Team Big ARMs

Vex Robot Phase 3
Project Report

Version <1.1>

Revision History

Date	Version	Description	Author
12/02/13	1.0	Modified template to fit our rubric requirements. Modified few sections to make details correct.	J. Jones
12/04/13	1.1	Modified sections 3.1 and added Figures 1.4 and 1.5. Added the Risk Mitigation for tasks in Phase 3. Created section 3.10.	SJ Kim

Table of Contents

1. Introduction 4
   - 1.1 Purpose 4
   - 1.2 Problem Statement 4
   - 1.3 Definitions, Acronyms and Abbreviations 4
   - 1.4 Equipment (Hardware) 4
   - 1.5 SDEs Used 5
   - 1.6 Overview 5

2. Project Specifications/Requirements 5

3. Project Design and Implementation 6
   - 3.1 Robot Mechanical Design 6
   - 3.2 Getting to Know Android and VEX 6
   - 3.3 Robot Movement 6
   - 3.4 Obstacle Avoidance 7
   - 3.5 Android to Android Communication 7
   - 3.6 Android to VEX Communication 7
   - 3.7 Android GPS 8
   - 3.8 Android GUI 9
   - 3.9 VEXpro and Sensors 9
   - 3.10 Navigation 10

4. Test 11
   - 4.1 Risk Mitigation 11
   - 4.2 Conclusion 13

5. References 14

6. Diagrams, Charts, and Figures 15

1. Introduction

1.1 Purpose

This Project Report provides an overview of the project, goals, requirements, risk analysis and the progress of the project.

1.2 Problem Statement

Our goal this semester is to build a robot that can navigate a designated course autonomously using a variety of embedded systems. To meet this goal, we use a four-wheeled vehicle controlled by a VEX microcontroller to traverse the course, while Android-powered devices send proper instructions to the VEX microcontroller to ensure that the vehicle is on the correct course.

1.3 Definitions, Acronyms and Abbreviations

- ADT – Android Development Tools
- API – Application Programming Interface
- GPS – Global Positioning System
- GUI – Graphical User Interface
- HTML – HyperText Markup Language
- I/O – Input/Output
- IDE – Integrated Development Environment
- IP – Internet Protocol
- MAC – Media Access Control
- SDE – Software Development Environment
- SSH – Secure Shell
- TCP – Transmission Control Protocol
- UDP – User Datagram Protocol
- UI – User Interface

1.4 Equipment (Hardware)

- Nexus 4 (S/N: 303kped532012)
- Nexus 7 (S/N: 015d21d93a20061c)
- VEXpro ARM9 Microcontroller
- 2 x VEXnet 802.11g (Model #: W541U)
- D-Link Wireless Router (Model #: DIR-615)
- 4 x VEX Two-wire motor
- 2 x VEX Bump Sensors
- VEX Ultrasonic Sensor
- 2 x VEX Touch Sensors

1.5 SDEs Used

- ADT
- Terk IDE
- Eclipse

1.6 Overview

The Project Report is organized into project specifications/requirements, project design, and project summary. The project specifications/requirements section describes the objectives and constraints of the project specifications, and the extent to which each specification needs to be met. The project design contains information about the robots mechanical design, Android to Android communication, Android to VEX communication, Android GPS, sensors, and application GUIs. The project summary section contains the review, test, and risk mitigation section; which contains risks for each tasks and how to mitigate those risks.

2. Project Specifications/Requirements

This section contains high-level information about the project and its specifications/requirements. To what extent each requirement was met will be discussed in the Review section of this document.

- Communication between the Android Tablet, Phone, and the VEX shall configure automatically.
- The Android Tablet shall allow tilt control of the VEX movement.
- The robot shall visit all waypoints in the central mall course.
- The robot shall return to the tablet when the “come home” button is pressed.
- The project team shall take all necessary precautions to prevent damage to the ARM9 VEX Microcontroller, the Nexus 7 Tablet, and Nexus 4 Phone including, but not limited to keeping all devices properly locked in storage when not in use, executing caution when working with the devices, and ensuring proper casing and protection have been fastened securely on the devices.

3. Project Design and Implementation

This section contains information about the design of the project and how each piece of the project plays its role.

3.1 Robot Mechanical Design

Two factors drove the design decisions for our VEX robot. The first was the size of the VEX microcontroller: our robot would need to have a large enough body to comfortably accommodate the microcontroller, as well as the additional parts that may need to be situated on the robot. The second factor was the power necessary to climb a 10% grade incline while carrying the aforementioned microcontroller, batteries, and more. Another consideration was the potentially adverse weather conditions at the end of the semester: our robot may need to be able to drive through snow-covered sidewalks. With these criteria in mind, we designed our robot to have a sizable rectangular base with large rubber wheels. Each wheel is independently powered by its own two-wire motor without using any gears, to ensure that the vehicle has enough power to perform its required functions. The motors are controlled by the VEX microcontroller, which is securely bolted in the center of the vehicle body. A bump sensor has been placed on each of the front and back ends of the robot. There is enough space to place the battery pack and the Nexus 4 on the body of the vehicle, which will likely lead to an even weight distribution on the top of the device. Six rechargeable AA batteries inside the battery pack powers the microcontroller, which in turn powers the four motors. Our completed VEX robot can be seen in Figure 1.1, Figure 1.2, and Figure 1.3. In Phase 2, we discovered that the magnetic sensors in the Android devices receive interference from the VEX microcontroller and motors. To mitigate this issue, we attached a harness to one end of the robot to distance the Android phone from the main body of the robot. The harness is composed of plastic and high-density foam, and bolted to the steel frame. The revised VEX robot can be seen in Figure 1.4 and Figure 1.5.

3.2 Getting to Know Android and VEX

In order to control the robot from an Android device, the VEX microcontroller must be able to listen to the Android commands, then control the motors to move the robot as commanded. The Android devices must in turn be able to communicate with each other, and send commands to the microcontroller. Before beginning to implement these functionalities, we began to familiarize ourselves with each of the embedded devices by reading relevant documentation and experimenting with the development environment for each device. As a result, we were successful in displaying the historic “Hello World!” message on all three of our main devices. Figure 2.1 shows “Hello World!” running on the Nexus 4[4], and Figure 2.2 and Figure 2.3 show “Hello World!” running on the Nexus 7[4] and the VEX microcontroller, respectively.

3.3 Robot Movement

Currently our robot can move forward, backwards, turn left, and turn right. Since our robot is not “rack and pinion” when it turns all the motors must move at once to turn left and turn right. Therefore, the robot turns by moving through a sequence of motions; stop-turn-stop. This sequence allows us to accurately keep track of the robots movement since there is an acceleration delay from when the VEX receives the command and when the robot is in full turn. Thanks to the four-wheel-drive design, and rugged rubber wheels that provide ample traction, our robot is

fully capable of climbing a ten percent grade. The weight distribution and low center of gravity in the shape of the robot also provide extra stability on a sloped surface. Figure 3.1 shows our robot as it climbs the slope between Haggerty Hall and Olin Engineering. With some tweaking, we were able to get our robot to move in a straight line for twenty meters. This was achieved by tweaking the speed of some of the motors so that all the speeds were identical.

Another method of robot control is using the Android devices gyroscope and accelerometer. Using those two sensors, we are able to use the tablet as a "tilt" controller to move the VEX robot. Based on the tilt of the tablet (ie. forward is forward, forward and right moves the robot forward and right, etc.) the robot moves in that direction. The tilt control is implemented by converting gyroscope readings into angles. The motor control class on the VEX determines the speed of the motors based on the degrees received from the application running on the tablet.

3.4 Obstacle Avoidance

Obstacle avoidance was implemented by adding sensors to the VEX robot which are connected through the digital I/O. When the sensors respond to obstacles in the area, the input interrupts the current process and stops the VEX. Because the sensors are interrupt-driven, we have ensured that the processing overhead for responding to obstacles is minimized. For more information about sensors, please see Section 3.9.

3.5 Android to Android Communication

In the subsequent phases of our project, we envision that the Nexus 4 phone will exchange messages with the Nexus 7 tablet. This will allow the Nexus 4 to be placed on the robot vehicle to help navigate the designated route, while the Nexus 7 will enable us to send commands to the phone from a comfortable distance. There are two different mediums that could be used for this type of communication; Bluetooth and Wi-Fi. Bluetooth can connect directly to the device without fear of interruption, but it is only limited to a distance of thirty feet. Wi-Fi can cover greater distances between the devices as long as both are connected to the same network. If we use the MU_Wireless or MU_Admin network on campus, the range is virtually unlimited. If we are to use our own wireless router out in the field, they would have a range of approximately 150 to 300 feet. Using Wi-Fi to communicate between the devices will enable us to stay at the start point as our robot makes its way around the given course.

A java application was built implementing a client-server framework.[5] The server is the phone and the client is the tablet. The UI allows us to enter text into a text field[6], set the IP address of the server, and press the "send" button, Figure 4.1, the application then forwards the message to the server's correct IP address[8] and port.[2] The Android devices connect automatically to each other using assigned IP Addresses based on the devices MAC Address. Once received, the application then displays the message onto the screen, Figure 4.2.

3.6 Android to VEX Communication

For the Android to VEX communication we tried several different strategies. First, we attempted using a modified version of a VEX Bluetooth driver for Wi-Fi that we found from an online resource, but quickly discovered that it would not be a viable option as the existing Wi-Fi drivers already capture the wireless data. Next, we tried using an SSH library to build an app that would

use a secure shell to start a basic command line application on the VEX that would then parse text input, but the SSH library was a little tricky and seemed like overkill for our purposes.

