2_INDEX)) {
                case "right":
                        turnRightBreakpoint = breakpointValue;
                        break;
                case "left":
                        turnLeftBreakpoint = breakpointValue;
                        break;
                default:
                        break;
                }
                break;
        }
        turnBreakpoint = breakpointValue;
        break;
case "speed":
        setspeedBreakpoint = breakpointValue;
        break;
case "sensor":
        if (command.size() > 3) {
                switch (command.get(PARAMETER2_INDEX)) {
```

```
                        case "touch":
                                readSensorTouchBreakpoint = breakpointValue;
                                break;
                        case "ultrasonic":
                                readSensorUltrasonicBreakpoint =
                                breakpointValue;
                                break;
                        case "microphone":
                                readSensorMicrophoneBreakpoint =
                                breakpointValue;
                                break;
                        case "light":
                                readSensorLightBreakpoint = breakpointValue;
                                break;
                        default:
                                break;
                        }
                        break;
                }
                readSensorBreakpoint = breakpointValue;
                break;
        case "swing":
                swingBreakpoint = breakpointValue;
                break;
        default:
                break;
        }
    }
}

private ArrayList<String> implementVariable(ArrayList<String> command) {
    ArrayList<String> messageData = new ArrayList<String>();
    String speed;
    switch (command.get(PARAMETER1_INDEX)) {
    case "drive":
        messageData.add("drive");
        switch(command.get(PARAMETER2_INDEX)){
        case "rotate":
            messageData.add("rotate");
            speed = Integer.toString((int) pilot.getRotateSpeed());
            messageData.add(speed);
            break;
        case "travel":
            messageData.add("travel");
            speed = Integer.toString((int) pilot.getTravelSpeed());
            messageData.add(speed);
            break;
        default:
            return new ArrayList<String>();
        }
        break;
    case "swing":
        messageData.add("swing");
        messageData.add("rotate");
        messageData.add("" + Motor.C.getSpeed());
```

```
                        break;
                default:
                        return new ArrayList<String>();
                }
                return messageData;
        }
}
```

MessageHandler

```
import java.util.ArrayList;

public class MessageHandler {
	static final int MESSAGE_LENGTH = 11;
	static final int COMMAND_LENGTH = 10;
	static final int CHECKSUM_INDEX = 10;
	static final int START_INDEX = 0;
	static final int COMMAND_TYPE_END_INDEX = 2;
	static final int SENSOR_TYPE_INDEX = 0;
	static final int SENSOR_VALUE_INDEX = 1;
	static final int SENSOR_DATA_SIZE = 2;
	static final int VARIABLE_DATA_SIZE = 3;

	public MessageHandler() {

	}

	public String createACK() {
		System.out.println("ACK created");
		String ack = "AK00000000";
		ack += getChecksum(ack);
		return ack;
	}

	public String encodeMessage(ArrayList<String> messageData) {
		if (messageData.size() == SENSOR_DATA_SIZE) {
			String encoded = "SD";
			String value = messageData.get(SENSOR_VALUE_INDEX);
			switch (messageData.get(SENSOR_TYPE_INDEX)) {
			case "touch":
				encoded += "T";
				break;
			case "sound":
				encoded += "M";
				break;
			case "ultrasonic":
				encoded += "U";
				break;
			case "light":
				encoded += "L";
				break;
			default:
				encoded = "";
				break;
			}
			if (isNumeric(value)) {
				encoded += getPadding(encoded.length(), value.length()) +
				value;
			}
			encoded += getChecksum(encoded);
			return encoded;
		} else if (messageData.size() == VARIABLE_DATA_SIZE) {
```

```
                String encoded = "VA";
                String speed = messageData.get(2);
                switch (messageData.get(0)) {
                case "drive":
                        encoded += "D";
                        break;
                case "swing":
                        encoded += "S";
                        break;
                default:
                        return "";
                }
                switch (messageData.get(1)) {
                case "rotate":
                        encoded += "R";
                        break;
                case "travel":
                        encoded += "T";
                        break;
                default:
                        return "";
                }
                if(isNumeric(speed)){
                        encoded += getPadding(encoded.length(), speed.length()) +
                        speed;
                }
                encoded += getChecksum(encoded);
                return encoded;
        } else
                return "";
}

// utility method to determine whether a String is a number
private boolean isNumeric(String number) {
        try {
                int i = Integer.parseInt(number);
        } catch (NumberFormatException nfe) {
                return false;
        }
        return true;
}

public boolean isAck(String message) {
        if (message.length() < 2)
                return false;
        else
                return message.charAt(0) == 'A' && message.charAt(1) == 'K';
}

// utility method to determine whether the parameters is empty space by
// communications protocol
private boolean isEmptyZeros(String parameters) {
        if (Integer.parseInt(parameters) == 0) {
                return true;
        } else {
```

```
			return false;
		}
	}

	private boolean verifyChecksum(String message) {
		if (message.length() == MESSAGE_LENGTH) {
			byte[] string = message.getBytes();
			if (getChecksum(message.substring(START_INDEX, COMMAND_LENGTH))
					.equals(message.substring(CHECKSUM_INDEX))) {
				return true;
			}
		}
		return false;
	}

	private String getPadding(int messageHeadingLength, int messageTailLength) {
		String returnString = "";
		for (int i = 1; i < MESSAGE_LENGTH - messageHeadingLength
				- messageTailLength; i++) {
			returnString += "0";
		}
		return returnString;
	}

	private String getChecksum(String message) {
		int sum = 0;
		String ret;
		byte[] buffer = message.getBytes();
		for (int i = 0; i < buffer.length; i++) {
			sum += (int) buffer[i];
		}
		sum = sum % 256;
		byte[] checksum = new byte[1];
		checksum[START_INDEX] = (byte) sum;
		ret = new String(checksum);
		return ret;
	}

	public ArrayList<String> decodeMessage(String message) {
		ArrayList<String> commandData = new ArrayList<String>();
		if (message.length() != MESSAGE_LENGTH) {
			return commandData;
		} else {
			if (verifyChecksum(message)) {
				String command = message.substring(START_INDEX,
						COMMAND_TYPE_END_INDEX);
				String params = message.substring(COMMAND_TYPE_END_INDEX,
						COMMAND_LENGTH);
				switch (command) {
				case "MS":
					commandData = decodeMoveStraight(params);
					return commandData;
				case "MA":
					commandData = decodeMoveArc(params);
					return commandData;
```

