ELEC3542 IoT Project

Roving Sprinkler System

Final Report

Yu Kuai David 2013507514

Link to GitHub Repo:   
https://github.com/MrDavidYu/Roving-Sprinkler-System.git

# Introduction

Often the most annoying and time-consuming aspect of botany is the incessant need for grass and flowers to be watered regularly. Although sprinkler systems exist to address this issue, the problem of plant watering has never been solved in its entirety; serious issues persist in modern-day solutions such as the lack of response to changes in rainfall, humidity and temperature as well as the messy trail of water pipes and spouts left behind by automated sprinkler systems.

The Roving Sprinkler System (RSS) is an attempt to tackle such problems in one go by mounting various Raspberry Pi devices onto all-terrain water-carrying rovers which sweep across the field to water arid areas in a manner akin to the Roomba robots of today. The RSS is infinitely scalable and additional rovers can be easily added to the existing fleet. Each individual rover will record data such as temperature, humidity, area covered and water usage and transmit them to a central router bot which processes the feed to be uploaded to the cloud. This feature allows for the status of the field to be monitored remotely via an Android app.

# Finalized System Architecture

**Router bot:** Every RSS fleet must contain one rover which acts as a router for controlling the behavior of other rovers and for receiving live feeds from the other rovers for processing. This usually involves calculating an average from all the data received. Every five seconds, the router bot would also upload the amalgamated data to the ThingSpeak cloud. The router bot could also perform sprinkling activities, but is unable to take in temperature / humidity readings owing to its lack of a SenseHat.

**Rover:** A rover can be easily integrated into the RSS fleet by syncing it with the existing router bot. The rover is an RC vehicle which contains an open water tank on the back for easy refill and the Raspberry Pi unit at the bottom (close to the ground) so the sensors can pick up the temperature and humidity levels of the field. Water dispenses from the Rover via a sprinkler valve. All movement activities will be initiated and recorded by the Raspberry Pi. The exact location of the rover relative to the rover refill dock will be stored as a combination of the time spent in motion, the direction of movement and known speed of the rover.

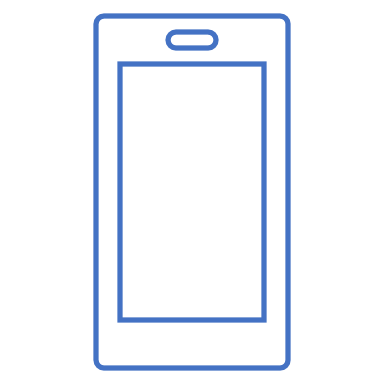
**Rover Refill Dock:** An automated refill platform at the edge of the watering area which allows for the router bot to be refilled. Magnets are placed at the bottom of the platform so the rover will be able to precisely pinpoint the location of the dock using the Sense HAT’s magnetometer. Once a rover is parked securely on the platform, it will send out a signal for the dock to begin refilling. The router bot will monitor the rover refill queue to ensure that only one rover approaches the dock at one time.

**Cloud:** All data collected will be transmitted by the router bot to the cloud via the user’s home wifi. This will allow the user to remotely monitor the RSS’s activity and behavior via the web or an Android device.

**Android App:** The Android application performs various monitoring functions. It is primarily responsible for parsing information obtained from the ThingSpeak cloud via a ThingSpeak JSON API. Every field on the cloud is pulled and displayed as an individual graph for the user to easily interpret the data. Such fields include average temperature, humidity, current power consumption and current water consumption.

Rover Refill Dock

Cloud  
Storage



Remote observation + control

Router bot

# Work Accomplished and Performance Analysis

For the purpose of the demonstration, each rover is simulated by a RaspberryPi, where the role of the router bot is simulated by a Pi configured as a wifi router for the others. This Pi is connected to the internet via an Ethernet cable. An ordinary rover is simulated by a stationary RaspberyPi with a SenseHat which outputs various information local to that Pi such as power/water levels, whether the rover is currently transmitting/receiving information, whether the rover is inactive owing to a power or water shortage as well as any commands to REFILL or SPRINKLE received from the router bot.

