REV3 (Remote EV3 Control) Application Design Document

Version 1.2
April 8, 2016

ECE 573, Spring 2016
University of Arizona

Bryan Walsh
Team Name: bpwalsh

Table of Contents

Table of Tables

Table of Figures

Version	**Date**	**Description**
1.0	February 22, 2016	• Initial Release
1.1	March 3, 2016	• Updated course coordinates on cover page.
1.2	April 8, 2016	• Refactored GUIPowerProcessor and SensorPowerProcessor into PowerProcessor. • Refactored GUITurnProcessor and SensorTurnProcessor into TurnProcessor. • Reworked Transmit Commands use case. • Ranamed some functions to make more consistent

Table 1: Design Document Version History

1. Executive Summary

The REV3 (Remote EV3 Control) application will allow the user to remotely control a Lego EV3 Robot, running the open source, debian based EV3Dev (www.ev3dev.org) operating system, from an Android smart phone. This application will include the ability to control the EV3 via GUI or by use of the phone's accelerometer and gyro sensors.

2. Project Overview

This project will focus on the REV3 Android application itself. It does not include the software to be hosted on the EV3 required to accept and interpret commands being sent from the phone. The messaging protocol specification between REV3 and the EV3 will be developed as part of this project. The EV3 Protocol TCP/IP stack will not be implemented for this project. For verification purposes a file-based testdriver (EV3 Proxy) will be used to stimulate necessary inputs and outputs across the EV3 protocol. All transmitted messages will be recorded for verification purposes. Received messages will be stimulated by prerecorded files. EV3 hosted software will be developed at a later point in time as part of a follow-on effort.

The REV3 application will be targeted for the Android Marshmallow 6.0 operating system and will be written using java 1.7.

3. Functional Requirements

Functional requirements are defined below. “B-Grade” requirements are prefixed with a “B”. “A-Grade” requirements are prefixed with an “A”.

3.1 Startup and Configuration

B.1.1 Upon startup the REV3 application shall immediately enter a configuration screen. The configuration screen shall allow the user to enter the IP address or hostname and port of the EV3 robot that they wish to connect to. The screen shall also have a connect button that upon clicking will establish connectivity to the EV3 robot. (No Dependencies)

A.1.2 If the REV3 application looses connectivity to the EV3 the application shall reinitialize and return to the configuration screen. (Dependencies: B.1.1)

3.2 Drive Controls

The REV3 application shall have a “drive” screen from which the user can control of the EV3 robot.

B.2.1 From the “drive” screen the user shall be able to adjust the commanded forward/reverse (+100%/-100%) speed via touch screen controls. The touchscreen commands will be translated to the proper EV3 protocol message necessary to accelerate/decelerate. (Dependencies: B.1.1, B.3.1)

A.2.2 From the “drive” screen the user shall be able to adjust the commanded turn rate (+100% turn left/-100% turn right) speed via touch screen controls. The touchscreen commands will be translated to the proper EV3 protocol message necessary to turn. (Dependencies: B.1.1, B.3.2)

A.2.3 The REV3 application shall be capable of receiving the change in heading (since configuration time) over the EV3 protocol. The application shall display the change in heading via an indicator arrow on the “drive” screen. (Dependencies: B.1.1, A.3.3)

A.2.4 From the “drive” screen the user shall be able to adjust the commanded speed (+100%/-100%) speed by accelerating the phone forward or backward. The user interaction shall be captured by the phones accelerometer sensor. The accelerometer outputs will be translated to the proper EV3 protocol “set speed” message. (Dependencies: B.1.1, B.3.1)

B.2.5 From the “drive” screen the user shall be able to adjust the commanded turn rate (+100% turn left/-100% turn right) speed by turning the phone left or right. The user interaction shall be captured by the phones gyro sensor. The gyro outputs will be translated to the proper EV3 protocol “turn” message. (Dependencies: B.1.1, B.3.2)

3.3 EV3 Protocol

B.3.1 The EV3 protocol shall contain a message to command the speed of the EV3 by specifying the percent power level. +100% shall be interpreted as full power forward. -100% shall interpreted as full power backwards. (No Dependencies)

B.3.2 The EV3 protocol shall contain a message to command the turn rate of the EV3. +100% shall correspond to a hard left turn. -100% shall correspond to a hard right turn. (No Dependencies)

A.3.3 The EV3 protocol shall contain a message to receive the delta change in heading (since configuration) from the EV3. The delta change in heading shall be expressed in degrees. (No Dependencies)

4. REV3 Design Overview

4.1 REV3 Application State

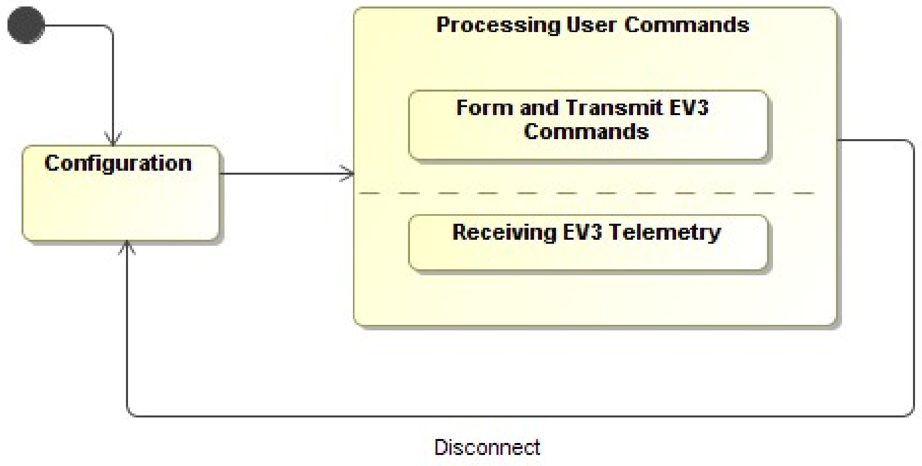
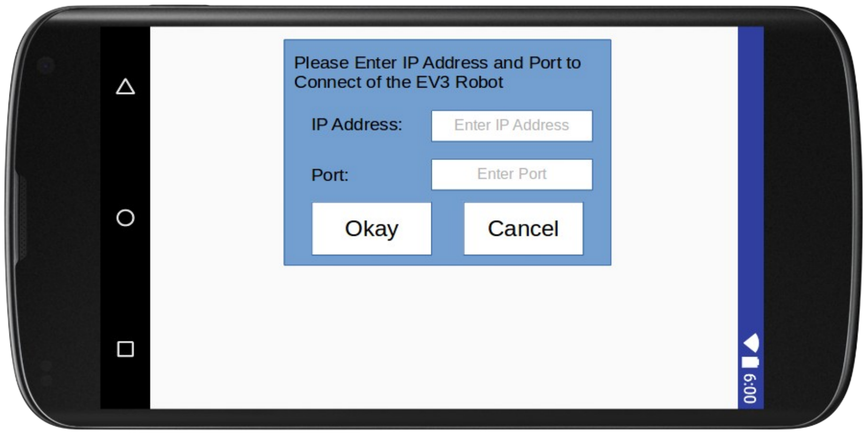


Figure 1: REV3 Application States

The REV3 contains 2 main states, a configuration state entered at application startup or if REV3 becomes disconnected from the EV3, and a Processing User Commands state. The Processing User Commands state is the main state in which the user can interact with the EV3. The user can use the UI or android sensors to issue commands to the EV3. The Processing User Commands state, contain two sub-states that run in parallel. A Form and Transmit EV3 Commands, which processes user UI interactions and sensor events, and is responsible for transmitting commands to the EV3 or its proxy. The Receiving EV3 Telemetry state is responsible for receiving telemetry data from the EV3 so that it can later be used by the UI.

