|  |
| --- |
| University of Virginia |
| Roboticorp Portfolio |
| David Bertoglio, Catherine Dworak, Steven Hauser, Jireh Miaw, Laura Morgan Team 19 |

|  |
| --- |
|  |

April 19, 2013

Team 19 Portfolio

**Laboratory # 10: Delivery**

**Morgan, Laura**

**Miaw, Jireh**

**Hauser, Steven**

**Dworak, Catherine**

**Bertoglio, David**

***Work Product***

**Compilation of all artifacts and documents of our project.**

***Document Revision Information***

**4/19/2013 Created.**

**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

[Table of Contents 3](#_Toc354220554)

[Overview 4](#_Toc354220555)

[Specification – Robot Control 5](#_Toc354220556)

[Specification - Onboard System and Debugger 14](#_Toc354220557)

[Design Document 15](#_Toc354220558)

[Software Source Code 25](#_Toc354220559)

[Inspection Report 26](#_Toc354220560)

[Debugging Interface 27](#_Toc354220561)

[Management Reports 28](#_Toc354220562)

# Team 19



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

# Specification – Robot Control

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 |

## 

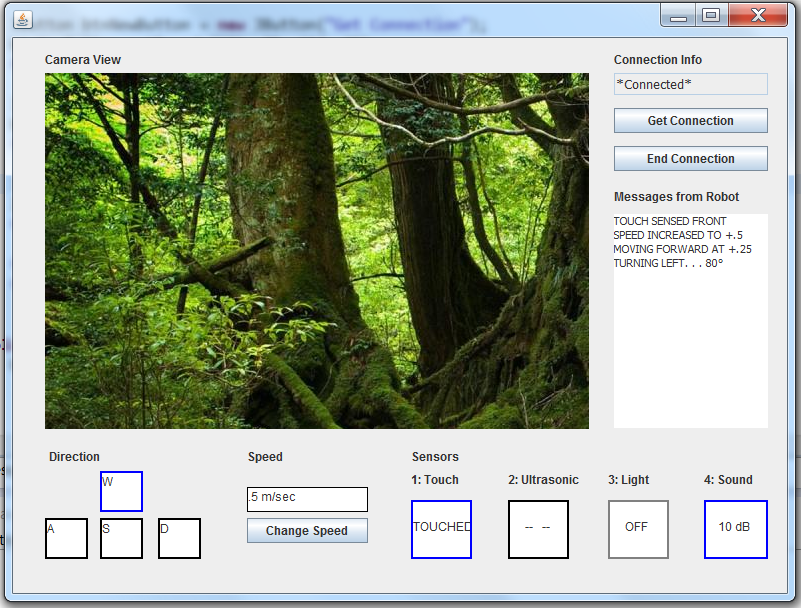
## Set of Modes

|  |  |
| --- | --- |
| 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\* |