```
                        case "TN":
                                commandData = decodeTurn(params);
                                return commandData;
                        case "ST":
                                commandData = decodeStop(params);
                                return commandData;
                        case "RS":
                                commandData = decodeReadSensor(params);
                                return commandData;
                        case "SS":
                                commandData = decodeSetSpeed(params);
                                return commandData;
                        case "RA":
                                break;
                        case "SW":
                                commandData = decodeSwing(params);
                        case "EC":
                                commandData.add("exit");
                                return commandData;
                        case "DM":
                                commandData = decodeDebugMessage(params);
                                return commandData;
                        default:
                                return new ArrayList<String>();
                        }
                } else {
                        System.out.println("Invalid Checksum");
                }
        }
        return new ArrayList<String>();
}

private ArrayList<String> decodeMoveStraight(String parameters) {
        ArrayList<String> commandData = new ArrayList<String>();
        commandData.add("straight");
        String direction = parameters.substring(0, 1);
        switch (direction) {
        case "F":
                commandData.add("forward");
                break;
        case "B":
                commandData.add("backward");
                break;
        default:
                return new ArrayList<String>();
        }

        String distance = parameters.substring(1);
        if (isNumeric(distance)) {
                commandData.add(distance);
        } else {
                return new ArrayList<String>();
        }
        return commandData;
}
```

```
private ArrayList<String> decodeTurn(String parameters) {
	ArrayList<String> commandData = new ArrayList<String>();
	commandData.add("turn");
	String direction = parameters.substring(0, 1);
	switch (direction) {
	case "R":
		commandData.add("right");
		break;
	case "L":
		commandData.add("left");
		break;
	default:
		return new ArrayList<String>();
	}
	String radius = parameters.substring(1);
	if (isNumeric(radius)) {
		commandData.add(radius);
	} else {
		return new ArrayList<String>();
	}
	return commandData;
}

private ArrayList<String> decodeStop(String parameters) {
	ArrayList<String> commandData = new ArrayList<String>();
	commandData.add("stop");
	if (!(isNumeric(parameters) && isEmptyZeros(parameters))) {
		return new ArrayList<String>();
	}
	return commandData;
}

private ArrayList<String> decodeMoveArc(String parameters) {
	ArrayList<String> commandData = new ArrayList<String>();
	commandData.add("arc");
	String direction = parameters.substring(0, 1);
	switch (direction) {
	case "F":
		commandData.add("forward");
		break;
	case "B":
		commandData.add("backward");
		break;
	default:
		return new ArrayList<String>();
	}

	String turn = parameters.substring(1, 2);
	switch (turn) {
	case "R":
		commandData.add("right");
		break;
	case "L":
		commandData.add("left");
```

```
			break;
		default:
			return new ArrayList<String>();
		}

		String distance = parameters.substring(4, 7);
		if (isNumeric(distance)) {
			commandData.add(distance);
		} else {
			return new ArrayList<String>();
		}

		String radius = parameters.substring(7);
		if (isNumeric(radius)) {
			commandData.add(radius);
		} else {
			return new ArrayList<String>();
		}
		return commandData;
	}

	private ArrayList<String> decodeReadSensor(String parameters) {
		ArrayList<String> commandData = new ArrayList<String>();
		commandData.add("readsensor");
		String direction = parameters.substring(0, 1);
		switch (direction) {
		case "U":
			commandData.add("ultrasonic");
			break;
		case "T":
			commandData.add("touch");
			break;
		case "M":
			commandData.add("sound");
			break;
		case "L":
			commandData.add("light");
			break;
		default:
			return new ArrayList<String>();
		}

		return commandData;
	}

	private ArrayList<String> decodeSetSpeed(String parameters) {
		ArrayList<String> commandData = new ArrayList<String>();
		commandData.add("setspeed");
		switch (parameters.substring(0, 1)) {
		case "A":
			commandData.add("motora");
			break;
		case "B":
			commandData.add("motorb");
			break;
```

```
		case "C":
			commandData.add("motorc");
			break;
		case "D":
			commandData.add("drivemotors");
			break;
		default:
			return new ArrayList<String>();
		}
		switch (parameters.substring(1, 2)) {
		case "T":
			commandData.add("travel");
			break;
		case "R":
			commandData.add("rotate");
			break;
		default:
			return new ArrayList<String>();

		}
		String speed = parameters.substring(2);
		if (isNumeric(speed)) {
			commandData.add(speed);
		} else {
			return new ArrayList<String>();
		}
		return commandData;
	}

	private ArrayList<String> decodeSwing(String parameters) {
		ArrayList<String> commandData = new ArrayList<String>();
		commandData.add("swing");
		if (!(isNumeric(parameters) && isEmptyZeros(parameters))) {
			return new ArrayList<String>();
		}
		return commandData;
	}

	private ArrayList<String> decodeDebugMessage(String parameters) {
		String debugCommand = parameters.substring(START_INDEX,
				COMMAND_TYPE_END_INDEX);
		ArrayList<String> commandData = new ArrayList<String>();
		switch (debugCommand) {
		case "SM":
			commandData = decodeDebugMode(parameters
					.substring(COMMAND_TYPE_END_INDEX));
			break;
		case "SB":
			commandData = decodeSetBreakpointMessage(parameters
					.substring(COMMAND_TYPE_END_INDEX));
			break;
		case "PV":
			commandData = decodePrintVariableMessage(parameters
					.substring(COMMAND_TYPE_END_INDEX));
			break;
```

```
		default:
			return new ArrayList<String>();
		}
		return commandData;
	}

	private ArrayList<String> decodeDebugMode(String parameters) {
		ArrayList<String> commandData = new ArrayList<String>();
		commandData.add("mode");
		int modeValue = Integer.parseInt(parameters);
		switch (modeValue) {
		case 0:
			commandData.add("false");
			break;
		case 1:
			commandData.add("true");
			break;
		default:
			return new ArrayList<String>();
		}
		return commandData;
	}

	private ArrayList<String> decodeSetBreakpointMessage(String parameters) {
		ArrayList<String> commandData = new ArrayList<String>();
		commandData.add("breakpoint");
		String breakpointMethod = parameters.substring(START_INDEX,
				COMMAND_TYPE_END_INDEX);
		switch (breakpointMethod) {
		case "MV":
			commandData.add("move");
			switch (parameters.substring(COMMAND_TYPE_END_INDEX, 3)) {
			case "0":
				break;
			case "F":
				commandData.add("forward");
				break;
			case "B":
				commandData.add("backward");
				break;
			default:
				return new ArrayList<String>();
			}
			break;
		case "MA":
			commandData.add("arc");
			switch (parameters.substring(COMMAND_TYPE_END_INDEX, 3)) {
			case "F":
				commandData.add("forward");
				break;
			case "B":
				commandData.add("backward");
				break;
			case "0":
				break;
```