Finally we realized that we should be able to use the existing Android Client application from the Android-Android communication with a basic server implemented in C/C++ on the VEX[9]. The existing Android application sent text based messages using TCP and writing a TCP server in C that parses text based messages was fairly simple and at first it was. However, after sending a few commands the communication would mysteriously stop working. We discovered the issue was that the server on the VEX thought the Android device was sending a stop message and stopped listening to the port for incoming messages. Switching to UDP became our best option.[1] We switched both the C server and Android client to UDP, and the communication worked as expected, Figure 5.1. In order to connect the Android to the VEX IP Addresses are automatically assigned based on the devices MAC Address.

On the Android side we started with the text field based client app from the Android-Android communication. This was sufficient, especially for initial testing, but it became difficult to send commands rapidly when working in tight spaces or if the speed of the robot was set relatively high. To take care of this and to limit the commands to make sure the commands are sent spelled correctly, in the correct case, we built a GUI for driving the robot. Since the original Android app was built modularly, building the GUI[7] app simply meant removing the text based interface and putting on the graphical interface. Figure 5.2 shows the user interface of the GUI-driven Android client.

In parallel with the server-client communication development, we also developed a web interface for communicating with the VEX. To do this we added a new HTML page to the existing VEX website and an application that would run on the VEX and parse the commands from the buttons that can be pressed on the web interface. In this way, any device with a web browser on the same network as the VEX can send movement commands.

In both the web interface and the server-client implementation we included stop buttons that will allow us to stop the VEX remotely in case of an emergency, or simply for basic navigation. Additionally for remote stop we will start the VEX applications using an Android SSH app, and can emergency stop the VEX by using Ctrl-C, which will end all inputs to the motors and stop the robot.

3.7 Android GPS

In order for the VEX robot to move from waypoint to waypoint, the Android devices must be able to determine their current location. This could be accomplished using the built-in GPS, gyroscope, and magnetometer in the Android devices, but to utilize them, a few preparatory steps were necessary. Google Play Services and Google Maps *.jar files were be added to the build path in our Eclipse SDE so that we could implement the Android GPS API. The Android devices must also allow the application to access special permissions to obtain the location settings via the GPS and Wi-Fi. When the application has the correct permissions, Android's locationManager[6] can be used to constantly display the phone's GPS coordinates.

One issue with constantly updating the phone's current position is that using GPS drains the

phone's battery. This could potentially be an issue if the application was for a few hours on end, but since it is only being used for an hour max, as long as the battery is fully charged when the application starts, battery life should be reliable.

In order for the robot to know which direction it is facing, the GPS application must calculate the bearing, the angle between the current facing direction and the direction to the other GPS waypoint. The bearing is calculated using the Algorithm 1. Once the VEX receives the bearing direction, it will turn left or right, then move straight to the next waypoint. Calculating the heading requires the use of the magnetometer and gyroscope. The gyroscope helps calculate the rotation of the phone, while the magnetometer calculates the strength of the Earth's magnetic field and helps find the phones orientation relative to true north. In Phase 2, we discovered that the VEX robot interferes with the magnetometer of the phone, thus giving us an incorrect bearing. This issue was resolved by attaching a harness to the robot to distance the VEX microcontroller from the Android phone. The harness is made of plastic and high-density foam. We are also using aluminum foil to partially shield the phone from electromagnetic interference; however, since the phone needs to be uncovered to ensure that it receives GPS and Wi-Fi signals, we found that the aluminum foil was not very effective.

3.8 Android GUI

As the Android phone will traverse the course with the VEX robot to help it navigate the course, the user will interact with the embedded systems through the interface of the Android tablet. To that end, we felt that having the Android tablet display a variety of information on its screen in addition to its functionalities as a remote controller device for the phone and VEX robot would be a great bonus feature. The GUI for our Android tablet application has an emergency stop button, simple robot controls (ie. forward, backward, left, right), a scrollable text area to view current IP address, port number, and incoming communications from another Android device, a map to view the last known location of the android devices, buttons for changing robot speed, selecting a waypoint, instructing the phone to "come home" to the tablet, running a program of waypoints, and to create new GPS waypoints, as well as a tilt control toggle switch. The simple remote control buttons have been hidden into a fragment that can be accessed at the touch of a button. Figure 8.2 shows the GUI described in this section. With this change, the UI is uncluttered by the control buttons when they are not in use. In response to user interaction with the GUI, the tablet application will send messages to the phone application using UDP.

In order to effectively and efficiently control the VEX robot, the Android to Android and Android to VEX applications must be merged into a dual purpose application that runs on the phone. This application should be able receive messages from another Android device, parse the message, then send a command to the VEXpro microcontroller. Although this app does not allow the user to directly interact with the phone, it displays pertinent information on the screen. The GUI of the phone application displays its IP address, port number, and instructions passed to the phone from the tablet GUI, and current location information such as latitude, longitude, facing direction, and distance to destination. Figure 8.3 shows this phone application.

3.9 VEXpro and Sensors

Currently we have one sensor implemented, the bump sensor, which is used for obstacle avoidance. The bump sensor is connected to the digital I/O ports of the microcontroller; as are

the rest of the sensors we are implementing. Another sensor that is being used is a touch sensor, seen in Figure 8.1, which is implemented the same way the bump sensor is. The touch sensor is placed at a height which is about the same level of the wheels in the front of the robot. This sensor acts as a form of obstacle avoidance in a way which allows us to move away from objects that the robot could not travel under. We found good use for this, because when we were testing if the robot could travel in a straight line it veered to the left and got stuck under a table. The third sensor we are using is the ultrasonic sensor. The ultrasonic sensor is used for the cliff test. The cliff test tells us whether the VEX is going to fall off of a surface that may damage it. The way this works is that the sensor constantly reads how far it is from the ground, how long the signal takes to leave then return back to the sensor. If the sensor reads that it is too far from the ground, meaning the signal took too long to come back, and interrupt will be sent and the robot will stop and reverse to avoid falling off a cliff/steep edge.

3.10 Navigation

For the VEX robot to navigate the designated course, it must traverse a series of waypoints that are defined along the course. In our implementation of this requirement, we stored a list of waypoints on the tablet application. When a button on the GUI is pressed to begin the program, the Tablet app gives the first waypoint coordinates to the phone. The phone uses its own location to navigate the robot to within a few-meter radius of the GPS coordinates. When the VEX robot reaches the waypoint, the phone signals back to the tablet application, which then gives the next waypoint in the list to the phone. The process is repeated until all waypoints are traversed and the robot reaches the final destination. Figure 9.2 shows a map of the area that the VEX robot is expected to traverse.

4. Test

To ensure that our project goals would be met in a timely manner, we dissected each goal into smaller tasks, and addressed each task in a modular approach. Each task, or functionality of the VEX or Android device, was tested individually for correctness and acceptable range of behavior before integrating it with other functionalities. The resulting unit was tested again to confirm that it performs the combined functionalities at the same or higher standard as the previous round of testing. For instance, the Android Client App that controls the VEX microcontroller with the corresponding C Server program on the VEX was created through a modular procedure that is outlined in Figure 6.1. As a result of this modular process, we were able to proceed to the subsequent testing phases with a high level of confidence.

4.1 Risk Mitigation

1. Android UI
    a. Risks: difficulty integrating our existing UI elements with the new requirements, and making the controls work well with the tablet instead of the phone
    b. Mitigation: Assign one person to be responsible for this early on and make it their primary task, allowing time for integration with the other subsystems later in the phase.
    c. Actual Mitigation: Integrated application in parts. One team member was in charge of the GUI and integration of the GPS and Server applications.
2. Communication between VEX, Phone, and Tablet
    a. Risks: There are problems running both a sending and receiving socket on the same Android device
    b. Mitigation: Complete this task early, doing research and using our existing code and expertise to ensure a functional final product
    c. Actual Mitigation: Merged both the Android to Android and Android to VEX code, then test it.
3. Android able to get GPS coordinates
    a. Risks: No team members have experience with using GPS API on Android.
    b. Mitigation: Assign one member to this task from the beginning of the phase to allow them to focus on this to the exclusion of other tasks. This member must begin researching as soon as the phase begins.
    c. Actual Mitigation: Assigned one student to research and implement GPS functionality into our Android2WayServer application.
4. Robot moves to a GPS coordinate.
    a. Risks: Getting GPS coordinates on the Android device must be complete before we can work on this task. If the robot does not move in a straight line this task becomes more difficult.
    b. Mitigation: Prioritize GPS coordinate retrieval and robot motion tasks early in the semester to allow sufficient time to complete this task.
    c. Actual Mitigation: Worked on the Android receiving and sending GPS coordinates. Got the VEX moving in a straight line. And calculated which direction the VEX should turn to get to the received GPS coordinate.
5. Robot moves in a straight line

a. Risks: Not many replacement mechanical parts are available.
b. Mitigation: Scavenge for spare parts, and try fixing the weight distribution on the robot to avoid needing replacement parts.
c. Actual Mitigation: Moved one motor slower than the other to get the VEX moving in a straight line.