4.2 REV3 UI Prototype

The REV3 will contain two screens. The first screen contains the Configuration Dialog. This screen will be enable on application startup and will allow the user to specify the IP address and the port of the EV3 robot to connect to. The second screen will be the main control screen, allowing the user to drive the EV3 robot around via the UI.


Please Enter IP Address and Port to Connect of the EV3 Robot

IP Address: Enter IP Address

Port: Enter Port

Figure 2: Configuration Screen

The main control screen contains buttons to allow the user to command the EV3 robot left or right, as well as set the speed (power level) incrementally backwards or forwards.

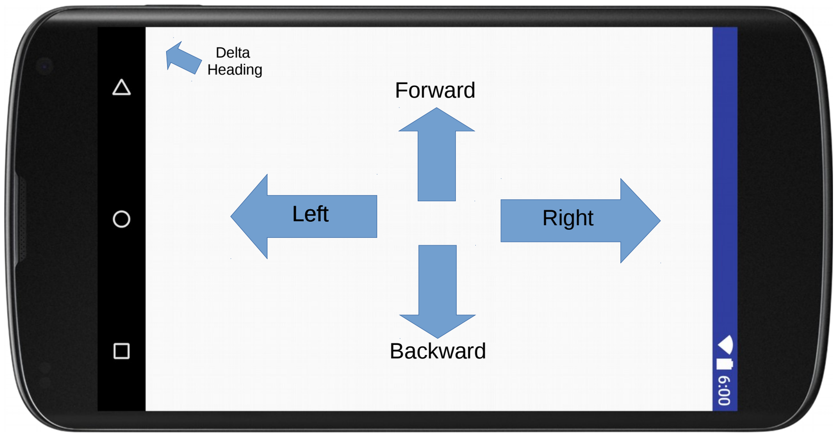


Figure 3: Main Control Screen

4.3 REV3 Class Model

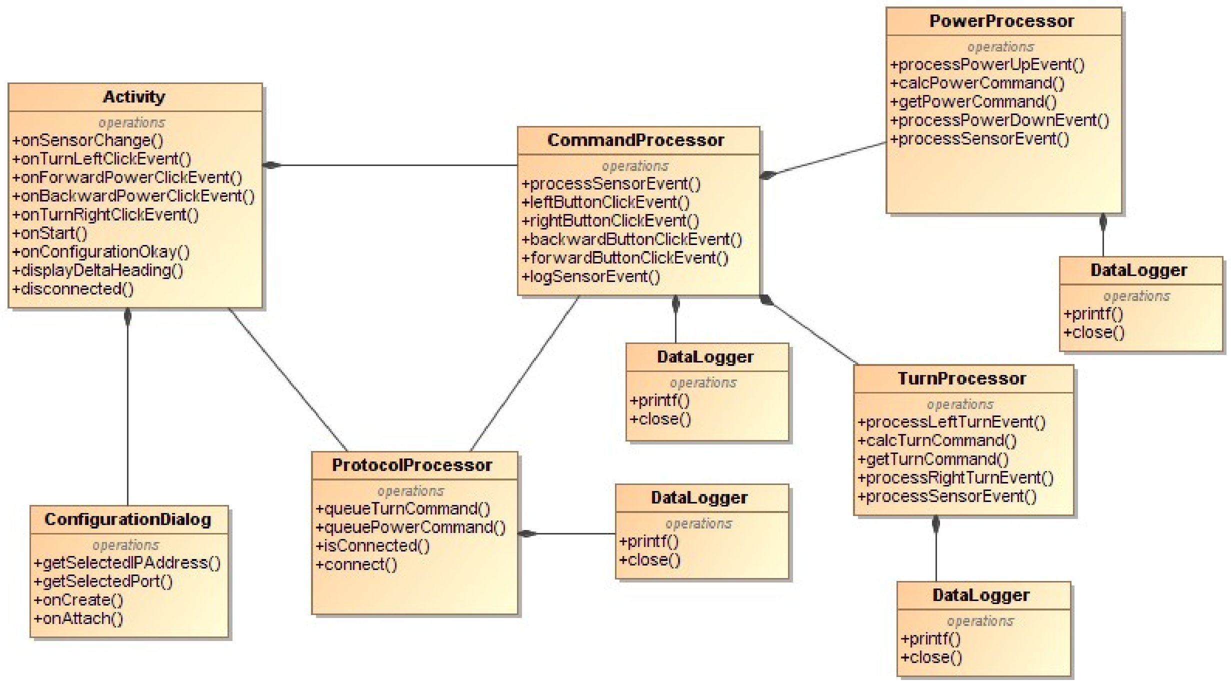


Figure 4: REV3 Class Diagram

The classes in the above class diagram are summarized below.

Activity

The Activity class is the main class of the android application all UI events go through this class. The Activity class is responsible for forwarding the UI events to the CommandProcessor.

ConfigurationDialog

The ConfigurationDialog class will be used to prompt the user for the IP address and port of the EV3 robot to connect to.

CommandProcessor

The CommandProcessor class will process button clicks from the main control screen, as well as sensor inputs. These inputs will be converted to appropriate Turn or Set Power Level commands and will be forwarded to the Protocol Processor.

ProtocolProcessor

The ProtocolProcessor is responsible for communicating and maintaining connection with the EV3 Proxy. It receives commands from the command processor and transmits them to the EV3 Proxy. It will also receive telemetry command (containing Delta Heading information) and forward the message on to be displayed by the Activity.

Datalogger

Helper class to help the various processors log their state. The logged data will then be used to support unit testing and reequirement.

PowerProcessor

Helper class for the CommandProcessor. The PowerProcessor is responsible for interpreting “Forward” and “Backward” button clicks, and the “TYPE_LINEAR_ACCELERATION” sensor inputs, in order to generate the appropriate Set Power Level commands.

TurnProcessor

Helper class for the CommandProcessor. The TurnProcessor is responsible for interpreting “Left” and “Right” button clicks, and the “TYPE_ACCELEROMETER” and “TYPE_GYRO” sensor inputs, in order to generate the appropriate Turn commands.

4.4 Protocol Specification

The REV3 to EV3 protocol will utilize an ASCII based text encoding, ie all data will be represented as valid ascii strings. The messages are defined as follows:

Initialization Message		
Field Name:	**Data Type:**	**Value Description:**
Header	string	“INIT”
Telemetry Port	integer	Port number that REV3 application is listening on for EV3 telemetry data.
REV3 IP address	string	IP Address of the host that the REV3 application is running on.
Set Power Level Message		
Field Name:	**Data Type:**	**Value Description:**
Header	string	“POWER”
Power Level	double	Desired power level, valid values between -100 and 100.
Command Turn Message		
Field Name:	**Data Type:**	**Value Description:**
Header	string	“Turn”
Turn Command	double	Desired Turn Command, valid values between -100 (full left) and 100 (full right).
Telemetry Message		
Field Name:	**Data Type:**	**Value Description:**
Header	string	“TM”
Delta Heading	double	Desired Turn Command, valid values between -100 (full left) and 100 (full right).

Table 2: Protocol Description

5. Analysis Model

Below is a diagram depicting nominal use cases for the REV3 application. The use cases are further described below along with the corresponding use case realization (sequence model).