```
		default:
			return new ArrayList<String>();
		}
		switch (parameters.substring(3, 4)) {
		case "R":
			commandData.add("right");
			break;
		case "L":
			commandData.add("left");
			break;
		case "0":
			break;
		default:
			return new ArrayList<String>();
		}
		break;
	case "SS":
		commandData.add("speed");
		break;
	case "RS":
		commandData.add("sensor");
		switch (parameters.substring(COMMAND_TYPE_END_INDEX, 3)) {
		case "T":
			commandData.add("touch");
			break;
		case "U":
			commandData.add("ultrasonic");
			break;
		case "M":
			commandData.add("microphone");
			break;
		case "L":
			commandData.add("light");
			break;
		case "0":
			break;
		default:
			return new ArrayList<String>();
		}
		break;
	case "TN":
		commandData.add("turn");
		switch (parameters.substring(COMMAND_TYPE_END_INDEX, 3)) {
		case "R":
			commandData.add("right");
			break;
		case "L":
			commandData.add("left");
			break;
		case "0":
			break;
		default:
			return new ArrayList<String>();
		}
		break;
```

```
			case "SW":
				commandData.add("swing");
				break;
			default:
				return new ArrayList<String>();
			}
			switch (parameters.substring(parameters.length() - 1)) {
			case "1":
				commandData.add("true");
				break;
			case "0":
				commandData.add("false");
				break;
			default:
				return new ArrayList<String>();
			}
			return commandData;
		}

		private ArrayList<String> decodePrintVariableMessage(String parameters) {
			ArrayList<String> commandData = new ArrayList<String>();
			commandData.add("variable");
			String motor = parameters.substring(START_INDEX, 2);
			switch (motor) {
			case "DR":
				commandData.add("drive");
				String speedType = parameters.substring(2, 4);
				switch (speedType) {
				case "TR":
					commandData.add("travel");
					break;
				case "RO":
					commandData.add("rotate");
					break;
				}
				break;
			case "SW":
				commandData.add("swing");
				commandData.add("rotate");
				break;
			default:
				return new ArrayList<String>();
			}
			return commandData;

		}
	}
```

Activator

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.util.ArrayList;

import lejos.nxt.comm.Bluetooth;
import lejos.nxt.comm.NXTConnection;
import lejos.nxt.comm.USB;
import lejos.util.Timer;
import lejos.util.TimerListener;

public class Activator extends Object {

	private static boolean debugMode = false;
	private static boolean usbTest = false;

	private static NXTConnection connection;
	private static byte[] buffer = new byte[256];
	private static DataInputStream readPipe;
	private static DataOutputStream writePipe;
	private static Driver driver;
	private static MessageHandler messageHandler;
	private static ArrayList<ArrayList<String>> storedCommands;

	public static void main(String[] args) {
		boolean connected = false;
		do {
			connected = createConnection();
		} while (connected == false);

		driver = new Driver();
		messageHandler = new MessageHandler();
		readPipe = connection.openDataInputStream();
		writePipe = connection.openDataOutputStream();
		storedCommands = new ArrayList<ArrayList<String>>();
		String input = "";

		(new Thread() {
			public void run() {
				while (true) {
					ArrayList<String> sensorData = driver.safetySense();
					if (sensorData.size() > 1) {
				sendMessage(messageHandler.encodeMessage(sensorData));
					}
				}
			}
		}).start();

		do {
			try {
				int count = readPipe.read(buffer);
				if (count > 0) {
```

```
                        input = (new String(buffer)).substring(0, 11);
                        if (!messageHandler.isAck(input)) {
                              System.out.println("Received:\n" + input);
                              ArrayList<String> commandData =
                              messageHandler.decodeMessage(input);
                              if (commandData.size() < 1) {
                                    System.out.println("Invalid Message");
                              } else {
                              sendMessage(messageHandler.createACK());
                              }
                              if (commandData.get(0).equals("exit"))
                                    System.exit(0);
                              if
                              (commandData.get(0).equalsIgnoreCase("mode"))
                                    debugMode =
                                    Boolean.parseBoolean(commandData.get(1)
                                    );
                              ArrayList<String> sensorData = driver
                                          .implementCommand(commandData);
                              if (sensorData.size() > 1) {
                                    sendMessage(messageHandle
                                    .encodeMessage(sensorData));
                              }
                        }
                  }
                  Thread.sleep(10);
            } catch (Exception e) {

            }
      } while (!input.equals("exit"));
}

public static boolean createConnection() {
      System.out.println("Waiting on Connection...");
      if (usbTest) {
            connection = USB.waitForConnection();
      } else {
            connection = Bluetooth.waitForConnection();
      }
      if (connection != null) {
            System.out.println("Connected!");
            return true;
      }
      System.out.println("Failed to Connect");
      return false;
}

public static void sendMessage(String message) {
      try {
            System.out.println("Send Message");
            System.out.println(message);
            writePipe.write(message.getBytes());
            writePipe.flush();
      }
```

```
                catch (Exception e) {
                        System.out.println("Write error: " + e.getMessage());
                }
        }
}
```

Debugger

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

import lejos.pc.comm.NXTComm;
import lejos.pc.comm.NXTCommException;
import lejos.pc.comm.NXTCommFactory;
import lejos.pc.comm.NXTInfo;

public class Debugger {

	private static DebuggerShell shell;
	private boolean isConnected;
	private Thread readThread;

	private static NXTComm connection;
	private static boolean USBTest = false;
	private static NXTInfo[] info;
	private static long start;
	private static long latency;

	final static int MESSAGE_LENGTH = 11;
	final static int DATA_START = 2;
	final static int NXT_PIN = 1234;
	final static int COMMAND_PARAMETER = 0;
	final static int PARAMETER1_INDEX = 1;
	final static int PARAMETER2_INDEX = 2;
	final static int ONLY_COMMAND_LENGTH = 1;
	final static int COMMAND_AND_PARAM_LENGTH = 2;
	static Boolean readFlag = true;
	static Object lock = new Object();
	private OutputStream os;
	private InputStream is;
	private DataOutputStream oHandle;
	private DataInputStream iHandle;
	private static byte[] buffer = new byte[256];

	public Debugger() throws NXTCommException {
		shell = new DebuggerShell(this);
		this.isConnected = false;
	}

	public boolean isConnected() {
		return this.isConnected;
	}

	public void readFromRobot() {
		readThread = new Thread() {
			public void run() {
				try {
					while (true) {
```

```
                                int count = iHandle.read(buffer);
                                if (count > 0) {
                                        String input = (new
                                        String(buffer)).substring(0,
                                                        MESSAGE_LENGTH);
                                        shell.printRobotMessage("Message from
                                        robot: "+ input);
                                        if (input.substring(0,
                                        2).equalsIgnoreCase("SD")) {
                                        shell.setSensorValue(input.charAt(DATA_
                                        START)+ "",
                                        Integer.parseInt(input.substring(
                                        3, 10)));
                                        }
                                        if (input.substring(0,
                                        2).equalsIgnoreCase("VA")) {
                                        shell.printRobotMessage(decodeVariable(
                                        input));
                                        }
                                        if (!input.contains("AK00000000")) {
                                                sendMessage("AK00000000");
                                        }
                                }
                        }
                } catch (Exception e) {
                        shell.printErrorMessage("Error Reading from Robot");
                }
        }
    };
    readThread.start();
}

public void stopReading() {
    if (readThread == null)
        return;
    else if (readThread.isAlive())
        readThread.stop();
}

public void establishConnection() {
    Thread t = new Thread() {
        public void run() {
            start = System.currentTimeMillis();
            try {
                if (USBTest) {
                    connection = NXTCommFactory
                            .createNXTComm(NXTCommFactory.USB);
                    info = connection.search(null, 0);
                } else {
                    connection = NXTCommFactory
                    .createNXTComm(NXTCommFactory.BLUETOOTH);
                    info = connection.search("NXT", NXT_PIN);
                }
                if (info.length == 0) {
                    shell.printMessage("Unable to find device");
```