6. Robot uses sensors
    a. Risks: The team has not yet had success using interrupts.
    b. Mitigation: Get interrupts working with the bump sensors before moving on to more complicated sensors.
    c. Actual Mitigation: Worked on getting the interrupts working with the bump sensors, then moved on to the ultrasonic sensors and touch sensors.
7. Cliff test
    a. Risks: The ultrasonic sensor must be integrated with the robot before this task can be attempted.
    b. Mitigation: Make the ultrasonic sensor the first sensor that is integrated in this phase so that parallel development can be done on both the cliff test and other sensors.
    c. Actual Mitigation: Ultrasonic sensor was integrated first, tested, then the other sensors were added.
8. Communication Auto-Configure
    a. Risks: We do not yet know what we will be using as an access point for the network since the goal is to have the robot be a completely encapsulated entity.
    b. Mitigation: Get the question of the network architecture answered early by Dr. Povinelli as the path we take will be largely determined by his decision of whether to power the routers on the robot or use a different solution.
    c. Actual Mitigation: Our devices were assigned static IP addresses by the devices' MAC addresses. Dr. Povinelli will provide a network with the predetermined static IP addresses available on demo day.
9. Android tilt control of robot
    a. Risks: Our current motor control class does not support how tilt control would intuitively function (degree of turning relative to the tilt of the device). No group members have implemented tilt control using Android devices in the past.
    b. Mitigation: A pair of people with both Vex and Android programming experience will be assigned this task from the beginning of the phase. The re-implemented motor control class will be tested separate from the tilt control to confirm it works before moving on to the tilt control.
    c. Actual Mitigation: The improved robot control class is able to control left and right wheels at different speeds, and is backwards compatible with old communication commands.
10. Robot completes course in central mall
    a. Risks: Currently facing direction on the phone while is sitting on the Vex does not work because of the interference from the metal frame of the Vex with the magnetometer in the phone.
    b. Mitigation: A pair of group members will be assigned to either finding a way to solve this problem so the facing direction can be determined despite the interference, or revising the GPS movement algorithm to use a different way to move toward the

destination.

c. Actual Mitigation: The VEX robot was modified to diminish the magnetic interference. As a result, the accuracy of the magnetometer improved immensely. In our tests, the magnetometer on the Android phone when placed on the plastic harness was only 5 degrees off from expected values. This is a tremendous improvement, since the phone placed directly next to the VEX microcontroller fluctuated between 30 to 200 degrees off from expected values.

11. Robot returns to the tablet when the "come home" button is pressed.
    a. Risks: Task is dependent on facing direction and GPS working correctly. If the robot cannot move to an arbitrary GPS coordinate then it will most likely not be able to move back to the tablet.
    b. Mitigation: A pair of group members is being assigned to the facing direction task as in requirement 10. Getting the current location of the tablet and sending it to the phone was accomplished during phase 2 in advance of this requirement.
    c. Actual Mitigation: The robot is able to "come home", similar to how it would navigate to a waypoint.

Top Risk: Our top risk is that the facing direction API for the Android phone does not work and we will need to find another way to get the robot to move to the GPS coordinates.

Plan to fix top risk: We plan to have this be the primary task from the start of the phase for a couple of our group members to make sure that development occurs early and there is sufficient time for troubleshooting, asking questions in class, and working outside before the weather gets too unpleasant to spend long periods of time outside.

Top Risk Resolution: With the improved directional accuracy from the plastic harness, the ability to adjust the left and right directional speeds from the updated robot movement code on the VEX, and the improved movement algorithm on the Android that takes advantage of these changes, we were able get the VEX robot to navigate to a waypoint determined by a GPS coordinate.

4.2 Conclusion

Our team has met all the necessary requirements for the third and final iteration of the project, and is in good shape to confirm that everything works in the field during the outdoor demonstration. The next stage will include improvements and additions to all of the requirements in this iteration. All the requirements of this phase have been met, as explained in detail in Section 3.

5. References

[1] (1999). *Sockets Tutorial* [Online] Available: http://www.linuxhowtos.org/C_C++/socket.htm

[2] (2009). *Android Socket Example* [Online]
Available: http://examples.javacodegeeks.com/Android/core/socket-core/Android-socket-example/

[3] (Oct. 3, 2011). *CQEGpioInt Class Reference* [Online]
Available: http://content.VEXrobotics.com/VEXpro_api/classCQEGpioInt.html

[4] (2013). *Building Your First App* [Online]
Available: http://developer.Android.com/training/basics/firstapp/index.html

[5] (2013). *Client Server Model – Architecture* [Online]
Available: http://www.tutorialspoint.com/unix_sockets/client_server_model.htm

[6] (2013). *Location Strategies* [Online]
Available: http://developer.android.com/guide/topics/location/strategies.html

[7] (2013). *User Interface* [Online]
Available: http://developer.Android.com/guide/topics/ui/index.html

[8] R. J. Lorimer. (Nov. 20, 2005). *General: Know How to Use InetAddress* [Online]
Available: http://content.VEXrobotics.com/VEXpro_api/classCQEGpioInt.html

[9] S. Walton. (2000). *simple-server.c* [Online]
Available: http://www.linuxhowtos.org/C_C++/socket.htm