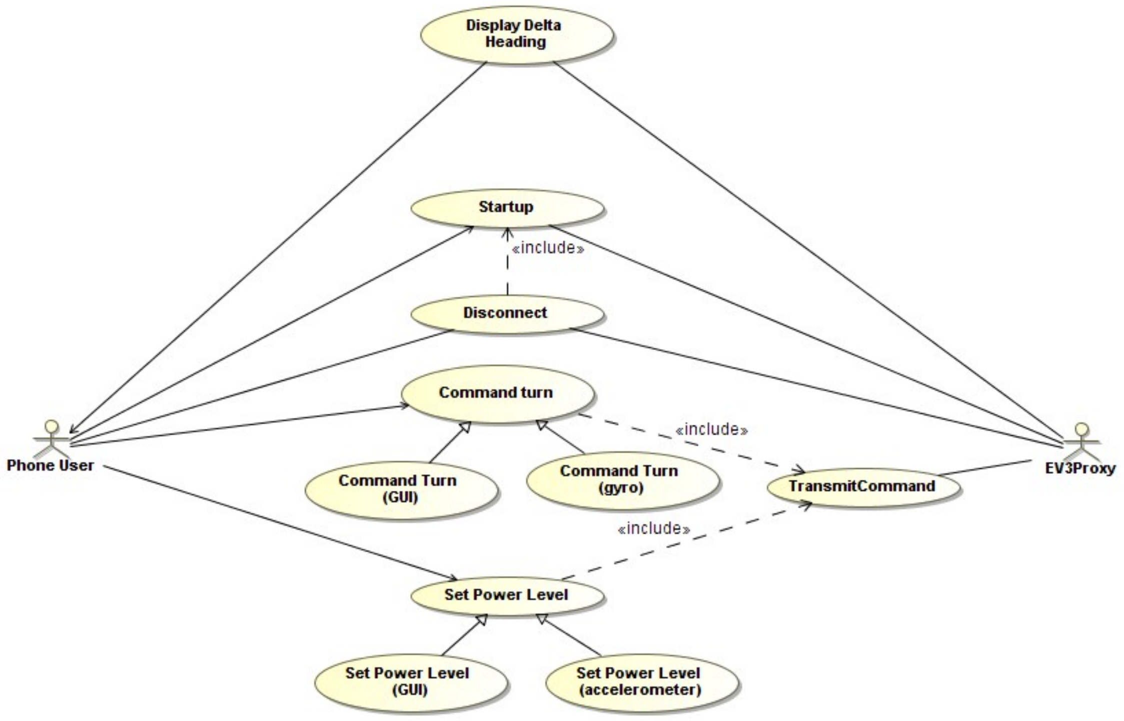


Figure 5: REV3 Use Case Diagram

5.1 Startup

5.1.1 Use Case

Summary:
The user starts the REV3 application and specifies the port and IP address of the EV3 robot to connect to. For this project the EV3 robot will be replaced by a file based EV3 proxy.

Realizes Requirement:
B.1.1

Actors:
Phone User
EV3 Proxy

Preconditions:
REV3 Application Installed on Phone, Phone is connected to an IP (WIFI) based network.

Description:
The user will start the REV3 application. Upon startup a dialog box will appear allowing the user to

specify IP address and port of the EV3 robot to connect to. The REV3 application will upon up a TCP/IP connection to the EV3 robot. Upon success REV3 will enter the main control screen.

Exceptions:
Unable to Connect: If the user is unable to connect, the configuration dialog will reset and the user will be prompted to renter the IP address and port.

Post Conditions:
The REV3 application is waiting at the main control screen, ready to process in the user's commands.

5.1.2 Sequence Model

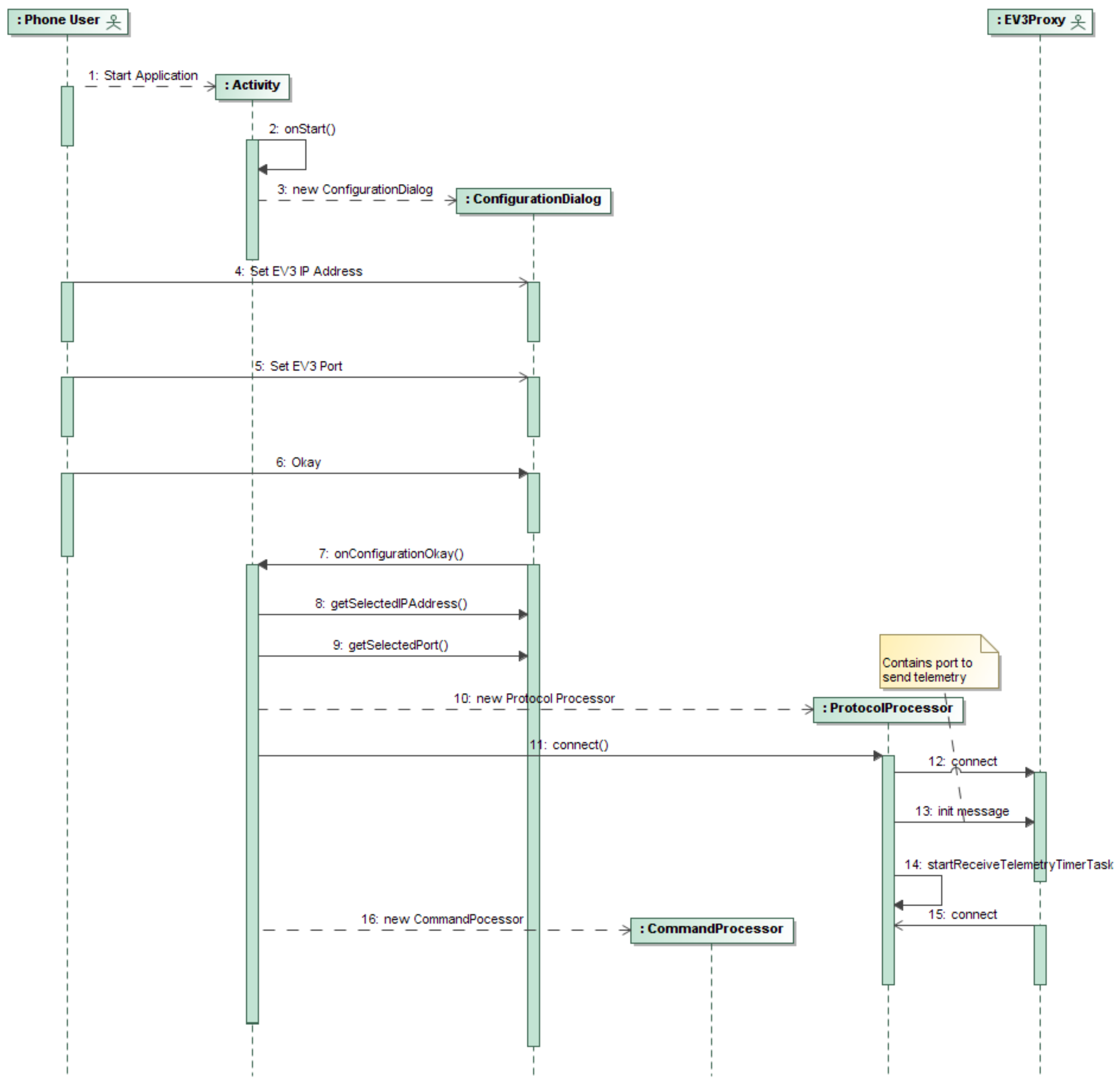


Figure 6: Startup Use Case Realization

1. User launches the REV3 application
2. The android OS invokes REV3's onStart activity call.
3. The REV3 application upon a new Configuration Dialog
4. The user enters the IP address of the EV3 robot.
5. The user enters the connection port of the EV3 robot
6. The user presses okay.

7. The Activity's onConfigurationOkay activity callback is called
8. The activity gets the selected IP from the Configuration Dialog
9. the activity gets the selected port from the Configuration Dialog
10. The activity creates a new ProtocolProcessor, passing in the IP address and port.
11. The ProtocolProcessor will create a new timer task to be used to transmit commands to the EV3 proxy.
12. The ProtocolProcessor will connect to the EV3 proxy.
13. The ProtocolProcessor will send an initialization message to the EV3 proxy, containing the port and IP of the REV3 application. The port will be used to send back EV3 telemetry data (Delta Heading Change).
14. The ProtocolProcessor will startup a timer task to receive EV3 telemetry.
15. The EV3 Proxy will connect to the ProtocolProcessors telemetry task.