```
                                return;
                        }

                        connection.open(info[0]);
                        os = connection.getOutputStream();
                        is = connection.getInputStream();

                        oHandle = new DataOutputStream(os);
                        iHandle = new DataInputStream(is);
                        latency = System.currentTimeMillis() - start;
                        shell.printRobotMessage("Connection is established
                        after "+ latency + "ms.");
                        isConnected = true;
                        readFromRobot();
                        sendMessage("DMSM000001");

                } catch (Exception e) {
                        shell.printErrorMessage("Connection failed to
                        establish.");
                }
        }
    };
    shell.printMessage("Establishing Connection...");
    t.start();
}

public void endConnection() {
    shell.printMessage("Ending Connection...");
    sendMessage("DMSM000000");
    sendMessage("EC00000000");
    stopReading();
    isConnected = false;
    shell.printRobotMessage("Disconnected from robot");
}

public void sendMessage(String message) {
    if (!isConnected) {
        shell.printErrorMessage("Not connected to NXT!");
        return;
    }
    try {
        message += getCheckSum(message);
        shell.printMessage("Sending message: \"" + message + "\"");
        oHandle.write(message.getBytes());
        oHandle.flush();
    } catch (Exception e) {

    }
}

public static boolean isNumeric(String number) {
    try {
        int i = Integer.parseInt(number);
    } catch (NumberFormatException nfe) {
        return false;
```

```
        }
        return true;
    }

    public static String getCheckSum(String str) {
        int sum = 0;
        String ret;
        byte[] buffer = str.getBytes();
        for (int i = 0; i < buffer.length; i++) {
            sum += (int) buffer[i];
        }
        sum = sum % 256;
        byte[] checksum = new byte[1];
        checksum[0] = (byte) sum;
        ret = new String(checksum);
        return ret;
    }

    public static String getCommandHelp() {

        return "To form each the commands follow the directions below.
        Arguments are sepearated by spaces. Case does not matter."
        + " \n\nMOVE: Type: 'move [specify direction: forward or backward]
        [distance in cm]' \nFor example to move forward 120 cm, type: move
        forward 120"
        + " \n\nARC: Type: 'arc [specify direction: forward or backward]
        [specify direction to arc in: left of right].\nFor example, to arc up
        and right, type: arc forward right"
        + " \n\nTURN: Type: 'turn [specify direction: right or left] [specify
        number of degrees to turn]"
        + " \n\nSTOP: Type: 'stop'"
        + " \n\nSET SPEED: Type: 'setspeed [specify: a, b, c, or d, representing
        motor a, b, c, or drive] [t or r for type of speed] [number of new
        speed].\nFor example to set motor a to speed 30 type: setspeed a 30"
        + " \n\nREAD: Type: 'read [specify: u, t, m, l, or all, for ultrasonic,
        touch, microphone, light, or all sensor information respectively].
        \nFor example to read information from the light sensor type: read l.
        \nTo read information from all sensors type: read all"
        + " \n\nNONE: To create NoOp message type: none"
        + " \n\nSWING: Type: 'swing'"
        + " \n\nBREAKPOINTS: Type: setbreakpoint or removebreakpoint followed
        by: "
        + " \n\t [move, arc, read, setspeed, swing] optionally add arguments for
        those methods "
        + " \n\nPRINT VARIABLES: Type: print [drive or swing for which speed]
        [rotate or travel for speed type]";
    }

    public void runCommand(String command) {
        if (command.equalsIgnoreCase("help") || command.equalsIgnoreCase("?")) {
            shell.printMessage(getCommandHelp());
        } else if (!isConnected) {
            shell.printErrorMessage("Not connected to NXT!");
        } else if (command.equalsIgnoreCase("exit")) {
            endConnection();
```

```
		} else {
			String message = createCommand(command);
			sendMessage(message);
		}

	}

	private static String addPaddingZeros(String command, String endOfCommand) {
		for (int i = command.length() - 1; i < 9 - endOfCommand.length(); i++) {
			command += "0";
		}
		return command + endOfCommand;
	}

	private static String addTrailingZeros(String message) {
		if (message.length() >= 10)
			return message;
		else {
			while (message.length() < 10) {
				message += "0";
			}
			return message;
		}
	}

	public static String createCommand(String cmd) {
		String message = "";
		String[] cmdWords = cmd.toLowerCase().split(" ");
		String command = getCommand(cmdWords);
		String[] args = getCommandArguments(cmdWords);

		if (cmdWords.length < ONLY_COMMAND_LENGTH) {
			return message;
		} else {
			if (command.equalsIgnoreCase("ack")) {
				message = "AK00000000";
			} else if (command.equalsIgnoreCase("move")) {
				message = createMoveMessage(args);
			} else if (command.equalsIgnoreCase("arc")) {
				message = createArcMessage(args);
			} else if (command.equalsIgnoreCase("turn")) {
				message = createTurnMessage(args);
			} else if (command.equalsIgnoreCase("stop")) {
				message = createStopMessage();
			} else if (command.equalsIgnoreCase("setspeed")) {
				message = createSetSpeedMessage(args);
			} else if (command.equalsIgnoreCase("read")) {
				message = createReadSensorMessage(args);
			} else if (command.equalsIgnoreCase("none")) {
				message = createNoOpMessage();
			} else if (command.equalsIgnoreCase("swing")) {
				message = createSwingMessage();
			} else if (command.equalsIgnoreCase("setbreakpoint")) {
				message = createBreakpointMessage(args, true);
			} else if (command.equalsIgnoreCase("removebreakpoint")) {
```