The following files/executables are created for this project:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Function** | **Router** | **Rover** |
| data\_received.py | Lights up an indicator light on SenseHat whenever a message is received from the router Pi. |  | ✔ |
| data\_transmitted.py | Lights up an indicator light on SenseHat whenever a message is sent to the router Pi. |  | ✔ |
| EnqueueDevice.java | Called whenever either water or power is depleted on the rover. Sends an enqueue request message to the router. Once the router receives this message, the rover’s IP would be placed into the refill queue. |  | ✔ |
| DequeueDevice.java | Called whenever both water and power are refilled on the rover. Sends a dequeue request message to the router. Once the router receives this message, the rover’s IP would be removed from the refill queue. |  | ✔ |
| detect\_power.py | Simulates the detection of changes in power/battery levels of the Pi. This script is run whenever a status reading is required. Currently this script simply returns the power level as displayed on the SenseHat. | ✔ | ✔ |
| detect\_water.py | Same as detect\_power.py but for the water level as displayed on the SenseHat. | ✔ | ✔ |
| joystick.py | Activates the joystick on the SenseHat, allowing the user to move the joystick NORTH, EAST, SOUTH and WEST to simulate water gain, power loss, water loss and power gain respectively. This script also enables the two white/blue bars on the right of the SenseHat matrix to display the current power/water levels. Additionally, when either power or water becomes depleted, SenseHat will display an “E” for empty on the matrix. Similarly, joystick.py is also responsible for removing the “E” symbol whenever both meters are filled. |  | ✔ |
| refill.py | Triggered whenever a command is received from the router for the rover to begin refilling at the dock. This lights up the REFILL indicator lights (see diagram below). |  | ✔ |
| sprinkle.py | Triggered whenever a command is received from the router for the rover to begin sprinkling. This lights up the SPRINKLE indicator lights (see diagram below). |  | ✔ |
| RSS\_thingspeak\_router.py | Used by the router Pi to upload data to the ThingSpeak server every 5 seconds. | ✔ |  |
| sense\_hat\_humid\_star.py | Returns the current humidity reading from SenseHat (router can have this function as long as it has a SenseHat) | ✔ | ✔ |
| sense\_hat\_temp\_star.py | Returns the current temperature reading from SenseHat (router can have this function as long as it has a SenseHat) | ✔ | ✔ |
| StarDevice.java | Main executable for rover Pi’s. Performs readings of temperature/humidity/power/water at 5 second intervals and sends the data to the router. In a separate thread, a listener is maintained for socket connections from the router, which would either tell the rover to stop to refill or resume sprinkling. |  | ✔ |
| StarMain.java | Main executable for router Pi. Pulls data from all connected rover Pi’s every 5 seconds and updates the ThingSpeak server. In a separate thread it listens for rover requests to enqueue/dequeue into the refill queue. If there is a rover at the top of the queue, then a REFILL command is sent to that rover. Once the FULL status is received from the rover, it is removed from the top of the queue. | ✔ |  |

The following illustration shows all the possible outputs from the SenseHat matrix and what each output represents:

Water level status bar

Power level status bar

Power/water depleted, stop watering status

Begin sprinkle command

Begin refill command

Data transmitted indicator

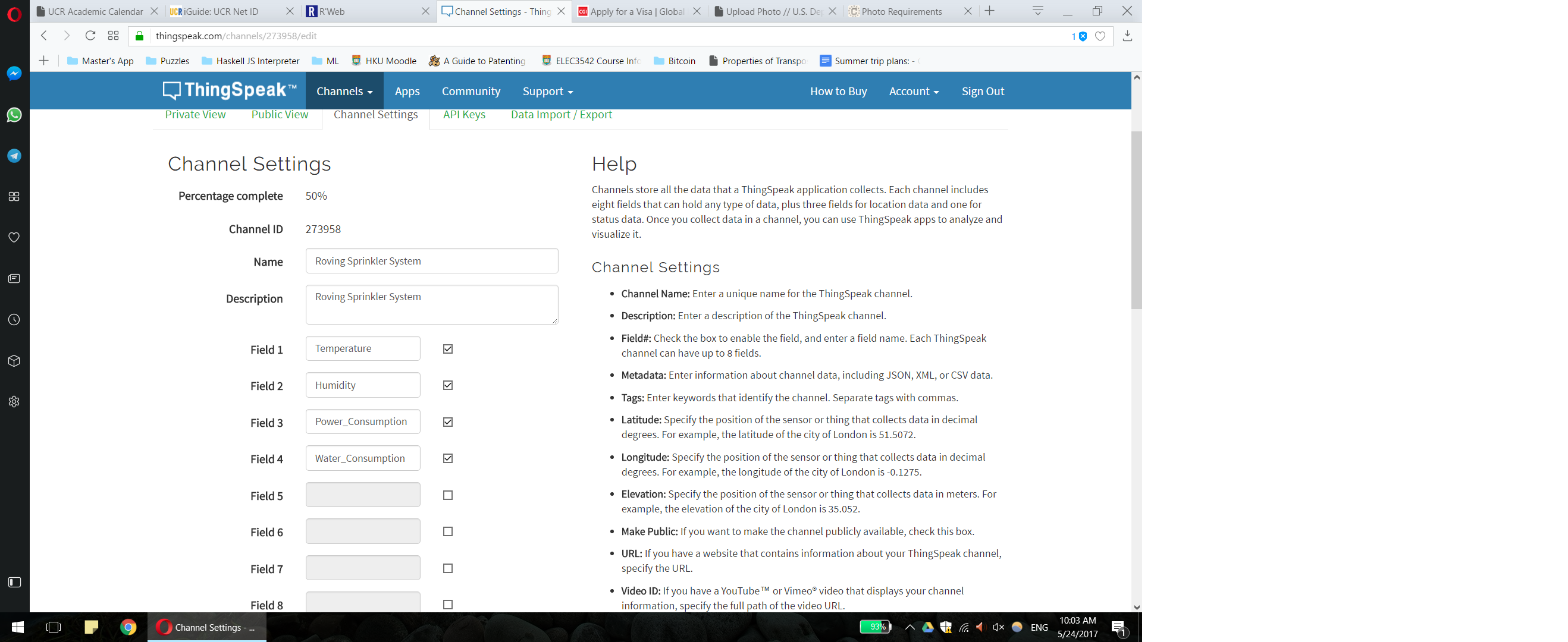
Data received indicator

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

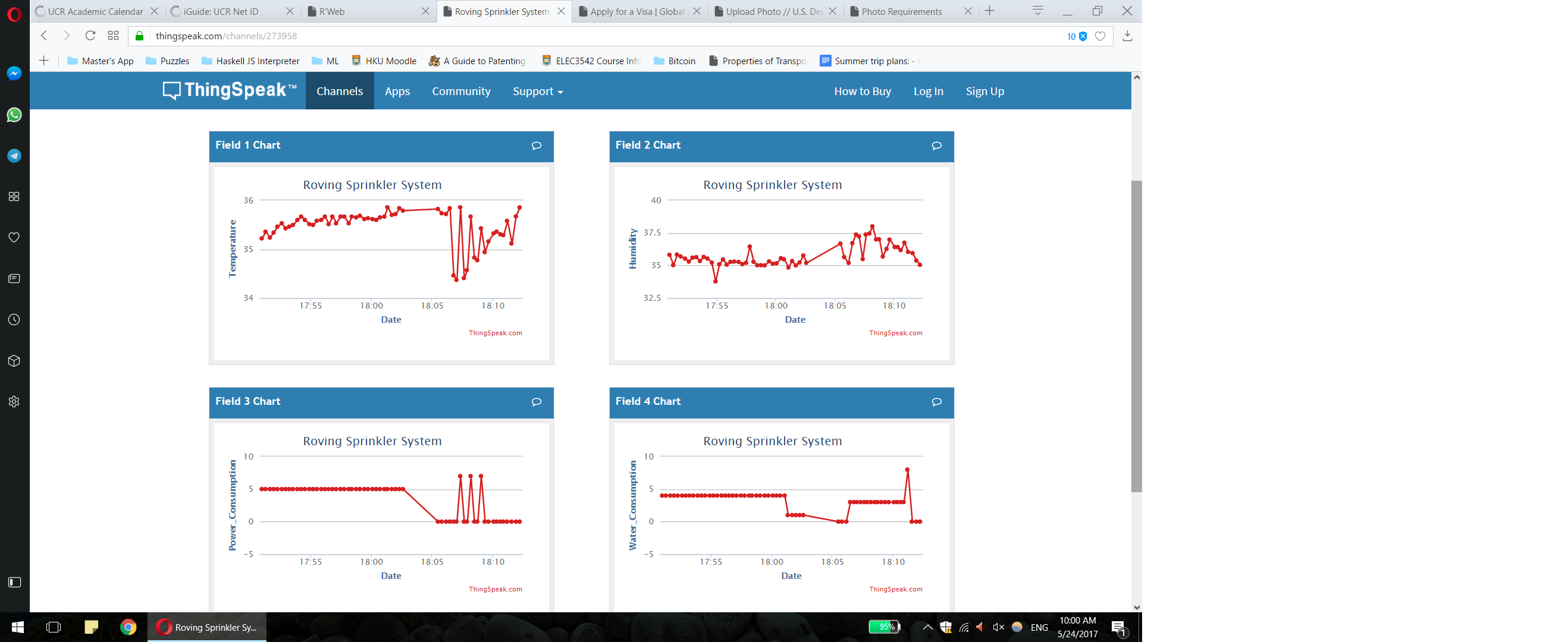
# Implementation REcord & Details

|  |  |
| --- | --- |
| **Picture** | **Description** |
| **C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153052.jpg** | Rover bot, upon first running StarDevice.class.  The joystick.py script is triggered, and the power/water level status bars light up. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153055.jpg | Every five seconds, the rover sends out a temperature + humidity + power usage + water usage reading to the router. This transmission is accompanied by the data transmitted indicator light. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153107.jpg | Upon pulling the toggle WEST and SOUTH, the user may reduce the water and power levels respectively. This change is reflected by the SenseHat’s power and water meters. Currently, 2 units of power and 4 units of water have been consumed. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153132.jpg | 5 units of power and 6 units of water have been consumed. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153136.jpg | Once either power or water is depleted, the yellow “E” symbol shows up indicating that the rover is empty. The transmission indicator lights up to indicate that a enqueue request has been transmitted to the router. In the meantime, the rover would stop all sprinkling activities. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153145.jpg | After around 1-2 seconds, a reply is transmitted by the router and received by the rover. The orange bar is a REFILL command by the router, meaning that the rover is now able to move to the refill station. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153202.jpg | If there is another device that has run empty, it would not receive a REFILL command from the server until the rover at the top of the refill queue has been dequeued. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153222.jpg | When the first rover has finished refilling both power and water, the “E” symbol disappears, and an dequeue request is sent to the router. Once the router receives this request and sees that the request is sent by the bot with a matching IP address to the bot at the top of the refill queue, it replies with a SPRINKLE command and dequeues that IP. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153223.jpg | When the SPRINKLE command is received, the data received indicator lights up, and the orange bar is replaced by a green bar, indicating that the router is now able to go back and sprinkle the lawn again. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153225.jpg | The green bar remains lit for 3 seconds. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153232.jpg | Now the second device receives the REFILL command. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153241.jpg | When the second device finishes refilling, the “E” would also disappear and a dequeue request sent out. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153242.jpg | Once approved, the orange REFILL bar is replaced by the green SPRINKLE bar, along with an orange indicator for data received from the router. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170522_153243.jpg | The green bar remains lit for 3 seconds. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170524_095925.jpg | This is an image of the router bot. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170524_100103.jpg | During initialization, the router is started first by the command *java StarMain* in the directory containing all the relevant files.  Rovers are then started by the command *java StarDevice* in the directory containing all the relevant files. No further steps are necessary, but in order for the joystick to work, the user must ensure that the SenseHat window that pops up following this command is in focus. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170524_100427.jpg | A screenshot of the rover device interface: Every five seconds, a reading of temperature/humidity/power/water is displayed on the command line. |
| C:\Users\David Yu\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG_20170524_100450.jpg | A screen shot of the router device interface: Every five seconds, a reading of the combined temperature /humidity/power/water is displayed on the command line. |

# Thingspeak server configuration & Output



This figure illustrates the Channel Settings for the Roving Sprinkler System. There are 4 fields configured to receive data from the router Pi.