5.2 Command Turn (GUI)

5.2.1 Use Case

Summary:
The user enters UI commands for the EV3 to turn left or right.

Realizes Requirement:
A.2.2

Actors:
Phone User

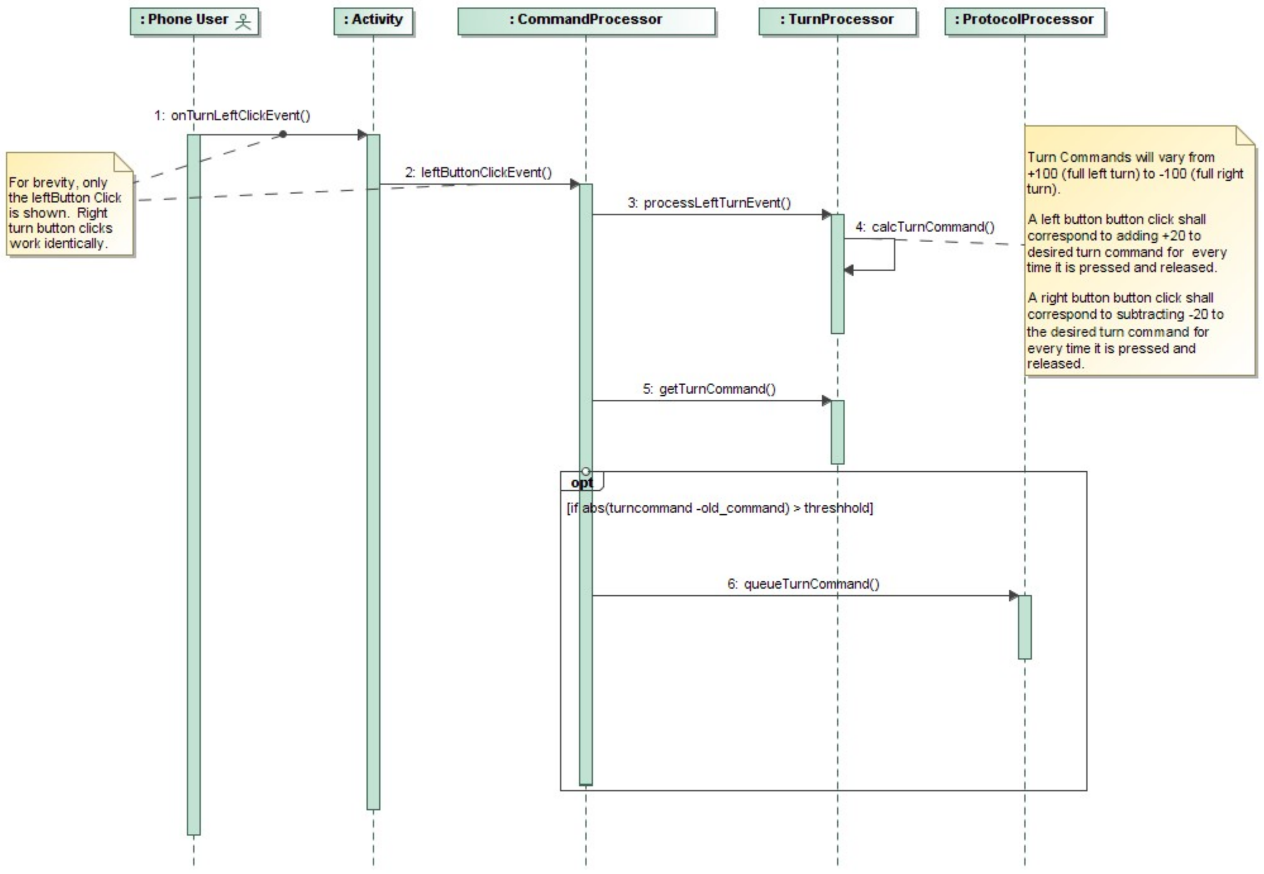
Preconditions:
Startup has completed and user is at the main control screen.

Description:
The user will press the turn left arrow to command a left turn.   The user will press the turn right arrow to command a right turn.   The REV3 will take the user's UI interaction and translate it to the proper command to be sent to the EV3 Proxy.

Exceptions:
Disconnect: The REV3 application is disconnected from the EV3 Proxy. Transition to the Disconnect use case.

Post Conditions:
Turn commands sent to the Protocol Processor are ready to be sent to to the EV3 Proxy.

5.2.2 Sequence Model



: Phone User
: Activity
: CommandProcessor
: TurnProcessor
: ProtocolProcessor
1: onTurnLeftClickEvent()
2: leftButtonClickEvent()
3: processLeftTurnEvent()
4: calcTurnCommand()
5: getTurnCommand()
opt
[if abs(turncommand -old_command) > threshhold]
6: queueTurnCommand()
For brevity, only the leftButton Click is shown. Right turn button clicks work identically.
Turn Commands will vary from +100 (full left turn) to -100 (full right turn).
A left button button click shall correspond to adding +20 to desired turn command for every time it is pressed and released.
A right button button click shall correspond to subtracting -20 to the desired turn command for every time it is pressed and released.

Figure 7: Command Turn (GUI) Use Case Realization

1. The user clicks on the left (or right) turn icon. The activities UI callback is called.
2. The activity notifies the Command Processor of the click event.
3. The Command Processor calls the Turn Processor.
4. The GUI Turn Processor will calculate the desired turn command. Turn Commands will vary from +100 (full left turn) to -100 (full right turn). A left button button click shall correspond to adding +20 to desired turn command every time it is pressed and released. A right button button click shall correspond to subtracting -20 to the desired turn command every time it is pressed and released.
5. The Command Processor will get the calculated turn command from the Turn Processor
6. The Command Processor will compare the newly generated command to the command that had been previously sent to the Protocol Processor. If the difference is greater than the threshold (5) a new command will be sent to the Protocol Processor.

5.3 Set Power Level (GUI)

5.3.1 Use Case

Summary:
The user enters UI commands for the EV3 to increase or decrease power (speed).

Realizes Requirement:
B.2.1

Actors:
Phone User

Preconditions:
Startup has completed and user is at the main control screen.

Description:
The user will press the forward arrow to command an increase in power. The user will press the backward arrow to command a decrease in power (through reverse power). The REV3 will take the user's UI interaction and translate it to the proper command to be sent to the EV3 Proxy.

Exceptions:
Disconnect: The REV3 application is disconnected from the EV3 Proxy. Transition to the Disconnect use case.

Post Conditions:
Power commands sent to the Protocol Processor are ready to be sent to to the EV3 Proxy.