```
                        message = createBreakpointMessage(args, false);
                } else if (command.equalsIgnoreCase("print")) {
                        message = createPrintVariableMessage(args);
                }
        }
        return message;
}

public static String[] getCommandArguments(String[] words) {
        String[] args = new String[words.length - 1];
        for (int i = 1; i < words.length; i++) {
                args[i - 1] = words[i];
        }
        return args;
}

public static String getCommand(String[] words) {
        return words[COMMAND_PARAMETER];
}

public static String createMoveMessage(String[] args) {
        String command = "MS";
        if (args.length < ONLY_COMMAND_LENGTH) {
                return "";
        } else {
                if (args[COMMAND_PARAMETER].equalsIgnoreCase("forward")
                || args[COMMAND_PARAMETER].equalsIgnoreCase("forwards")) {
                        command += "F";
                } else if (args[COMMAND_PARAMETER].equalsIgnoreCase("backward")
                || args[COMMAND_PARAMETER].equalsIgnoreCase("backwards")) {
                        command += "B";
                }
                if (args.length == COMMAND_AND_PARAM_LENGTH) {
                        if (isNumeric(args[PARAMETER1_INDEX])) {
                        for (int i = 0; i < 7 - args[PARAMETER1_INDEX].length();
                        i++) {
                                        command += "0";
                                }
                                command += args[PARAMETER1_INDEX];
                        } else {
                                return "";
                        }
                } else if (args.length > COMMAND_AND_PARAM_LENGTH) {
                        return "";
                } else {
                        command = addTrailingZeros(command);
                }
        }
        return command;
}

public static String createArcMessage(String[] args) {
        String command = "MA";
        if (args.length < ONLY_COMMAND_LENGTH) {
                return "";
```

```
        } else {
                if (args[COMMAND_PARAMETER].equalsIgnoreCase("forward")
                || args[COMMAND_PARAMETER].equalsIgnoreCase("forwards")) {
                        command += "F";
                } else if (args[COMMAND_PARAMETER].equalsIgnoreCase("backward")
                || args[COMMAND_PARAMETER].equalsIgnoreCase("backwards")) {
                        command += "B";
                } else {
                        return "";
                }
                if (args.length >= COMMAND_AND_PARAM_LENGTH) {
                        if (args[PARAMETER1_INDEX].equalsIgnoreCase("left")) {
                                command += "L";
                        } else if
                (args[PARAMETER1_INDEX].equalsIgnoreCase("right")) {
                                command += "R";
                        } else {
                                return "";
                        }
                }
                command += "090";
                command = addTrailingZeros(command);
        }
        return command;
}

public static String createTurnMessage(String[] args) {
        String command = "TN";
        if (args.length < ONLY_COMMAND_LENGTH) {
                return "";
        } else {
                if (args[COMMAND_PARAMETER].equalsIgnoreCase("right")) {
                        command += "R";
                } else if (args[COMMAND_PARAMETER].equalsIgnoreCase("left")) {
                        command += "L";
                } else {
                        return "";
                }
                if (args.length > ONLY_COMMAND_LENGTH) {
                        if (isNumeric(args[PARAMETER1_INDEX])) {
                                for (int i = 0; i < 7 -
                                args[PARAMETER1_INDEX].length(); i++) {
                                        command += "0";
                                }
                                command += args[PARAMETER1_INDEX];
                        } else {
                                return "";
                        }
                } else {
                        command = addTrailingZeros(command);
                }
        }
        return command;
}
```

```
public static String createStopMessage() {
	return addTrailingZeros("ST");
}

public static String createSwingMessage() {
	return addTrailingZeros("SW");
}

public static String createSetSpeedMessage(String[] args) {
	String command = "SS";
	if (args.length != 3) {
		return "";
	} else {
		if (args[COMMAND_PARAMETER].equalsIgnoreCase("a")
				|| args[COMMAND_PARAMETER].equalsIgnoreCase("b")
				|| args[COMMAND_PARAMETER].equalsIgnoreCase("c")
				|| args[COMMAND_PARAMETER].equalsIgnoreCase("d")) {
			command += args[COMMAND_PARAMETER].toUpperCase();
		} else {
			return "";
		}
		if (args[PARAMETER1_INDEX].equalsIgnoreCase("t")
				|| args[PARAMETER1_INDEX].equalsIgnoreCase("r")) {
			command += args[PARAMETER1_INDEX].toUpperCase();
		} else {
			return "";
		}
		if (isNumeric(args[PARAMETER2_INDEX])) {
			for (int i = 0; i < 6 - args[PARAMETER2_INDEX].length();
			i++) {
				command += "0";
			}
			command += args[PARAMETER2_INDEX];
		} else {
			return "";
		}
	}
	return command;
}

public static String createReadSensorMessage(String[] args) {
	String command = "";
	if (args.length != ONLY_COMMAND_LENGTH) {
		return "";
	}
	if (args[COMMAND_PARAMETER].equalsIgnoreCase("all")) {
		command = addTrailingZeros("RA");
	} else {
		if (args[COMMAND_PARAMETER].equalsIgnoreCase("u")
				|| args[COMMAND_PARAMETER].equalsIgnoreCase("t")
				|| args[COMMAND_PARAMETER].equalsIgnoreCase("m")
				|| args[COMMAND_PARAMETER].equalsIgnoreCase("l")) {
			command = addTrailingZeros("RS"
					+ args[COMMAND_PARAMETER].toUpperCase());
		}
```

```
		}
		return command;
	}

	public static String createNoOpMessage() {
		return "0000000000";
	}

	private static String createBreakpointMessage(String[] parameters,
			boolean value) {
		String message = "DMSB";
		if (parameters.length < 1) {
			return "";
		} else {
			switch (parameters[0]) {
			case "move":
				message += "MV";
				if (parameters.length == 2) {
					switch (parameters[1]) {
					case "forward":
						message += "F";
						break;
					case "backward":
						message += "B";
						break;
					}
				}
				break;
			case "arc":
				message += "MA";
				if (parameters.length >= 2) {
					switch (parameters[1]) {
					case "forward":
						message += "F";
						break;
					case "backward":
						message += "B";
						break;
					}
					if (parameters.length >= 3) {
						switch (parameters[2]) {
						case "left":
							message += "L";
							break;
						case "right":
							message += "R";
							break;
						}
					}
				}
				break;

			case "setspeed":
				message += "SS";
				break;
```

```
			case "read":
				message += "RS";
				if (parameters.length >= 2) {
					switch (parameters[1]) {
					case "touch":
						message += "T";
						break;
					case "ultrasonic":
						message += "U";
						break;
					case "microphone":
						message += "M";
						break;
					case "light":
						message += "L";
						break;
					}
				}
				break;
			case "turn":
				message += "TN";
				if (parameters.length >= 2) {
					switch (parameters[1]) {
					case "right":
						message += "R";
						break;
					case "left":
						message += "L";
						break;
					}
				}
				break;
			case "swing":
				message += "SW";
				break;
			}
		}
		if (value) {
			message = addPaddingZeros(message, "1");
		} else {
			message = addPaddingZeros(message, "0");
		}
		message += getCheckSum(message);
		return message;
	}

	private static String createPrintVariableMessage(String[] args) {
		String message = "DMPV";
		switch (args[0]) {
		case "drive":
			message += "DR";
			switch (args[1]) {
			case "rotate":
				message += "RO";
				break;
```

