|  |
| --- |
| HAROLD's Mechanics |
| Iteration Two Report |
| Embedded Systems |
|  |
| **Jenny Reiman** |
| **Andrew Kirkham**  **Jason Laqua** |

**10/31/2013**

|  |
| --- |
|  |

Table of Contents

Table of Contents 2

Specifications 4

Android UI 4

Communication between Vex and Android Devices 4

Android can get GPS coordinates 4

Robot able to move to GPS coordinates 4

Robot move in straight line 4

Robot uses sensors 4

Cliff test 4

Design 5

Android UI 5

Communication between Vex and Android 5

Android can get GPS coordinates 5

Robot able to move to GPS coordinates 5

Robot move in straight line 5

Robot uses sensors 5

Cliff test 5

Review 6

Android UI 6

Communication between Vex and Android 6

Android can get GPS coordinates 6

Robot able to move to GPS coordinates 6

Robot move in straight line 6

Robot uses sensors 6

Cliff test 6

Test 7

Android UI 7

Communication between Vex and Android 7

Android can get GPS coordinates 7

Robot able to move to GPS coordinates 7

Robot move in straight line 7

Robot uses sensors 7

Cliff test 7

Risk Mitigation 8

Sources 9

# Specifications

### Android UI

Design a user interface for the Android table that includes remote control of the robot, a go home button, and allows the entry of waypoints.

### Communication between Vex and Android Devices

This feature allows the android tablet to send packets containing information through the phone to VEX to interpret.

### Android can get GPS coordinates

The phone should be able to receive GPS coordinates and know its own location.

### Robot able to move to GPS coordinates

Based on the coordinate received by the phone the robot should be able to drive along the correct bearing/heading.

### Robot move in straight line

The robot should be able to drive straight for 20m.

### Robot uses sensors

The robot needs to utilize three different sensors to aid in its functionality.

### Cliff test

The robot should perform a function to prevent itself from diving off a large drop in elevation.

# Design

### Android UI

The android UI has a functional layout that includes remote control buttons (forward, backward, turn left and turn right), a waypoint entry button which launches Google maps to allow waypoints to be entered, and a home button to drive back to the tablet.

### Communication between Vex and Android

The tablet sends packets containing either GPS coordinates or remote control commands to the Android phone. From there, the packets containing remote control commands are forwarded to the VEX while packets containing GPS coordinates are turned into bearing packets.

### Android can get GPS coordinates

The tablet checks to see if a GPS (preferable) or network location is known. If it is not, then it will attempt get one, if this is not possible an error message will be sent. The tablet uses Google maps to get the location of the waypoints. The phone gets its current location in the same way that the table does.

### Robot able to move to GPS coordinates

The phone will get a bearing to the waypoint, and send this value to the VEX which will drive in the direction of that bearing.

### Robot move in straight line

This was attempted by implementing an optical encoder. Although the behavior of the encoder was verified using an oscilloscope the data wasn’t readable via the VEX. Since this did not work, the straightness will be corrected by using the phones compass to keep a constant heading. Currently straightness is achieved by manually setting motors to drive at different speeds for each wheel.

### Robot uses sensors

The three sensors being used in this iteration are a bump sensor, ultrasonic sensor, and a limit sensor. The bump sensor will stop the robot if it back into any obstacles. The ultrasonic sensor uses echo location to determine the distance from an object in front of it. When an object is within three inches the robot will stop moving.

### Cliff test

The limit sensor is designed to drag along the ground at logical high, and flip to a logical low when the switch clicks due to a drop in elevation. Using interrupts, when the limit sensor enters the callback the robot will receive the stop command and not drive over the cliff.

# Review

### Android UI

The android UI functionally works the way it is intended to, and is simple for the user to figure out.

### Communication between Vex and Android

The communication is fairly consistent in sending packets, although packet overloads tend to crash the application.

### Android can get GPS coordinates

The GPS successfully sends coordinates to the phone in a comma delimited list. The phone retrieves its location from its internal GPS system.

### Robot able to move to GPS coordinates

The design has not been tested to work yet.

### Robot move in straight line

Theoretically the new design plan should would to correct the straightness of the robot, but it has not been implemented or tested yet. The manual fix works for a few meters.

### Robot uses sensors

The bump sensor, ultrasonic sensor and limit sensor all work as expected. All of these sensors will stop the robot to prevent damage to itself. The ultrasonic sensor detects objects at a range of three inches so it does not stop at false obstacles that it might not end up running into.

### Cliff test

The cliff test was successful and the robot stops in time to prevent it from moving over the cliff. Potential problems with this implementation may involve the limit sensor going off due to sidewalk cracks or being dragged and bent.

# Test

### Android UI

This was tested by running the java application on the Android devices and making sure that it appeared the way we intended it to. Buttons were later testing to make sure that the robot performed the correct functionality.