5.3.2 Sequence Model

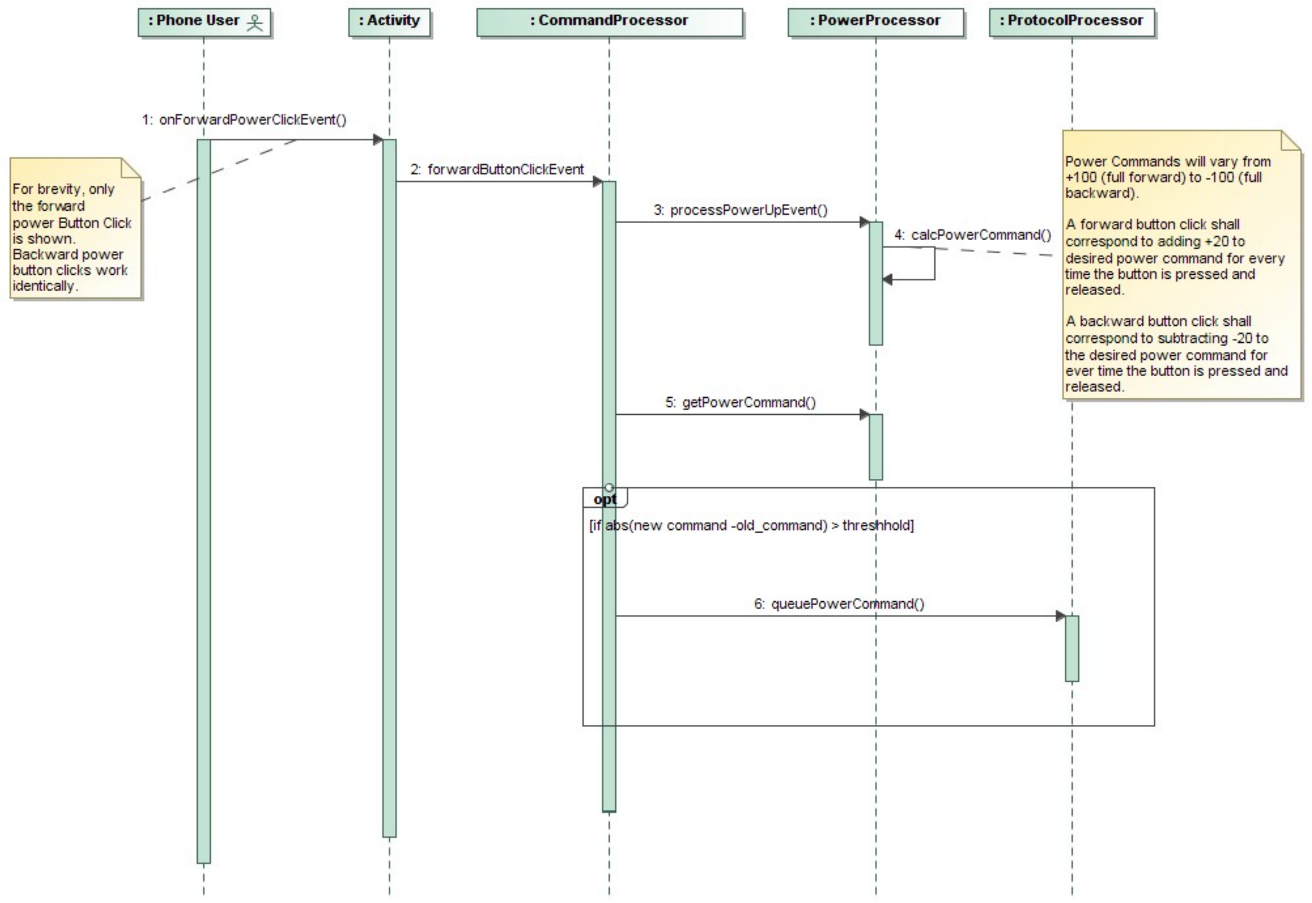

: Phone User
: Activity
: CommandProcessor
: PowerProcessor
: ProtocolProcessor
1: onForwardPowerClickEvent()
2: forwardButtonClickEvent
3: processPowerUpEvent()
4: calcPowerCommand()
5: getPowerCommand()
opt
[if abs(new command -old_command) > threshhold]
6: queuePowerCommand()
For brevity, only the forward power Button Click is shown. Backward power button clicks work identically.
Power Commands will vary from +100 (full forward) to -100 (full backward).
A forward button click shall correspond to adding +20 to desired power command for every time the button is pressed and released.
A backward button click shall correspond to subtracting -20 to the desired power command for ever time the button is pressed and released.

Figure 8: Set Power Level (GUI) Use Case Realization

1. The user clicks on the forward (or backwards) power icon. The activities UI callback is called.
2. The activity notifies the Command Processor of the click event.
3. The Command Processor calls the Power Processor.
4. The Power Processor will calculate the desired power command. Power Commands will vary from +100 (full forward) to -100 (full backward). A forward button click shall correspond to adding +20 to desired power command for every time the button is pressed and released. A backward button click shall correspond to subtracting -20 to the desired power command for ever time the button is pressed and released.
5. The Command Processor will get the calculated power command from the Power Processor
6. The Command Processor will compare the newly generated command to the command that had been previously sent to the Protocol Processor. If the difference is greater than the threshold (5) a new command will be sent to the Protocol Processor.

5.4 Command Turn (Gyro)

5.4.1 Use Case

Summary:
The user enters commands for the EV3 to by turn left or right by rotating the phone (similar to how a driver would turn a steering wheel).

Realizes Requirement:
B.2.5

Actors:
Phone User

Preconditions:
Startup has completed and user is at the main control screen. User is holding phone in portrait fashion.

Description:
The user will utilize REV3 to issue a command to the EV3 Proxy to turn left or right.   The user will rotate the phone counterclockwise to issue a turn left command, the user will rotate the phone clockwise to issue a turn right command. The REV3 will take the user's interaction and translate it to the proper command to be sent to the EV3 Proxy.

Exceptions:
Disconnect: The REV3 application is disconnected from the EV3 Proxy. Transition to the Disconnect use case.

Post Conditions:
Turn commands sent to the Protocol Processor are ready to be sent to to the EV3 Proxy.