```
                case "travel":
                        message += "TR";
                        break;
                default:
                        return "";
                }
                break;
        case "swing":
                message += "SW";
                message += "RO";
                break;
        default:
                return "";
        }
        message = addTrailingZeros(message);
        return message;
    }

    private String decodeVariable(String message) {
        String variableString = "";
        String variable = message.substring(2, 3);
        String variableType = message.substring(3, 4);
        switch (variable) {
        case "D":
                variableString += "Drive motors ";
                break;
        case "S":
                variableString += "Swing motor ";
                break;
        default:
                return "";
        }
        switch (variableType) {
        case "R":
                variableString += "rotate speed is ";
                break;
        case "T":
                variableString += "travel speed is ";
                break;
        default:
                return "";
        }
        variableString += Integer.parseInt(message.substring(4, 10));
        return variableString;
    }

    public static void main(String[] args) throws Exception {
        Debugger d = new Debugger();
    }
}
```

Inspection Document

Laboratory # 8: Inspection

Work Product

Documentation of inspection of Group 20's source code, following three-phase inspection.

Document Revision Information

4/12/13 – Document created
4/15/13 – Phase 1 Inspection documented
4/19/13 – Phase 1 Rework documented
4/21/13 – Phase 2 Inspection and Rework documented
4/26/13 – Phase 3 Inspection and Rework documented

Approval Sheet

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

Morgan, Laura
Miaw, Jireh
Hauser, Steven
Dworak, Catherine
Bertoglio, David

Inspection Schedule

Phase 1 – Internal documentation & source-code layout
Monday, April 15 – 3:30 p.m. during in-lab.

Phase 2 – Coding practices
Sunday, April 21 – 1:30 p.m. Rice Hall.

Phase 3 – Functional correctness
Friday, April 26 – 1:00 p.m. Rice Hall.

Group Member Responsibilities

Inspections
Phase 1 Inspector – Catherine
Phase 2 Inspector – Laura
Phase 3 Inspector – David, Jireh, Steven, Catherine

Rework
Rework of source code after Phase 1 – Steven
Rework of source code after Phase 2 – Jireh
Rework of source code after Phase 3 - David

Checklists Used

Phase 1

Internal documentation & source-code layout (single inspector).

Structure

- proper use of indentation for “levels” in code
- proper use of tabbing when declaring variables
- existence of columns of related items
- existence of white space (spaces after commas, variables, between methods etc.)
- use of new line when line is too long
- consistency followed with use of braces {} throughout

Documentation

- sparing use of comments; only used to document unavoidable complexity
- identifiers
    - meaningful
    - underscores used as separators
    - capitalization of types, Classes
    - names indicate purpose
- constants
    - mixed case capitalization
    - no magic numbers (no embedded literals or constants)

    - only symbolic constants used
    - symbolic constants in all capital letters, separated by underscores
    - avoid abbreviations in names
- methods
    - mixed case for name
    - abbreviations avoided
    - names indicate function
    - “get/set” used where attribute is accessed directly
    - “is” used for Boolean methods
    - “find” used for methods that look something up
- variables
    - name should reveal purpose and/or type
    - plural if representing group of objects
    - iterator variables consistent (for example: i and j)
    - abbreviations avoided

Phase 2

- Switch statements
    - Switch statements used rather than if-else-if blocks
    - Every switch statement has a break in each case statement
- Variables
    - All variables initialized prior to use
    - All variables declared at top of function
    - All loop variables initialized just before loop
    - No variables initialized that are not used
- Classes
    - All classes have complete set of get() and set() methods
- Bounds
    - All arithmetic checked for ranges within bounds
    - All loop bounds are correct
- Methods
    - All method arguments are used in method
    - All functions named after what they return
    - All procedures named after what they do
- Conditionals
    - Complexity of conditionals should be avoided
    - All relational operators correct (> vs >=)
    - Executable statements not included in conditionals
- Miscellaneous

- No public data
- Every file is included in given file uses
- All file open commands checked for failure
- Type conversion done explicitly

Phase 3

- Methods
  - All methods return what they supposed to return
  - All methods execute what they are supposed to execute
- Variables
  - All variables exist for the purpose for which they are named
- All specified functionality is implemented

Inspection and Rework

4/15/13 Phase 1 Inspection completed
4/19/13 Rework of Phase 1 completed

4/21/13 Phase 2 Inspection completed
4/21/13 Rework of Phase 2 completed

4/26/13 Phase 3 Inspection completed
4/26/13 Rework of Phase 3 complete

Questions about Implementation by Group 19

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 about Implementation for Group 20

1. *Why do we utilize a readFlag instead of a while(True) statement in line 48 of Basestation.java*
   The readFlag allows an engineer to control whether or not the GUI will read input from the robot more easily than having to find the while(true) statement.

2. *Why do we choose to hardcode values in methods as opposed to creating a communications class and finding the correct message from there as the communication's document evolves?*
   Hardcoding allows for less complexity within the design of the code while still allowing messages to be changed relatively easily if the communications document evolves.

3. *How do we verify that we receive the sensor data after requesting it?*
   The returned message is checked through the checksum to verify that it is a valid message. The first three characters of the message are then used to determine what sensor the data is for.

4. *What is the inherent speed limit to our setSpeed method in line 271?*