6. Diagrams, Charts, and Figures

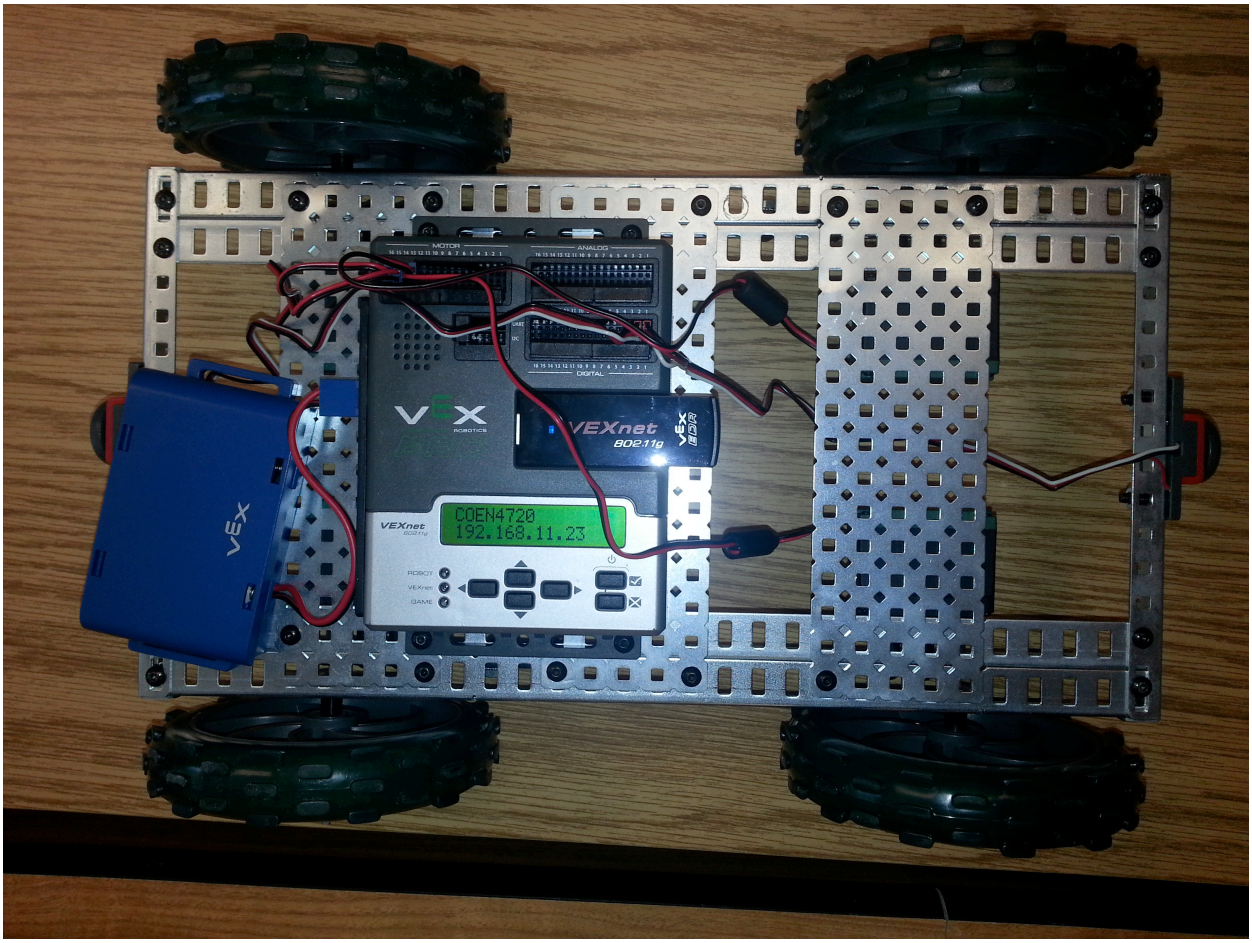

Figure 1.1 – Top View of VEX Robot

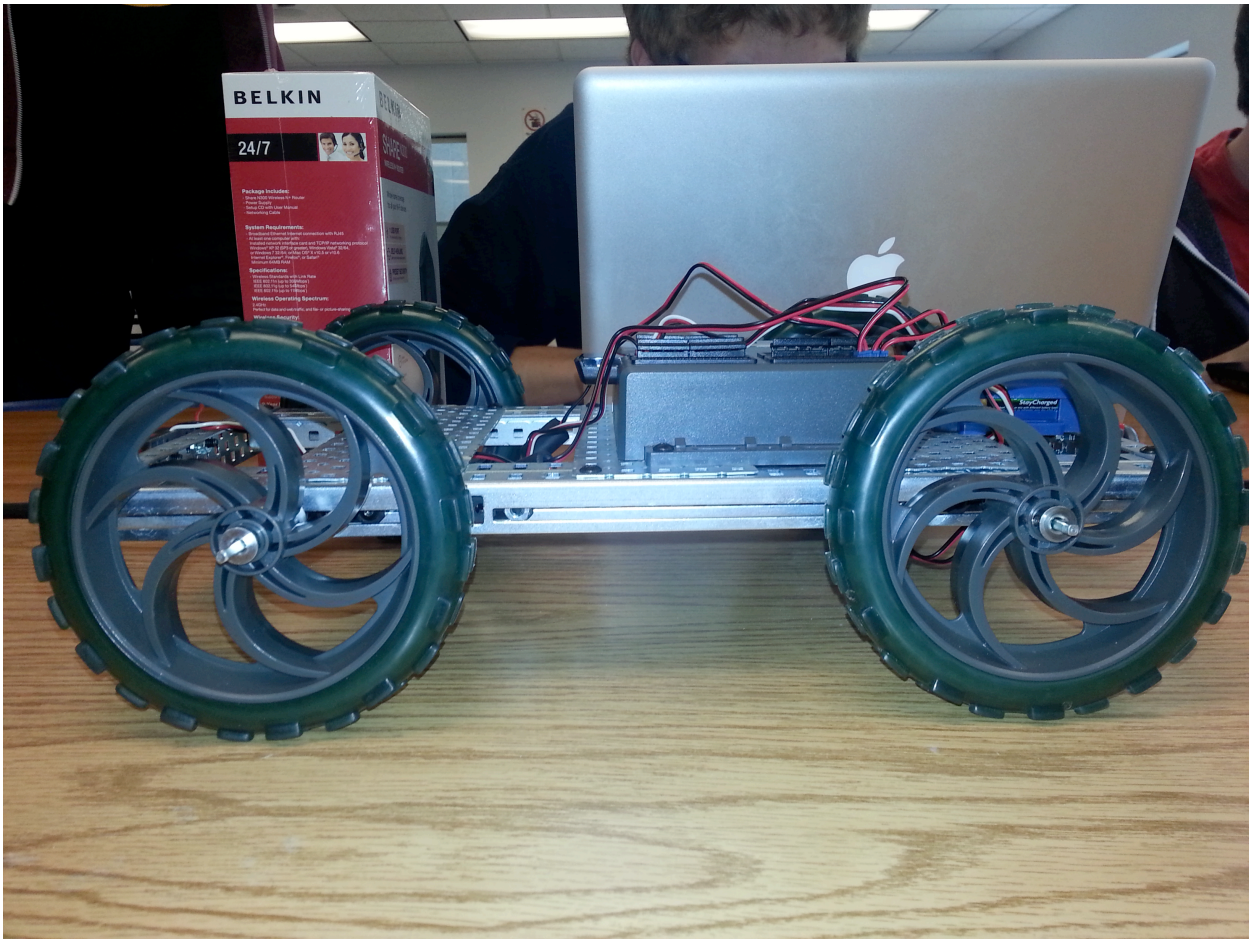

Figure 1.2 – Side View of VEX Robot

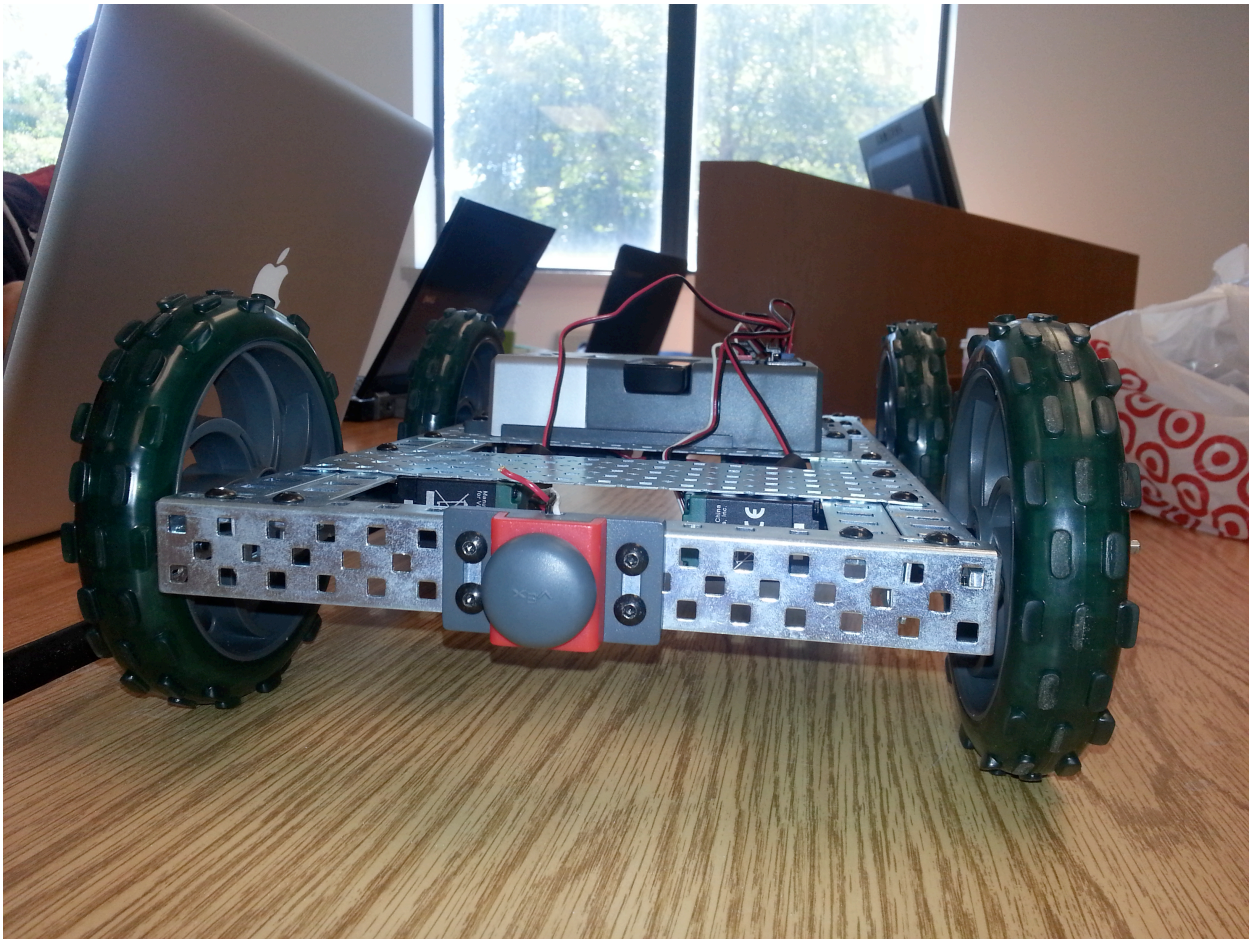

Figure 1.3 – Front View of VEX Robot

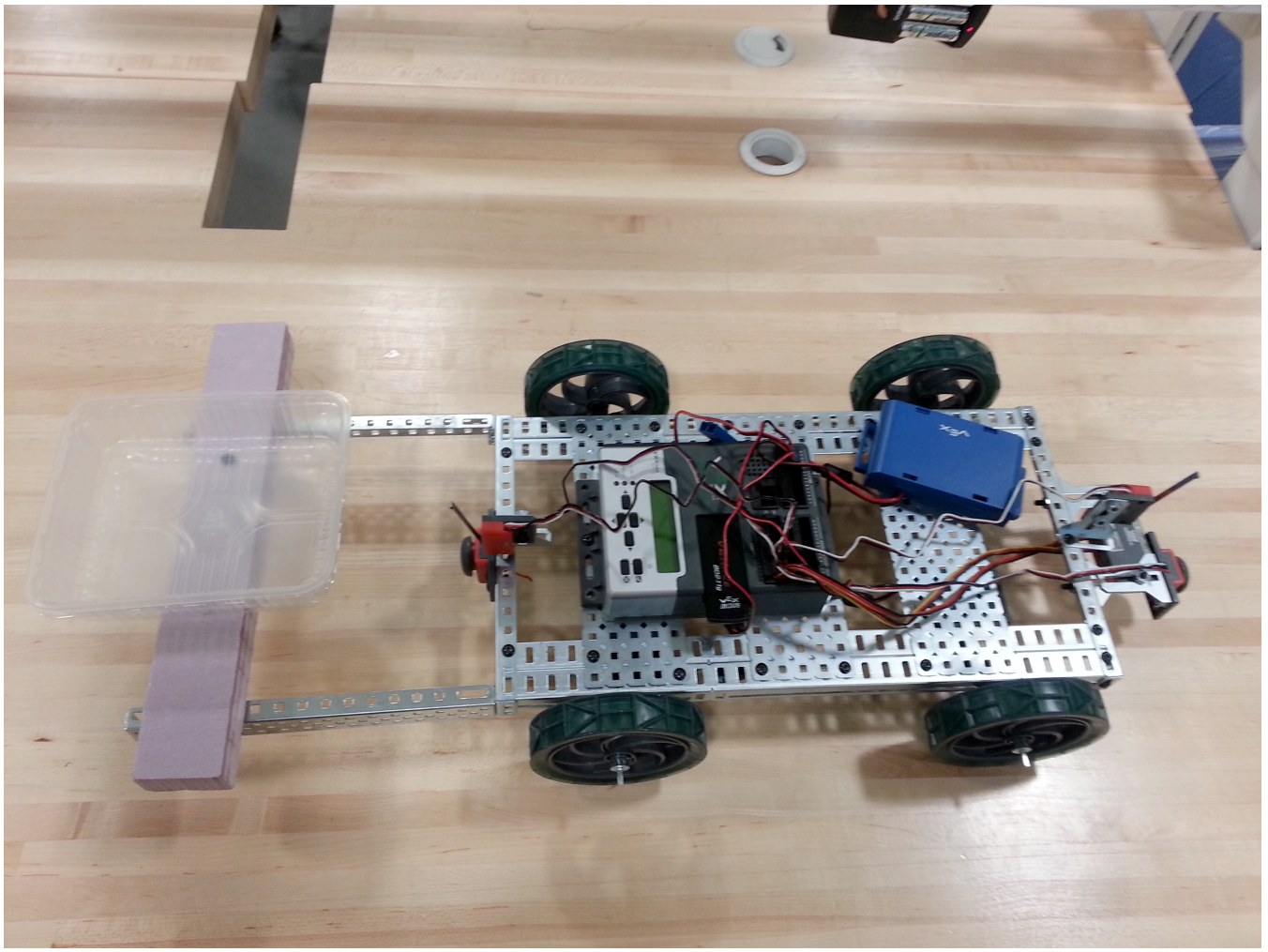

Figure 1.4 – Top View of the revised VEX robot

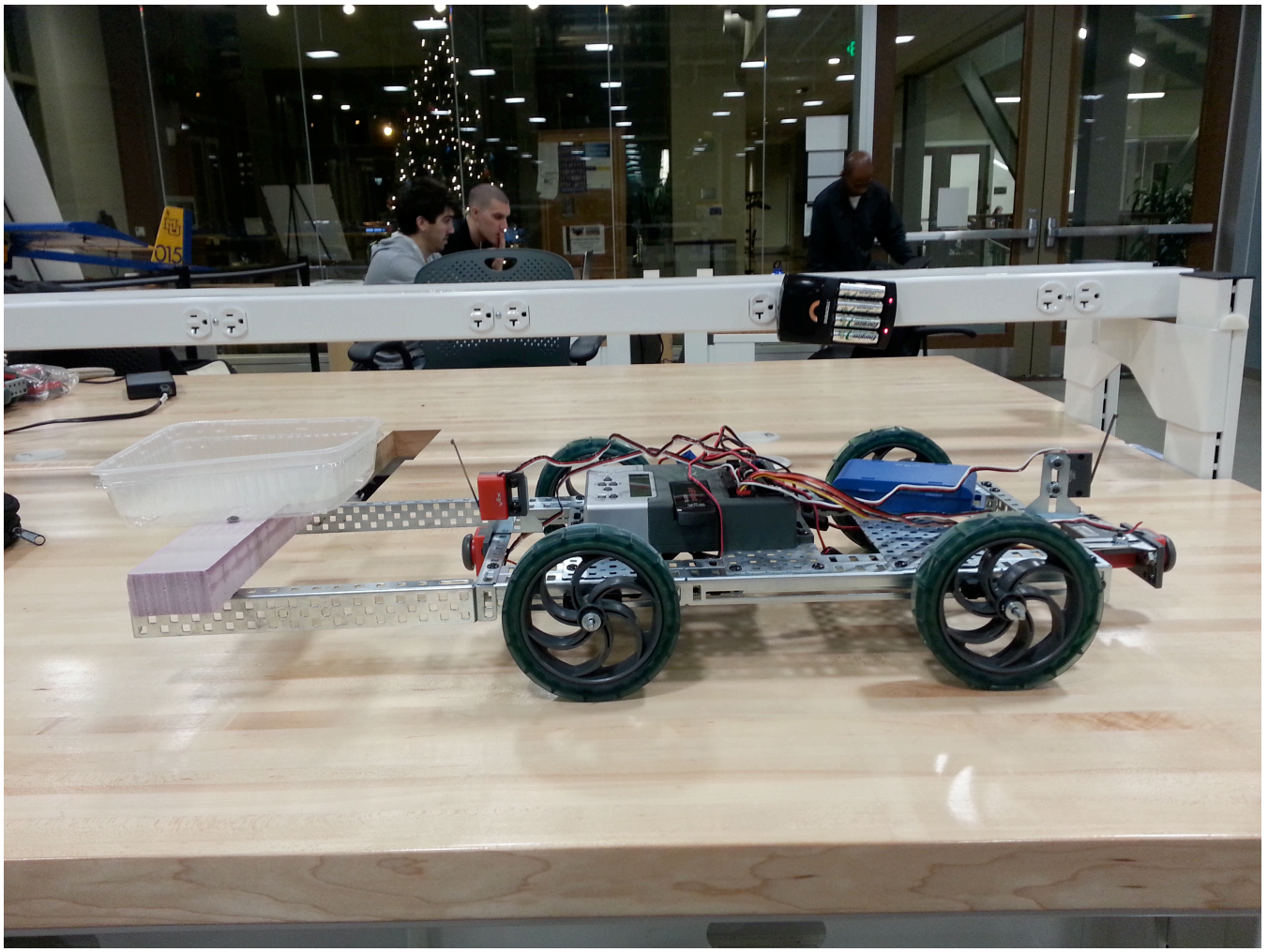

Figure 1.5 –Side View of the revised VEX Robot

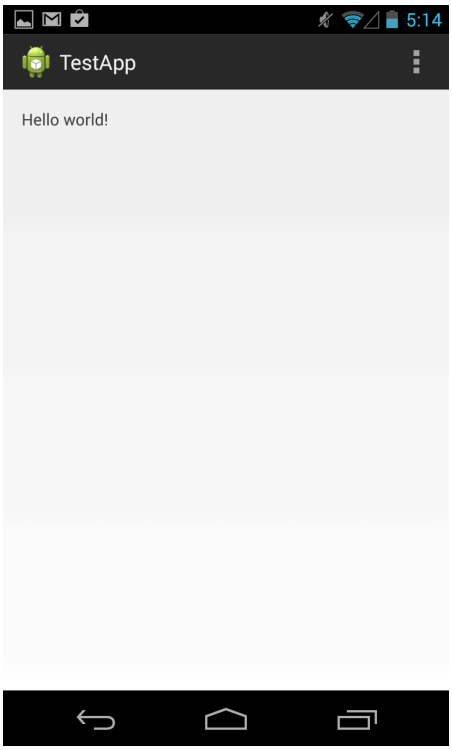

Figure 2.1 – “Hello World!” on the Nexus 4

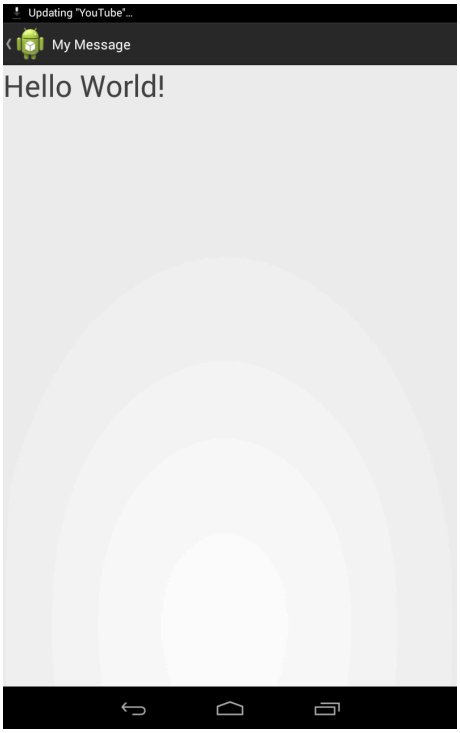

Figure 2.2 – “Hello World!” on the Nexus 7

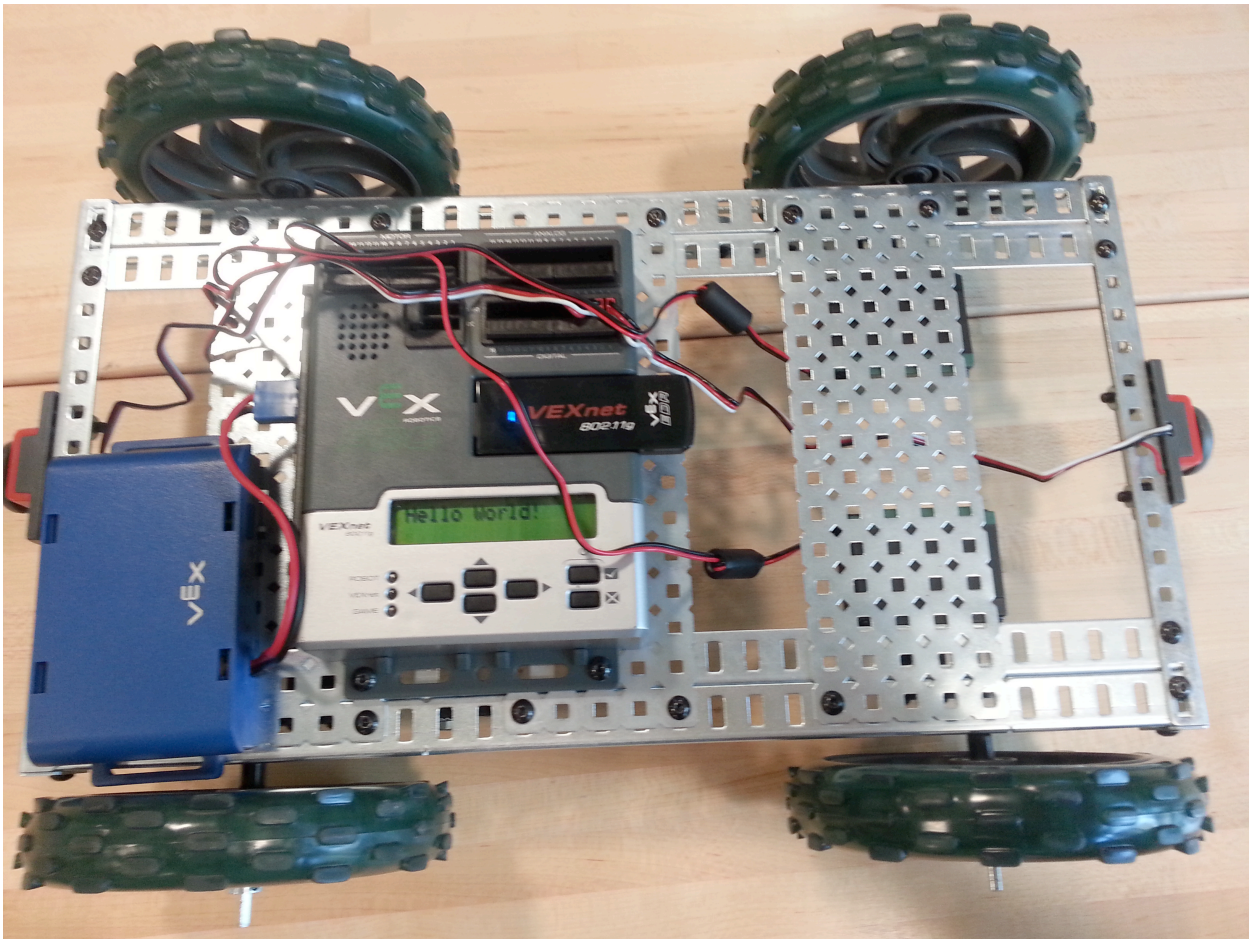

Figure 2.3 – “Hello World!” on the VEX Microcontroller

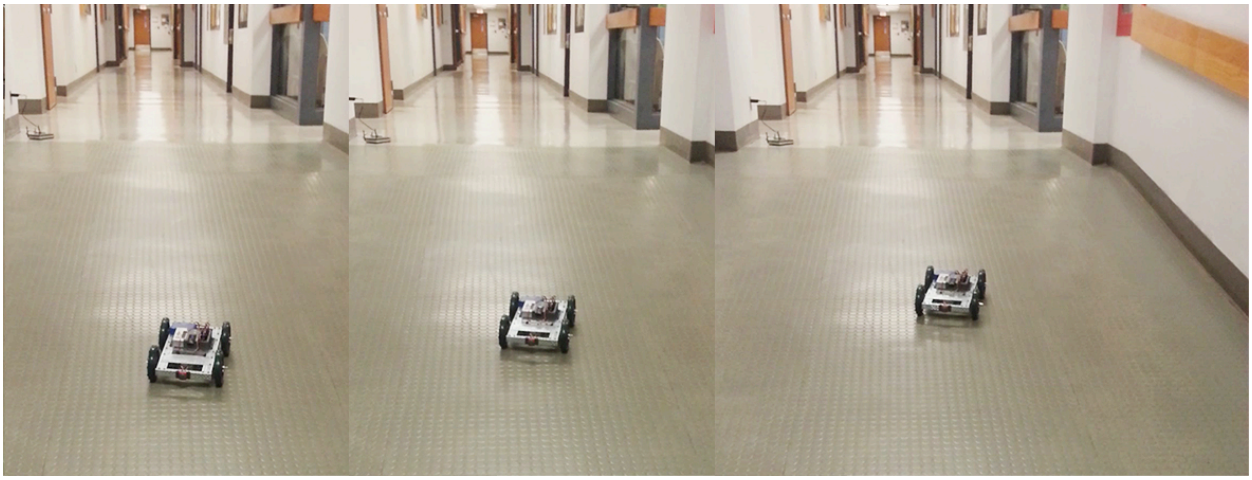