5.4.2 Sequence Model

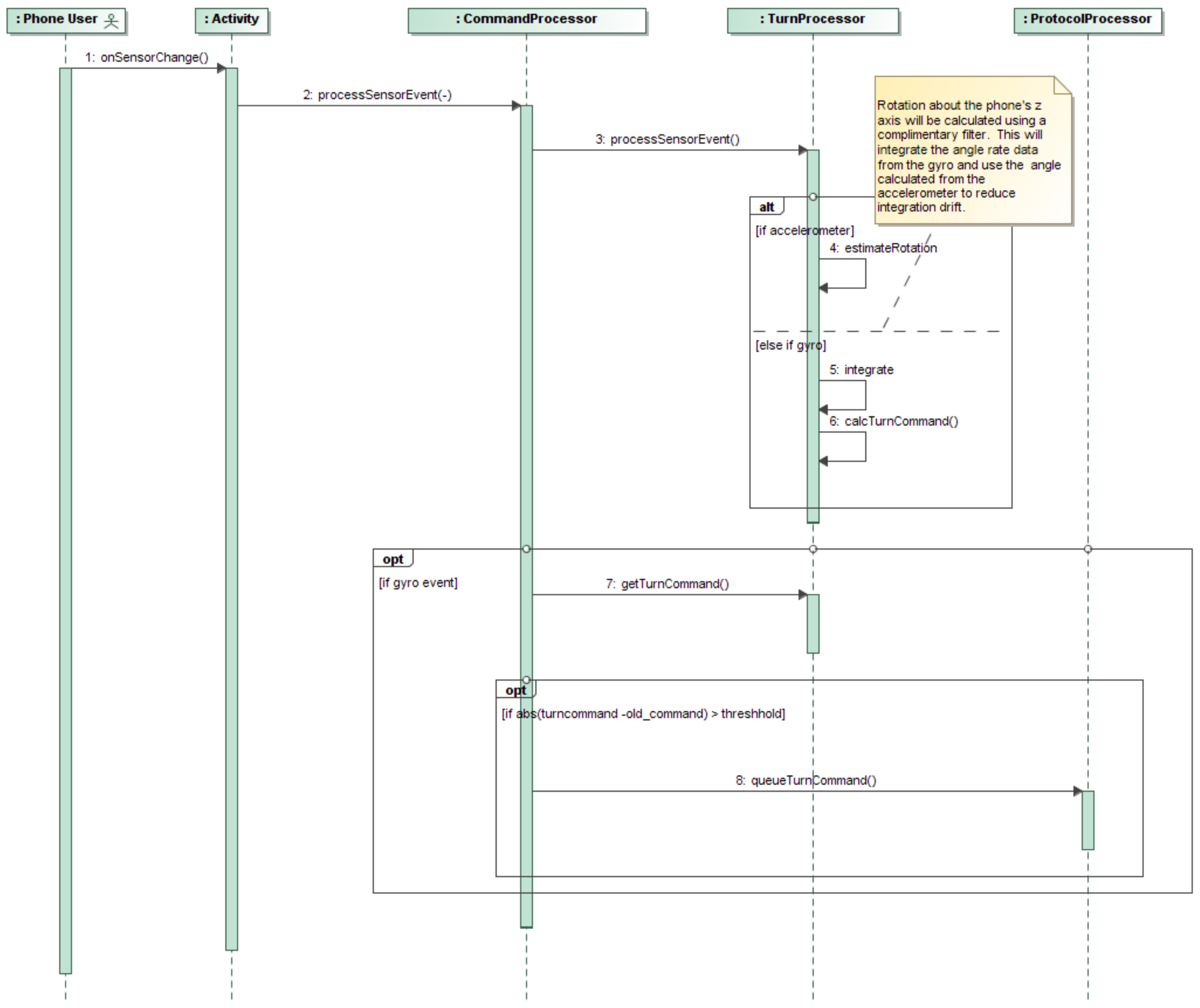


Figure 9: Command turn (Gyro) Use Case Realization

1. The user rotates the phone left or right. The activities onSensorChanged callback is called.
2. The activity notifies the Command Processor of the sensor event.
3. The Command Processor calls the Turn Processor.
4. If the sensor event type is from the accelerometer, the Turn Processor will approximate the rotation angle about the phone's Z axis (pointing out of the phone's screen, see link). This rotation angle will be saved for future use.
5. If the sensor event type is from the gyro, rotation about the phone's z axis will be calculated using a complimentary filter. This will integrate the angle rate data from the gyro and use the angle calculated from the accelerometer to reduce integration drift.

6. The Turn Processor will calculate the desired turn command from the reported rotation angle. Turn Commands will vary from +100 (full left turn) to -100 (full right turn). A rotation angle of 90 will correspond to a turn command of 100, a rotation angle of -90 will correspond to a turn command of -90, and the turn command will be linearly mapped in between those values.
7. The Command Processor will get the calculated turn command from the Turn Processor
8. The Command Processor will compare the newly generated command to the command that had been previously sent to the Protocol Processor. If the difference is greater than the threshold (5) a new command will be sent to the Protocol Processor.

5.5 Set Power Level (Accelerometer)

5.5.1 Use Case

Summary:
The user commands for the EV3 to increase or decrease power (speed) by moving the phone forwards or backwards.

Realizes Requirement:
A.2.4

Actors:
Phone User

Preconditions:
Startup has completed and user is at the main control screen. User is holding phone in portrait fashion.

Description:
The user will accelerate the phone forward to command an increase in power. The user will accelerate the phone backwards to command a decrease in power (through reverse power). The REV3 will take the user's interaction and translate it to the proper command to be sent to the EV3 Proxy.

Exceptions:
Disconnect: The REV3 application is disconnected from the EV3 Proxy. Transition to the Disconnect use case.

Post Conditions:
Power commands sent to the Protocol Processor are ready to be sent to to the EV3 Proxy.

5.5.2 Sequence Model

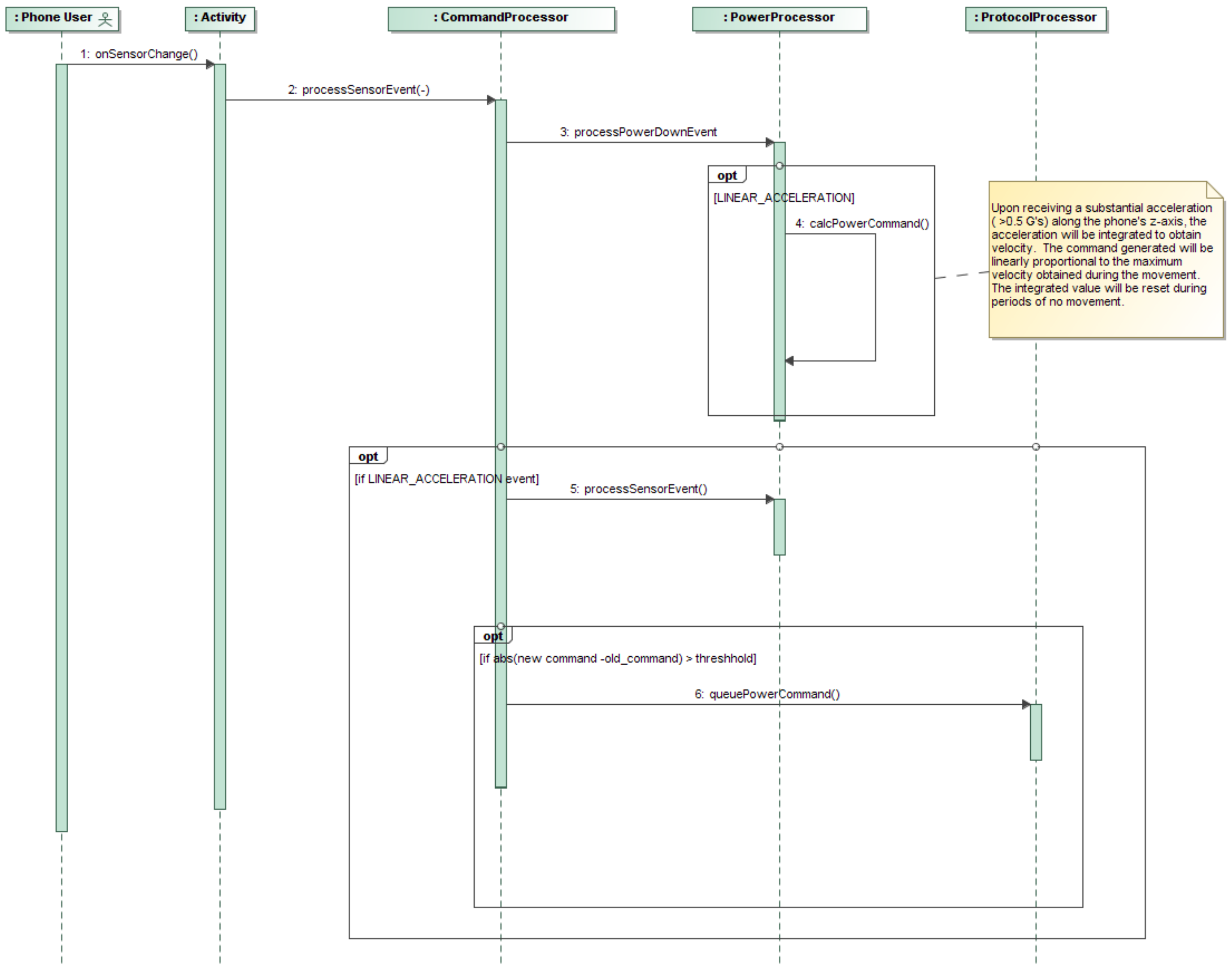

: Phone User
: Activity
: CommandProcessor
: PowerProcessor
: ProtocolProcessor
1: onSensorChange()
2: processSensorEvent(-)
3: processPowerDownEvent
opt
[LINEAR_ACCELERATION]
4: calcPowerCommand()
Upon receiving a substantial acceleration (>0.5 G's) along the phone's z-axis, the acceleration will be integrated to obtain velocity. The command generated will be linearly proportional to the maximum velocity obtained during the movement. The integrated value will be reset during periods of no movement.
opt
[if LINEAR_ACCELERATION event]
5: processSensorEvent()
opt
[if abs(new command -old_command) > threshhold]
6: queuePowerCommand()

Figure 10: Set Power Level (Accelerometer) Use Case Realization

1. The user accelerates the phone forwards (or backwards). The activities onSensorChanged callback is called.
2. The activity notifies the Command Processor of the sensor event.
3. The Command Processor calls the Power Processor.
4. The Sensor Power Processor will calculate the desired power command. Power Commands will vary from +100 (full forward) to -100 (full backward). Upon receiving a substantial acceleration (>0.5 G's) along the phone's z-axis, the acceleration will be integrated to obtain velocity. The command generated will be linearly proportional to the maximum velocity obtained during the movement. A value of 4 G's will correspond to +100 power command, -4 G's will correspond to a -100 power command. The integrated value will be reset during periods of no movement.
5. The Command Processor will get the calculated power command from the Power Processor

6. The Command Processor will compare the newly generated command to the command that had been previously sent to the Protocol Processor. If the difference is greater than the threshold (5) a new command will be sent to the Protocol Processor.