# Specification - Onboard System and Debugger

## 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.

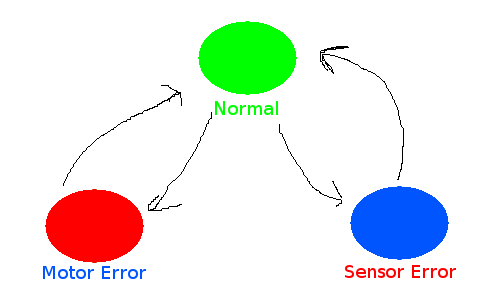
## Modes

|  |  |
| --- | --- |
| **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\*  ACTION | @T(!motor\_2\_speed! != 0 && //motor\_2\_speed// = 0)  !motor\_2\_speed! := 0 //ERROR// shows $motor\_2\_error$ |
| \*motor\_error\*  ACTION | @T(!motor\_3\_speed! != 0 && //motor\_3\_speed// = 0)  !motor\_3\_speed! := 0 //ERROR// shows $motor\_3\_error$ |
| \*sensor\_error\*  ACTION | @F(%sensor\_mic\_running%)  !sensor\_mic! := 0  //ERROR// shows $sensor\_mic\_error$ |
| \*sensor\_error\*  ACTION | @F(%sensor\_ultrasonic\_running%)  !sensor\_ultra! := 0 //ERROR// shows $sensor\_ultrasonic\_error$ |
| \*sensor\_error\*  ACTION | @F(%sensor\_touch\_running%)  !sensor\_touch! := 0 //ERROR// shows $sensor\_touch\_error$ |
| \*sensor\_error\*  ACTION | @F(%sensor\_light\_running%)  !sensor\_light! := 0 //ERROR// shows $sensor\_light\_error$ |
| \*sensor\_error\*  ACTION | @F(%bluetooth\_sensor\_running%)  !motor\_1\_speed! := 0 !motor\_2\_speed! := 0 !motor\_3\_speed! := 0 //ERROR// shows $connection\_error$ |
| \*sensor\_error\*  ACTION | @F( !time\_robot\_sent! - !time\_base\_received! <= $timeout$)  !motor\_1\_speed! := 0 !motor\_2\_speed! := 0 !motor\_3\_speed! := 0 //ERROR// shows $connection\_error$ |
| \*sensor\_error\*  ACTION | @F( !time\_base\_sent! - !time\_robot\_received! <= $timeout$)  !motor\_1\_speed! := 0 !motor\_2\_speed! := 0 !motor\_3\_speed! := 0 //ERROR// shows $connection\_error$ |
| \*sensor\_error\*  ACTION | @T(!poll\_time! > $poll\_interval$  !motor\_1\_speed! := 0 !motor\_2\_speed! := 0 !motor\_3\_speed! := 0 //ERROR// shows $connection\_error$ |

## Glossary

### Symbolic Constants

|  |  |  |
| --- | --- | --- |
| **Name** | **Definition** | **Value** |
| $no\_error$ | String | “Running well” |
| $motor\_1\_error$ | String | “ERM0000001” |
| $motor\_2\_error$ | String | “ERM0000002” |
| $motor\_3\_error$ | String | “ERM0000003” |
| $sensor\_mic\_error$ | String | “ERS0000001” |
| $sensor\_ultrasonic\_error$ | String | “ERS0000002” |
| $sensor\_touch\_error$ | String | “ERS0000003” |
| $sensor\_light\_error$ | String | “ERS0000004” |
| $connection\_error$ | String | “Connection to the robot has been interrupted.” |
| $poll\_interval$ | Time (in ms) between base-station polls to determine Bluetooth connectivity. | 1000ms |
| $timeout$ | Time in ms that the !message\_received! signal has to be received between !message\_sent! before \*connection\_lost\* is declared. | 5000ms |
| $max\_forward\_speed$ | Maximum speed at which motor can operate without reaching an !unsafe\_speed! | TBD |
| $max\_backward\_speed$ | Maximum speed at which motor can operate without reaching !unsafe\_speed! | TBD |

### Text Macros

|  |  |
| --- | --- |
| **Name** | **Definition** |
| !motor\_1\_speed! | Integer to contain motor 1 speed |
| !motor\_2\_speed! | Integer to contain motor 2 speed |
| !motor\_3\_speed! | Integer to contain motor 3 speed |
| !sensor\_light! | Value to hold output from light sensor |
| !sensor\_mic! | Value to hold output from microphone |
| !sensor\_ultra! | Value to hold output from ultrasonic sensor |
| !sensor\_touch! | Value to hold output from touch sensor |
| !poll\_time! | Amount of time required to poll from base station from robot. |