Figure 3.1 – Robot Climbing a 10% Grade

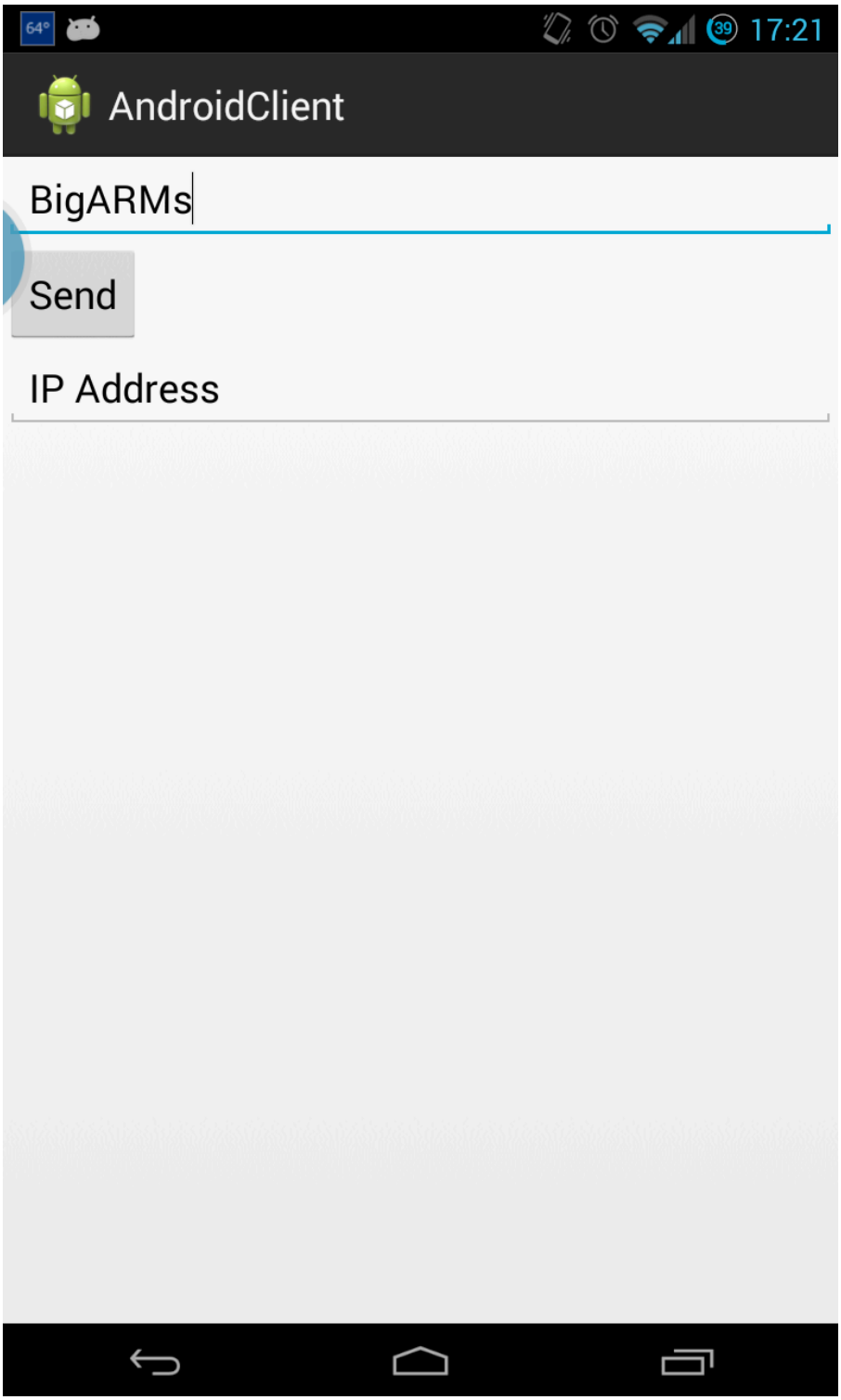

Figure 4.1 – Android Client Application

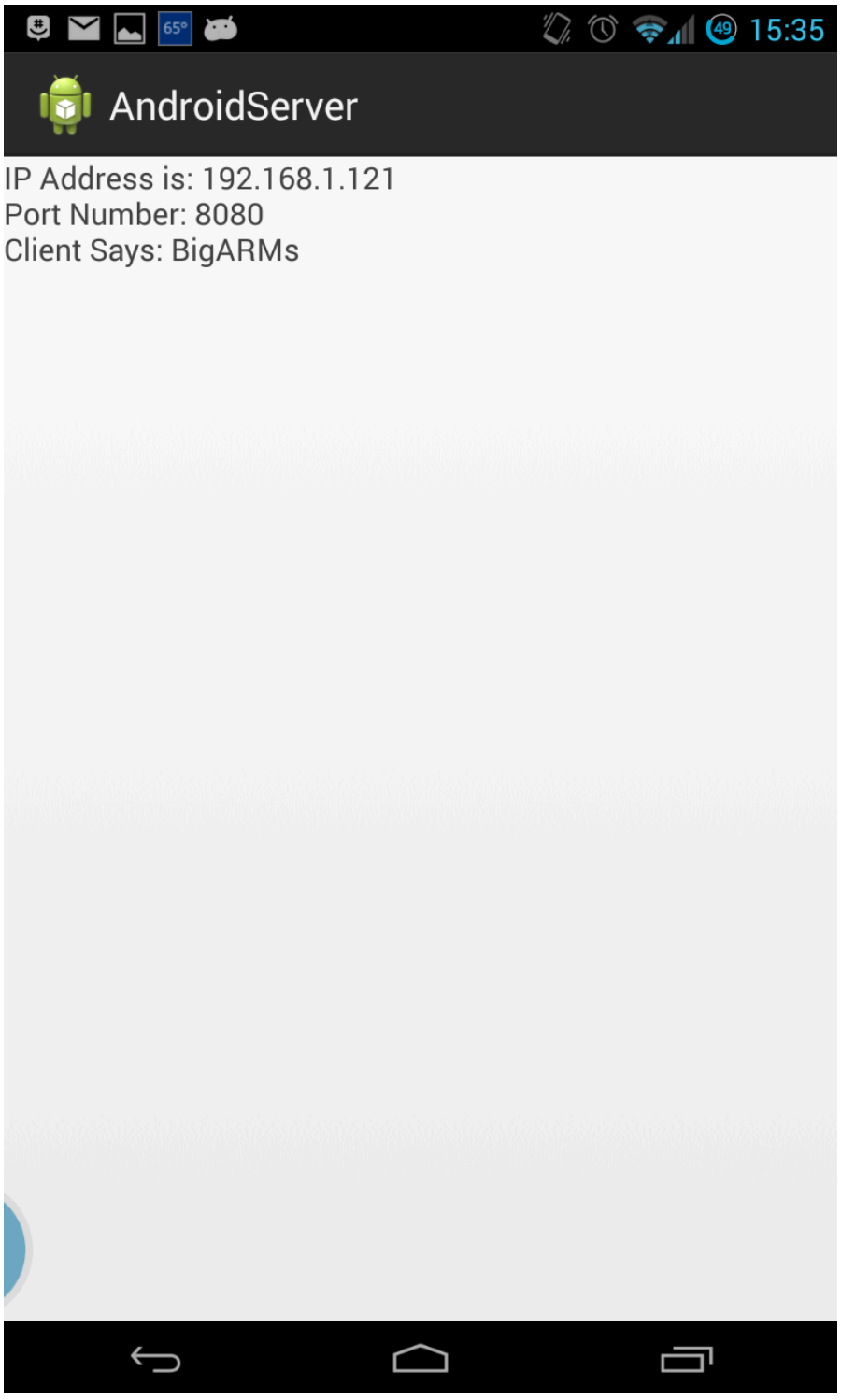

Figure 4.2 – Android Server Application

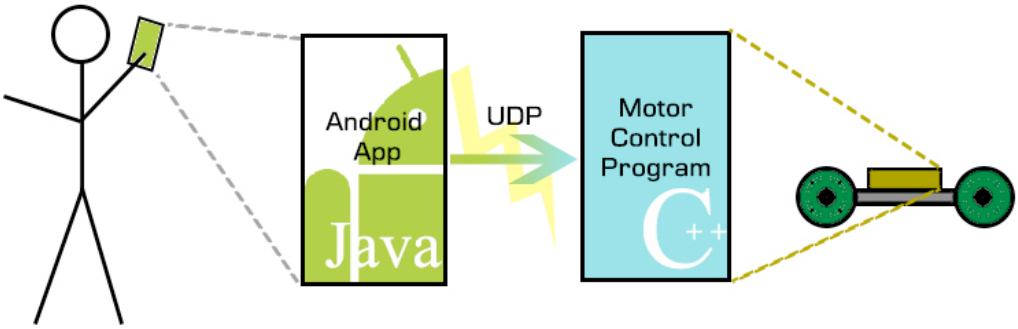

Figure 5.1 – Diagram of Android to VEX Communication

Figure 5.2 – Android Client GUI

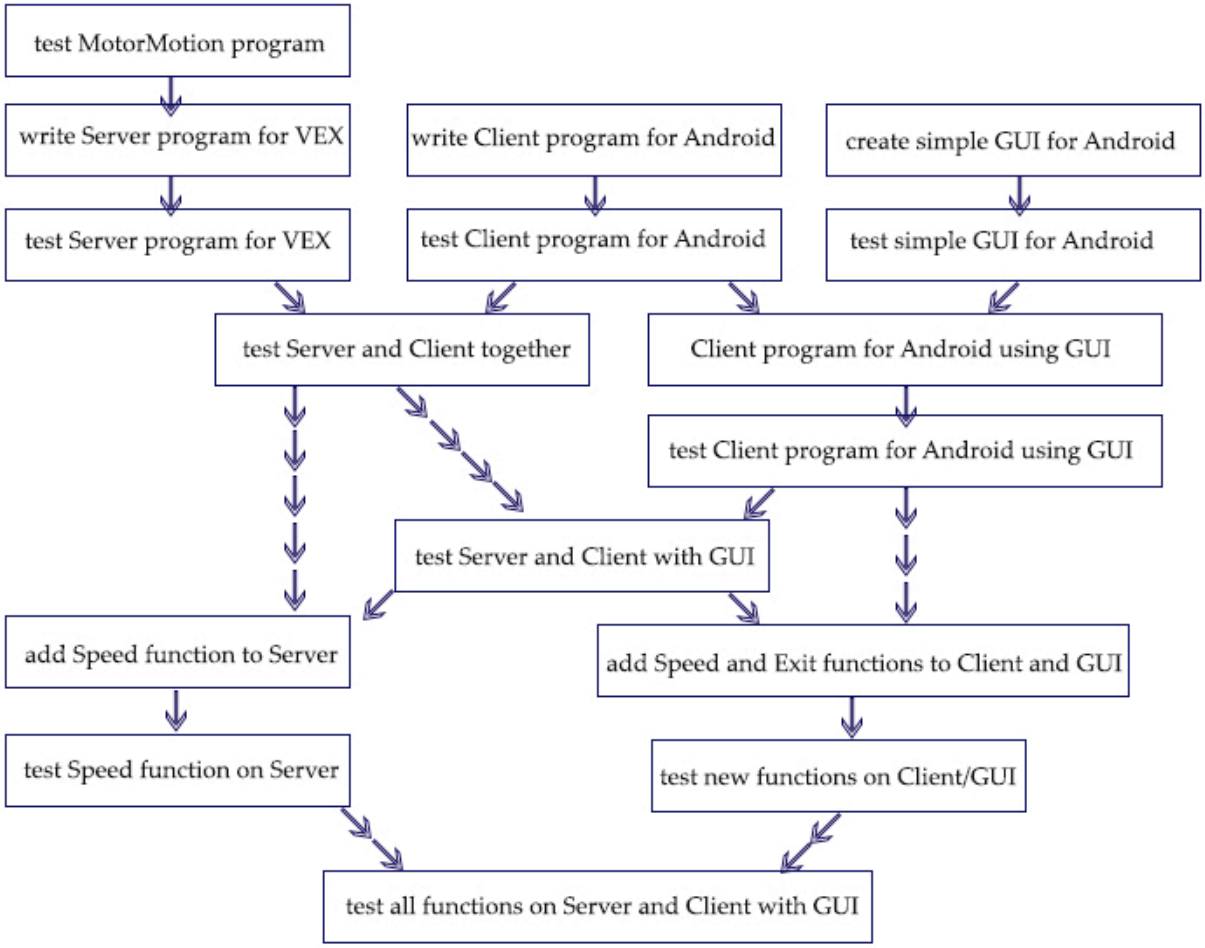

Figure 6.1 – Modular development and testing of Server-Client-GUI functions

Who	What	When	%Complete
All	Ensure all devices are secured in storage	ongoing	0.00%
Sei Jung	Android Tablet GUI	10/16/2013	100.00%
Sei Jung	Design new UI on paper to accommodate all new requirements in addition to existing	10/4/2013	100.00%
Sei Jung	Create basic UI structure and intgrate with existing client from Android-Android	10/7/2013	100.00%