5.6 Transmit Commands

5.6.1 Use Case

Summary:
The REV3 application will send the commands to the EV3.

Realizes Requirement:
B.2.1, A.2.2 , A2.4, B.2.5

Actors:
EV3Proxy

Preconditions:
Command has been queued to be sent.

Description:
As commands are generated, they will be sent to the EV3 proxy.

Exceptions:
Disconnect: The REV3 application is disconnected from the EV3 Proxy. Transition to the Disconnect use case.

5.6.2 Sequence Model

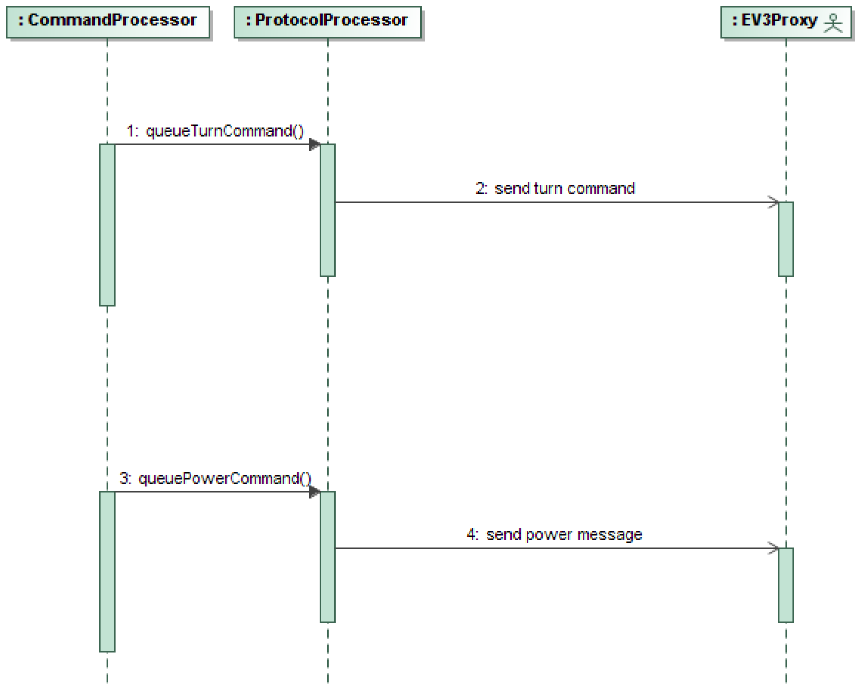


Figure 11: Transmit Command Use Case Realization

1. The turn or power command will be queued to the Protocol Processor
2. The Protocol Processor will forward the command on to the EV3 Proxy.

5.7 Display Delta Heading

5.7.1 Use Case

Summary:
The REV3 will receive delta heading (the change in heading since startup) from the EV3. REV3 will display this to the screen as an indicator arrow.

Realizes Requirement:
A.2.3

Actors:
Phone User, EV3Proxy

Preconditions:
Startup has completed and user is at the main control screen.

Description:
The EV3 Proxy will periodically send its change in heading over to the REV3 application. The application will listen on its own thread and will display the updated information to the user's phone screen.

Exceptions:
Disconnect: The REV3 application is disconnected from the EV3 Proxy. Transition to the Disconnect use case.

Post Conditions:
Power commands sent to the Protocol Processor are ready to be sent to to the EV3 Proxy.

5.7.2 Sequence Model

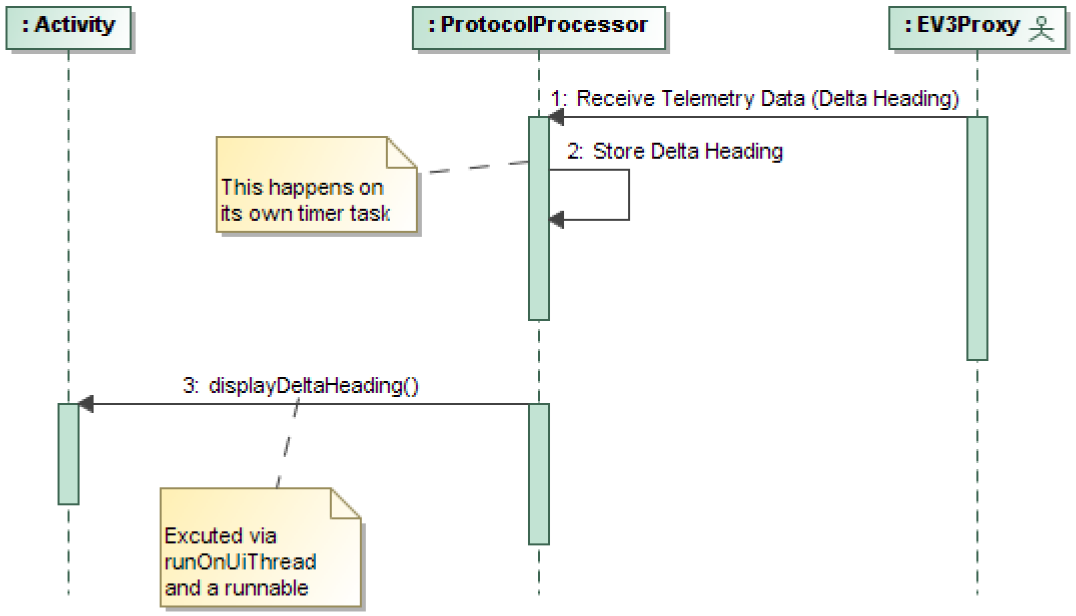

: Activity
: ProtocolProcessor
: EV3Proxy
1: Receive Telemetry Data (Delta Heading)
2: Store Delta Heading
This happens on its own timer task
3: displayDeltaHeading()
Excuted via runOnUiThread and a runnable

Figure 12: Display Delta Heading Use Case Realization

1. The Protocol Processor will listen on a separate task for incoming telemetry data (delta heading) from the EV3 Proxy.
2. The delta heading information will be stored by the Protocol Processor.
3. Periodically the Protocol Processor will call the Activity's displayDeltaHeading function. The activity will update delta heading indicator arrow. This will be executed in the main UI thread.

5.8 Disconnect

5.8.1 Use Case

Summary:
The REV3 is disconnected from EV3.   REV3 will bring up the configuration dialog and prompt the user to reconnect.

Realizes Requirement:
A.1.2

Actors:
Phone User, EV3Proxy

Preconditions:
The REV3 receives a SocketException

Description:
Upon receiving a SocketException the REV3 application will terminate any processing that it is currently conducting. The REV3 will bring up the Configuration Dialog prompting the user to reconnect.

Exceptions:
Unable to Connect: If the user is unable to connect, the configuration dialog will reset and the user will be prompted to renter the IP address and port.

Post Conditions:
The REV3 application is waiting at the main control screen, ready to process in the user's commands.