   The inherent speed limit for set Speed is 999.

5. *What happens when the speed limit is set above this number?*
   A message that is longer than 11 bytes will be sent.

6. *What does exit robot do?*
   An End Connection message will be sent to the robot.

Results of Inspection

Phase 1

Date and time: Monday April 15, 2013 4:00 p.m.
Inspector: Catherine

Defects found

- proper use of indentation for "levels" in code
  line 201, else should be on next line
- existence of white space (spaces after commas, variables, between methods etc.)
  line 58, extra space between (0, 3)
  in GUI, spaces between "import" lines
  white space in beginning public class GUI
- use of new line when line is too long
  line 82 does not need to be on new line (" + e.toString());")
- consistency followed with use of braces {} throughout
  should check consistency. Starting line 199 you being to put { on the same line as the method declaration and the if statement, rather than the next line. These braces should be moved to the next line. Check methods: getTouchValue(), verifyChecksum(), getChecksum()
- sparing use of comments; only used to document unavoidable complexity
  comment on line 34 runs off screen
  comment on line 53 doesn't clarify code
  unneeded code should be removed lines 95-100

comments in moveForward(), moveBackward(), turnLeft(), turnRight(), turn180(), stop() most likely unncecessary
in GUI, comment line 20
in GUI, line 163, 362, 460, 495, 596, 632, 637, 650

- constants
  in GUI class, all private variables should be before public
  - no magic numbers (no embedded literals or constants)
    in setSpeed() what are numbers 10 and 100?
- methods
  methods between line 163 and 181 – unimplemented or unnecessary?
  - abbreviations avoided
    getUltraValue() – consider changing to getUltrasonicValue()
    getMicroValue() – consider not abbreviating
- variables
  - name should reveal purpose and/or type
    in method setSpeed, int s does not reveal purpose
  - abbreviations avoided
    variable "ret" – abbreviated for return? name does not indicate purpose,
    in methods: establishConnection(), getCheckSum()

Result of rework:

Rework performed by: Steven
Effort used in corrections: Small amount of effort as not many fields of inspection checklist were violated, rework time, 30 minutes
Approximate number of statements that had to be added: 3
Approximate number of statements that were changed: 10

Phase 2

Date and time: Sunday April 22, 2013 2:00-4:00 p.m.
Inspector: Laura

Defects found:

- Switch statements
  - Switch statements used rather than if-else-if blocks
    - *BaseStation lines 72-92: If-else-if block should be switch-case block*
    - *GUI lines 468-700: if-else-if blocks should be switch-case blocks*
- Variables
  - All variables initialized prior to use
    - *GUI lines 246 & 247: txtConnectionButtonOn and txtConnectionButtonOff background set before textFields initialized (lines 257 and 265)*

        - *GUI lines 525, 543, 560, 577: wIsPressed, aIsPressed, sIsPressed, dIsPressed used in conditionals before initialized – consider initializing them all to false in initialize()*
        - *GUI line 652: Boolean isSent never initialized*
        - *GUI line 470: int speed used before initialized on line 475*
    - All variables declared at top of function
        - *BaseStation line 71: String message declared in middle of function*
        - *BaseStation line 216: byte[] checksum should be declared at top of function*
        - *GUI line 643: Boolean valueHolder declared twice (also declared on line 25)*
    - All loop variables initialized just before loop
        - *BaseStation line 30: Initialize readFlag on line 62 (just before loop) rather than at declaration*
    - No variables initialized that are not used
        - *GUI line 28: private static GUI window never used*

Result of rework:

Rework performed by: Jireh
Effort used in corrections: rework time, 20 minutes
Approximate number of statements that had to be added: 5
Approximate number of statements that were changed: 10

Phase 3

Date and time: Performed at different times between Wednesday April 24 and Friday April 26
Inspector: David, Catherine, Steven, Jireh
Defects found:

- Methods
    - All methods return what they supposed to return]
        - *Good though all methods return void (might want to add a Boolean value to ensure correct return)*
    - All methods execute what they are supposed to execute
        - *The base station should send back an "ack" message when it receives sensor data from the robot. Otherwise the methods execute properly.*
        - *See methods that are missing implementation*
        - *Unable to connect to a NXT that is not the given one.*
        - *Terminate Connection button does not work for me, for some reason the connection was shutdown prior to clicking that event.*
            - *Button does not correctly send message to the robot.*

        - *Stop is not functioning correctly, when I tell it to move forward then release the "W" key, the robot stops then continues forward until the Ultrasonic sensor causes the robot to stop.*
        - *If I tell the robot to move in a certain direction, then release the key it doesn't always trigger the stop event.*
        - *Touch sensor is not always refreshing to display correct value (may also need to look into other sensors but those are harder to examine without actually knowing what the values should be).*
        - *os and is are abbreviations, as well as iHandle and oHandle*
        - *I understand the use of the "10" and "100" in some functions, but I would suggest at least adding a comment to explain them if you are going to leave them in instead of symbolic constants, otherwise they just seem like magic numbers*
- Variables
    - All variables exist for the purpose for which they are named
        - *Looked like it to me, however I didn't extensively look at this, was more focused on actual functionality.*
- All specified functionality is implemented
    - *Is only able to connect to a specific NXT, should be more generic and able to connect to any NXT nearby with the specific PIN.*
    - *If connection fails, the GUI has no way of instantiating a new connection without restarting program.*
    - Missing Implementation
        - *Sending back an "ack" message upon receiving sensor data*
        - *moveForwardLeft(), moveForwardRight(), moveBackwardLeft(), moveForwardRight().*