### 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! >= $max\_backward\_speed$ |

# Design Document

## 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 others’ 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.

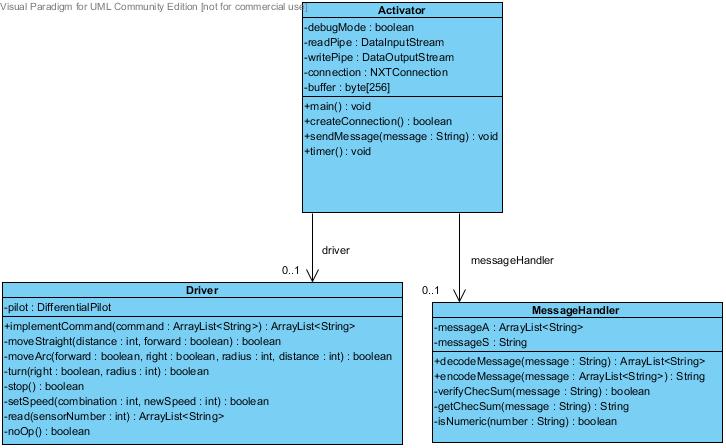
### MessageHandler

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

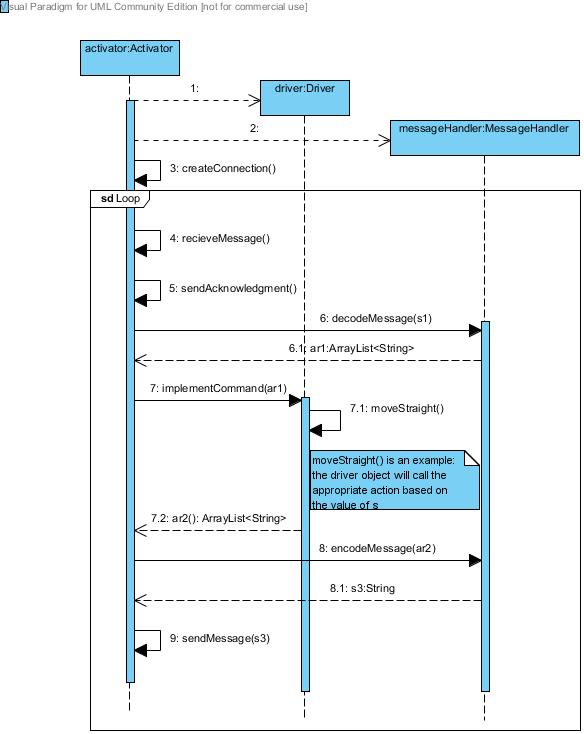
The Driver class is in charge of performing robot actions. It will contain an instance of the Differential Pilot Object from Lejos, which contains classes that control robot movement, such as setting the speed and rotating. The Driver class will contain a method for each action the robot should be able 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

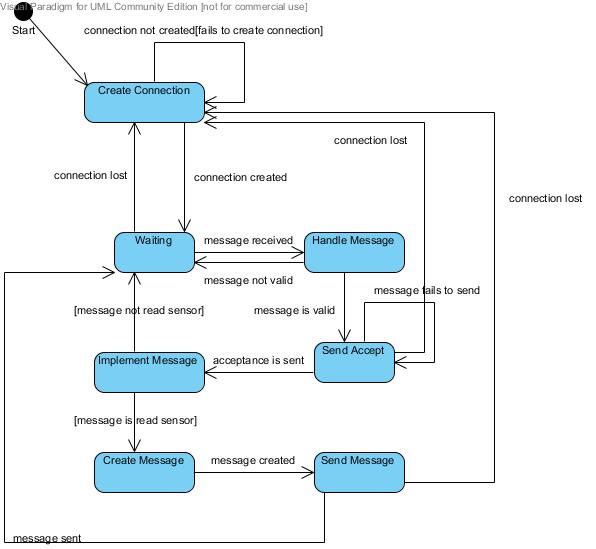


## 

## UML Sequence Diagram

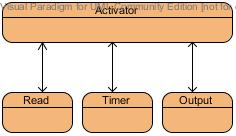


## Finite State Diagram



## 

## 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

# Inspection Report

# Debugging Interface

# Management Reports

## 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