5.8.2 Sequence Model

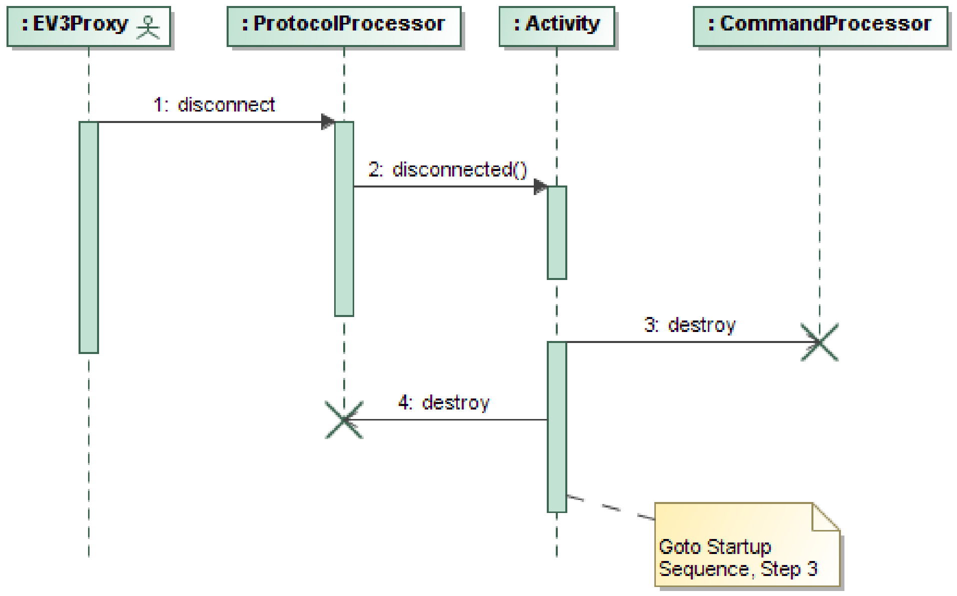


Figure 13: Disconnect Use Case Realization

1. The EV3 Proxy is disconnected from the REV3 application. The REV3 application will receive a SocketException.
2. The ProtocolProcessor will notify the main activity that it has been disconnected. Use runnable and runOnUIThread.
3. The activity will destroy the Protocol Processor. The Sequence will then transition to the Start Up sequence, Step 3.

6. Testing Strategy

A scenario based testing strategy will be used to verify that all use case and requirements have been met. Unit level tests will also be developed for all classes. This tests will be run using the AndroidJUnitRunner framework and Espresso (for UI interactions).

The EV3 Proxy will record all incoming messages that are intended to be sent to the EV3. It will also be capable of playing back a hand generated file intended to mimic the delta heading information received from the EV3. The Command Processor will also log any sensor events it receives. All sensor events will be logged in a single file with all events timestamped. Matlab scripts will be develop to allow the developer to inspect the recorded inputs and outputs. Once recorded data inputs and outputs are verified that they are performing as expected by the developer, they will be made available to be used as a part of the test framework.

Use Case Tested	Requirements Addressed	Approach
Startup	B.1.1	Scenario based test using AndroidJunitRunner and Espresso.
Disconnect	A.1.2	Scenario based test using AndroidJunitRunner and Espresso. AndroidJunitRunner will be used SocketException to the ProtocolProcessor and to ensure that it reacts properly.
Set Power Level (GUI)	B.2.1	Scenario based test using AndroidJunitRunner and Espresso. EV3Proxy outputs are recorded. Initial tests are verified by hand. Subsequent tests are compared against initial output files.
Command Turn (GUI)	A.2.2	Scenario based test using AndroidJunitRunner and Espresso. EV3Proxy outputs are recorded. Initial tests are verified by hand. Subsequent tests are compared against initial output files.
Display Delta Heading	A.2.3	Scenario based test using AndroidJunitRunner and Espresso.
Set Power Level (Accelerometer)	A.2.4	Scenario based test using AndroidJunitRunner. Initial tests are verified by hand with CommandProcessor sensor inputs and EV3Proxy outputs recorded. On subsequent tests, Command Processor inputs are replayed, outputs are compared against initial output files. Should include scenarios with phone being used in multiple orientations.
Command Turn (Gyro)	B.2.5	Scenario based test using AndroidJunitRunner. Initial tests are verified by hand with CommandProcessor sensor inputs and EV3Proxy outputs recorded. On subsequent tests, Command Processor inputs are replayed, outputs are compared against initial output files. Should include scenarios with phone being used in multiple orientations.
Transmit Commands	B2.1,A2.2,A2.4,B2.5,B3.1,B3.2,A3.3	Command and Set Power Level commands will be sent to the ProtocolProcessor using AndroidJUnitRunner. AndroidJUnitRunner will be used to verify proper command is being sent out at proper rate.

Table 3: Test Strategy by Use Case

7. Implementation Plan

The below table summarizes the various tasks by build release. Three build releases are planned. An Alpha build will be released on March 11, 2016. The alpha release will have the initial UI developed, the user will be able to send commands to the EV3 proxy via the turn command (gyro) use case. A Beta build will be released on April 15, 2016. All “B” level requirements will be implemented and tested for this build. The user will have the capability to issue turn and power commands via the UI, and command turns by using the gyro. The final release will be on May 4, 2016. This release will fix any bugs discovered in the Beta release. The set power level (accelerometer) and disconnect "A" level use case will be implemented as well.

Task	**Target Release**	**Requirements Addressed**	**Responsible Team Member**
Initial Drive Control Screen UI Developed	Alpha	B.2.1,A.2.2,A.2.3 (Partial)	Bryan Walsh
Drive Control UI Activity Callbacks Implemented	Alpha	B.2.1,A.2.2,A.2.3 (Partial)	Bryan Walsh
Configuration Dialog Implemented	Alpha	B.1.1 (Partial)	Bryan Walsh
Command Processor Stubbed Out	Alpha	X.2.X (Partial)	Bryan Walsh
Protocol Processor Stubbed Out	Alpha	X.3.X (Partial)	Bryan Walsh
Protocol Finalized, Command Recording Implemented	Alpha	B.3.1,B.3.2 (Partial)	Bryan Walsh
Turn Command (Gyro) Use Case Implemented	Alpha	B.2.5 (Partial)	Bryan Walsh
Prepare Alpha Release	Alpha		Bryan Walsh
Set Power Level (GUI) Use Case Implemented	Beta	B.2.1 (Partial)	Bryan Walsh
Drive Control Screen UI Finalized	Beta	B.2.1,A.2.2,A.2.3 (Partial)	Bryan Walsh
Protocol Processing Tested (Transmit Command Use Case)	Beta	B.3.1,B.3.2 (Complete)	Bryan Walsh
Start Up Use Case Testing Implemented	Beta	B.1.1 (Complete)	Bryan Walsh
Turn Command (Gyro) Use Case Testing Implemented	Beta	B.2.5 (Complete)	Bryan Walsh
Set Power Level (GUI) Use Case Testing Implemented	Beta	B.2.1 (Complete)	Bryan Walsh
Turn Command (GUI) Use Case Implemented with Tests	Beta	A.2.2 (Complete)	Bryan Walsh
Conduct Beta Release Testing	Beta		Bryan Walsh
Prepare Beta Release	Beta		Bryan Walsh
Display Delta Heading Use Case Implemented With Tests	Final	A.2.3, A3.3 (Complete)	Bryan Walsh
Disconnect Use Case Implemented With Tests	Final	A.1.2 (Complete)	Bryan Walsh
Set Power Level (Accelerometer) Implemented With Tests	Final	A.2.4 (Complete)	Bryan Walsh
Conduct Final Release Testing	Final		Bryan Walsh
Prepare Final Release	Final		Bryan Walsh

Table 4: Tasks by Release