Sei Jung	Port existing movement GUI to new structure	10/11/2013	100.00%
Sei Jung	Learn format for GPS coordinates on Android to be used by GUI	10/14/2013	100.00%
Sei Jung	Create GPS (Waypoint entery and come home) part of GUI	10/16/2013	100.00%
Jacob	Communication between Vex, Phone, and	10/11/2013	100.00%
Jacob	Read information on using an Android device as both a send and receive server	10/3/2013	100.00%
Jacob	Code send/receive server on single Android	10/4/2013	100.00%
Jacob	Test send/receive server as receiver from second Android device	10/7/2013	100.00%
Jacob	Code printing and forwarding of movement commands, and printing of GPS commands, printing of errors for non-recognized	10/7/2013	100.00%
Jacob	Test forwarding of movement commands from Tablet to the Vex	10/11/2013	100.00%
Jerrell	Android able to get GPS coordinates	10/16/2013	100.00%
Jerrell	Read documentation on Android location API and location services API from Google Play	10/3/2013	100.00%
Jerrell	Create collection of 5 known GPS locations on campus using Google Maps	10/4/2013	100.00%

Figure 7.1 – Example of how we managed our tasks for this iteration

```
double dLon = endLon - startLon;
double dPhi =
Math.log(Math.tan(endLat/2.0+Math.PI/4.0)/Math.tan(startLat/2.0+Math.PI/4.0));
if(Math.abs(dLon) > Math.PI) {
        if(dLon > 0.0) {
                dLon = -(2.0 * Math.PI - dLon);
        }
        else {
                dLon = (2.0 * Math.PI - dLon);
        }
}

bearing[0] = (Math.toDegrees(Math.atan2(dLon, dPhi)) + 360.0) % 360.0;
```