### Communication between Vex and Android

We ensured that the different types of packets were received and correctly interpreted on each of the different devices.

### Android can get GPS coordinates

The coordinates were confirmed to be appearing in the packets being sent.

### Robot able to move to GPS coordinates

This feature is currently untested.

### Robot move in straight line

The straightness at which the robot drives was determined by lining the wheels up with a line in the floor and measuring how long it was able to continue along a straight path. The initial straightness without correction was measured to be around 2ft. The optical encoded was tested to produce the correct output with an oscilloscope, but the results were not able to be replicated with the VEX. The manual method has shown the robot to move straight for a few meters.

### Robot uses sensors

The bump sensor, ultrasonic sensor were tested by driving the robot towards the wall to see if it stopped. The limit sensor was tested by trying to drive the robot off the table.

### Cliff test

This was tested by driving the robot off of a table and having someone waiting to catch it in case the limit sensor code did not work.

# Risk Mitigation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Estimated Time To**  **Completion (hours)** | **Actual Hours to Completion** | **Points** | **Risk** | **Final Value (points/hour)\*Risk** |
| Android UI | 1 | 1 | 20 | 1.0 | 20 |
| Communication Vex to Android | 4 | 1 | 20 | 0.7 | 3.5 |
| Android can get GPS coordinates | 10 | 15 | 15 | 0.5 | 0.75 |
| Robot able to move to GPS coordinate | 10 | 6+ | 10 | 0.2 | 0.2 |
| Robot moves in straight line | 5 | 10+ | 10 | 0.4 | 0.8 |
| Robot uses sensors | 10 | 20 | 10 | 0.6 | 0.6 |
| Cliff test | 2 | 2 | 15 | 0.5 | 3.75 |
| Totals: | 42 |  | 100 | N/A | N/A |

Note: A plus in the about Actual Hours to Completion column indicates that the task is still being worked on.

The low risk associated with the Communication and the Android UI was correctly predicted as we had few issues in completing these tasks. It took us less time to complete the Communication than we originally expected. Using sensors and moving the robot in a straight line consequently incurred a much higher risk than we predicted after spending several work sessions trying to figure out the optical encoder. The optical encoder solution, which would have counted for both a sensor and been the preferred method for driving straight, never worked and ended up delaying these tasks. We spent several more hours on them than predicted. The ease of use of the limit sensor helped to compensate for the amount of work done for the sensor task. This also made the cliff test fairly easy to do and we probably could have assigned it a lower risk. Incorporating the ultrasonic sensor was proven to be an easier task as well, but we struggled with reading it while we had the VEX USB powered. We later discovered that the output ports don’t work with USB power and the ultrasonic sensor was resolved. Having the Android receive GPS coordinates was given an appropriate risk and only took a few more hours to complete than estimated. For the robot moving to the GPS coordinate a higher risk was needed because the heading is not consistent or accurate.

# Sources

1. <http://www.abc.se/~m6695/udp.html> (client/server tutorial code)
2. <http://content.vexrobotics.com/docs/VEXnet_Cortex_UserGuide_081811.pdf>
3. <http://developer.android.com/reference/android/net/wifi/p2p/package-summary.html>
4. <http://developer.android.com/training/connect-devices-wirelessly/wifi-direct.html>
5. <http://developer.android.com/reference/android/widget/Button.html>
6. <http://developer.android.com/training/basics/firstapp/starting-activity.html>
7. <http://www.helloandroid.com/tutorials/simple-udp-communication-example>
8. <http://en.wikipedia.org/wiki/Wi-Fi_Direct>
9. <http://www.education.rec.ri.cmu.edu/products/teaching_robotc_cortex/reference/VEXnet_setup_guide.pdf>
10. <http://www.vexforum.com/wiki/index.php/VEXpro_Programming_Resources>
11. <http://www.vexforum.com/wiki/Bumper_Switch>
12. <http://www.vexrobotics.com/276-2174.html>
13. <http://en.wikipedia.org/wiki/Rotary_encoder>
14. <https://www.cresis.ku.edu/~cgifford/eecs690/Vex_Robotics_Tutorial.pdf>
15. <http://stackoverflow.com/questions/14986146/update-android-ui-from-a-thread-in-another-class>
16. <https://developers.google.com/maps/documentation/android/start>
17. <https://developers.google.com/maps/documentation/android/map>
18. <https://code.google.com/p/terkos/source/browse/trunk/src/examples/windows/SonarRover/encoder.h?spec=svn1217&r=1217>
19. <http://developer.android.com/guide/topics/location/index.html>
20. <http://stackoverflow.com/questions/4308262/calculate-compass-bearing-heading-to-location-in-android>
21. <http://stackoverflow.com/questions/14085965/start-more-than-one-async-task-on-android>
22. <http://www.vexforum.com/showthread.php?t=60766>

Note: Sources are specific to the second iteration staring with source 12.