        - *Add ability to move swinging arm (not specified but should be added as a "cool" feature)*
        - *Need to implement the timeout for if an ACK is not receive*
    - Getting a bunch of exceptions that shouldn't be occurring, usually of the IOException: Stream Closed type.

Result of rework:

Rework performed by: David
Effort used in corrections: rework time, 2 hours
Approximate number of statements that had to be added: 100
Approximate number of statements that were changed: 3

Debugging Interface

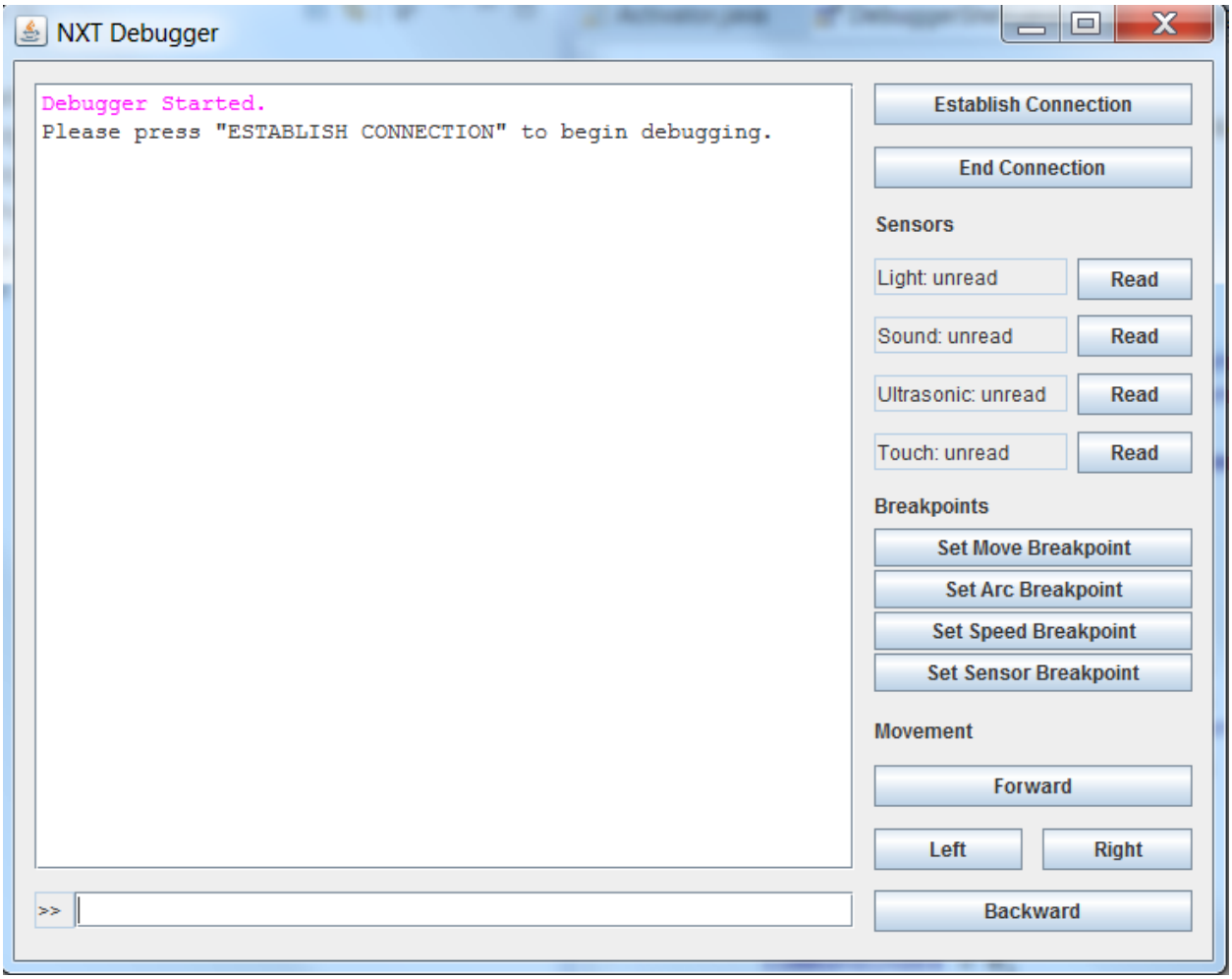
Work Product

Image of Debugger GUI

Approval Sheet

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

Morgan, Laura
Miaw, Jireh
Hauser, Steven
Dworak, Catherine
Bertoglio, David


NXT Debugger
Debugger Started.
Please press "ESTABLISH CONNECTION" to begin debugging.
>>
Establish Connection
End Connection
Sensors
Light: unread
Read
Sound: unread
Read
Ultrasonic: unread
Read
Touch: unread
Read
Breakpoints
Set Move Breakpoint
Set Arc Breakpoint
Set Speed Breakpoint
Set Sensor Breakpoint
Movement
Forward
Left
Right
Backward

Management Reports

Work Product

Weekly management report containing the responsibilities of each member over the course of the week and his/her contributions to the weekly tasks. Also includes tasks completed during the meeting, a schedule for the upcoming week, and any significant unresolved problems encountered.

Document Revision Information

2/1/13 – Laboratory 1 Management Report created
4/14/13 – Laboratory 7 Management Report created

Approval Sheet

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

Morgan, Laura
Miaw, Jireh
Hauser, Steven
Dworak, Catherine
Bertoglio, David

Laboratory #1: Risk Reduction Prototypes - February 8, 2013

Management Responsibilities

Laura Morgan:	Document preparation
Jireh Miaw:	Scheduling and task assignment
David Bertoglio:	Configuration management and file system control
Catherine Dworak:	Web site development
Steven Hauser:	Presentation preparation

Contributions

Laura Morgan:

- Created the Risk Reduction document and compiled work completed for it
- Aided in the completion of the Risk Reduction document

Jireh Miaw:

- Created documentation for the management report
- Oversaw and documented tasks completed during the management meeting in the report
- Aided in the completion of the Risk Reduction document

Steven Hauser:

- Aided in the completion of the Risk Reduction document

Catherine Dworak:

- Managed and uploaded documents onto the team website
- Aided in the completion of the Risk Reduction document

David Bertoglio:

- Began creation of the evolutionary prototype for the robot system
- Aided in the completion of the Risk Reduction document

Meeting Overview

Attendees:

Laura Morgan
Jireh Miaw
Steven Hauser
Catherine Dworak
David Bertoglio

Meeting Location:

Rice Hall: Third Floor

Meeting Time:

1:00 p.m. to 4:30 p.m.

Agenda:

Postlab 1:

- Choose a name for the team robotics company
- Prepare website for uploading documents
- Create risk reduction document for Post-Laboratory assignment 1
- Develop a template for the system specification document
- Prepare management report
- Develop evolutionary prototype for the robot system

Prelab 2:

- Determine requirements for the robot control system
- Develop ideas to present to the customer
- Develop a mock-up GUI
- Create an SRS for the mock-up GUI
- Prepare a presentation for our GUI
- Develop a draft of specifications for the communications protocol

List of Completed Tasks

- Choose a name for the team robotics company
- Prepare website for uploading documents
- Create risk reduction document for Post-Laboratory assignment 1
- Develop a template for the system specification document
- Prepare management report

Schedule for Upcoming Week

Additional Meeting:

Feb 10, 2013 at 1:00 p.m

Tasks to be Completed:

Laura Morgan

- Create specifications document — Feb 11

Jireh Miaw

- Create specifications document — Feb 11

Steven Hauser

- Develop mock-up GUI — Feb 10

Catherine Dworak

- Create specifications document — Feb 11
- Get in contact with group 20 for communications protocol — Feb 10

David Bertoglio

- Finish evolutionary prototype — Feb 10

Unresolved Problems

Laboratory #7: Enhanced Prototype - April 14, 2013

Management Responsibilities

Laura Morgan:	Document preparation
Jireh Miaw:	Scheduling and task assignment
David Bertoglio:	Configuration management and file system control
Catherine Dworak:	Web site development
Steven Hauser:	Presentation preparation

Contributions

Laura Morgan:

- Participated in creation of goals and milestones of enhanced prototype

Jireh Miaw:

- Looking into additional functionality
- Editing code for implementation of sensors

Steven Hauser:

- Edited source code

Catherine Dworak:

- Participated in creation of goals and milestones of enhanced prototype
- Began developing checklist for inspection

David Bertoglio:

- Worked on code for debugger

Meeting Overview

Attendees:

Laura Morgan
Jireh Miaw
Catherine Dworak
David Bertoglio

Meeting Location:

Rice Hall: Third Floor

Meeting Time:

April 12, 2013 Friday, 1:00 p.m. to 4:00 p.m.

Agenda:

Postlab 7:

- Complete enhanced prototype for functionality of all sensors
- Prepare for second integration test with Group 20

Prelab 8:

- Begin developing checklist for inspection
- Assign responsibilities for inspection
- Develop schedule for inspection

List of Completed Tasks

- Assigned responsibilities for inspection
- Scheduled integration test of enhanced prototype for Sunday, April 14 with Group 20
- Created schedule for inspection
- Prepared prototype for test (not yet completed, to be finished Sunday)

Schedule for Upcoming Week

Additional Meeting:
April 14, 2013 at 3:00 p.m. in Rice 3rd floor

Agenda:
- Meet for third enhanced prototype integration test
- Document the results of enhanced prototype integration testing

Tasks to be Completed:
Laura Morgan
- Finish documentation for enhanced prototype April 14

Jireh Miaw
- Finish editing enhanced prototype code April 14

Steven Hauser
- Edit code for debugger April 14

Catherine Dworak
- Edit website, prepare management report, checklist for prelab April 14

David Bertoglio
- Finish the enhanced prototype April 14

Unresolved Problems
- Some functionality to implement in prototype (specifically the timer)
- Debugger not completed (breakpoints, printing variables, setting variables, and displays)