Algorithm 1 – Bearing Calculation

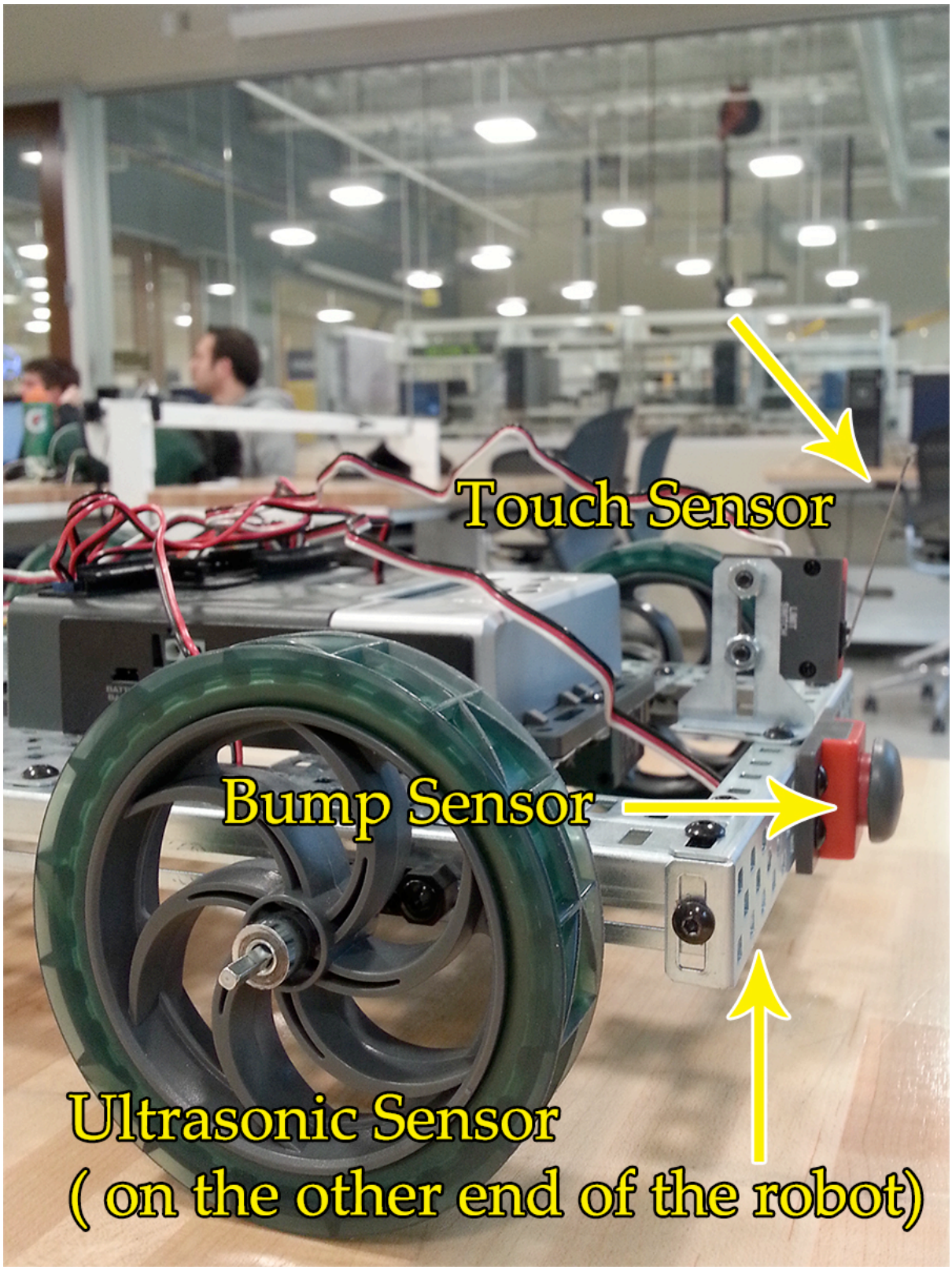

Figure 8.1 – Touch Sensors on the VEX Robot

Figure 8.2 – Android Tablet Application

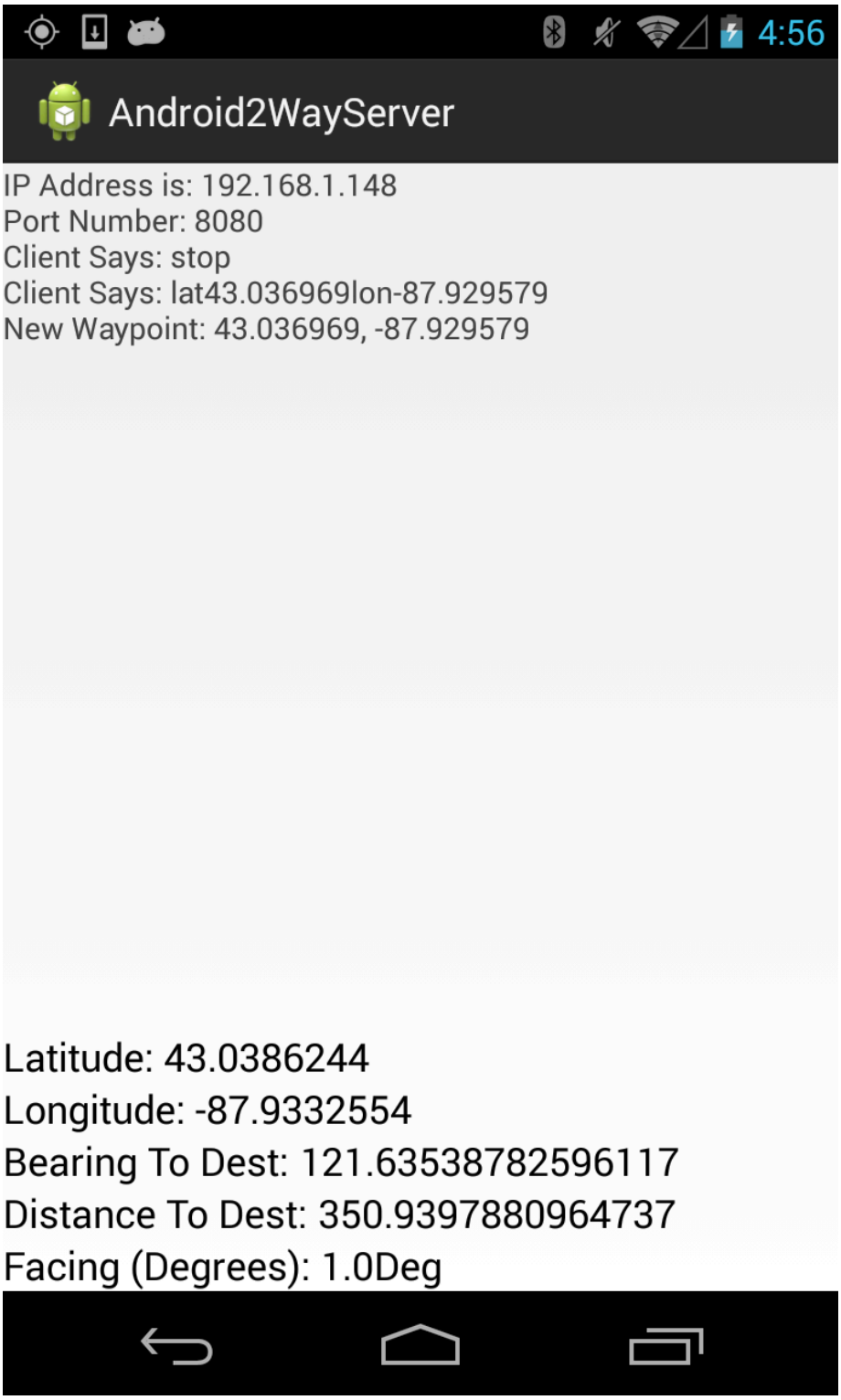

Figure 8.3 – Android Phone Application

Figure 9.1 – Android to VEX Communication

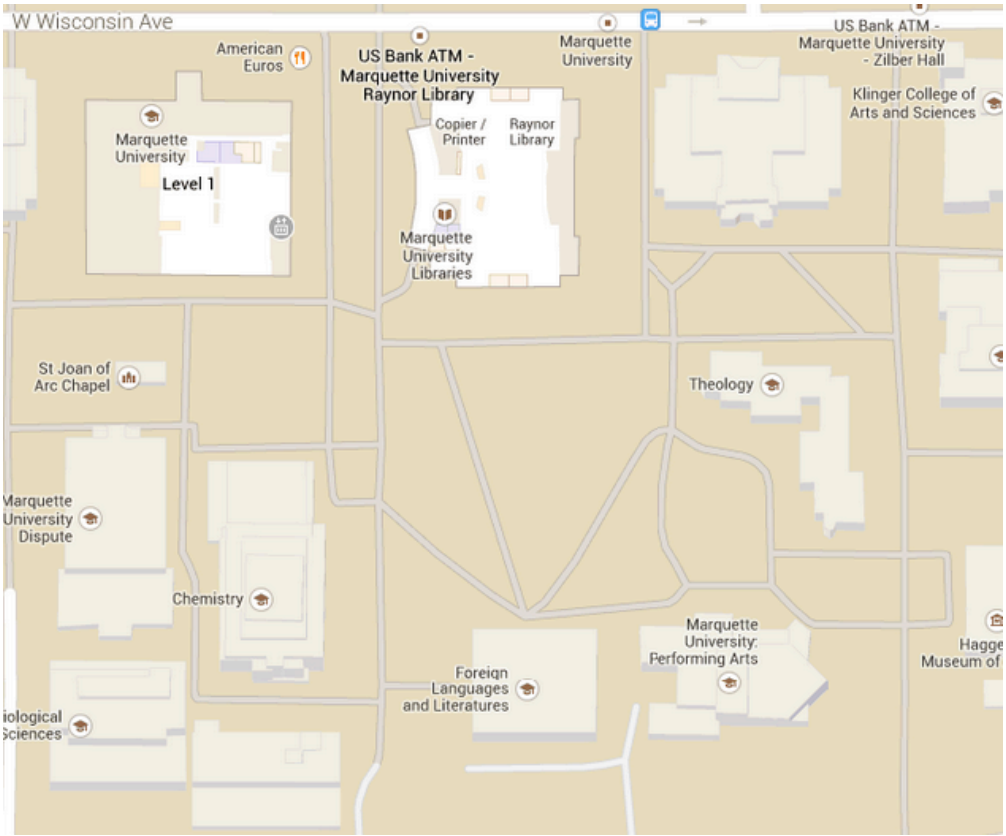

Figure 9.2 – Map of the Navigation Area