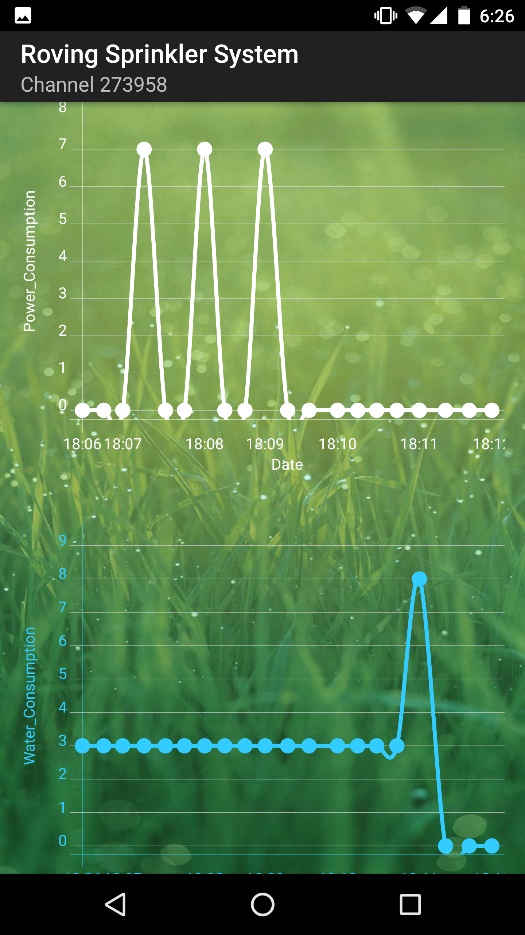
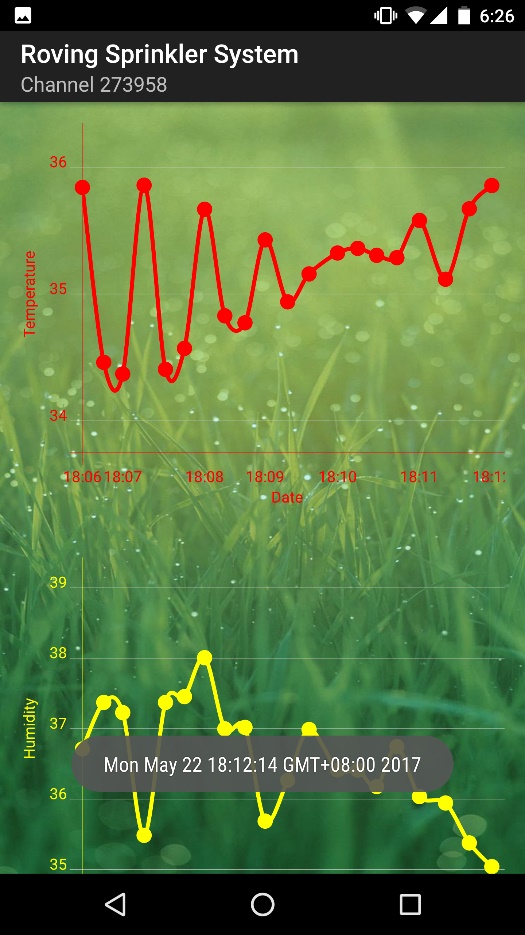


This figure shows the live-feed status of all four channels. For some of these channels, the fit line shows very abrupt movements. This is not a technical glitch, but rather a feature that shows the readings of different devices. Owing to a lack of data for individual rover outputs, all data amalgamated by the router Pi is displayed on the same graph so users can clearly spot any discrepancy in readings from all connected devices.

# Thingspeak APP configuration & Output

The accompanying android application for this project is called Sprinkle!, and it utilizes Macro Yau’s ThingSpeak API to pull and display data from the ThingSpeak cloud on an Android device.

The application interface is shown below:



**Figure 1.** The application is circled.

**Figure 2.** The application interface displaying temperature and humidity levels with 20 data points.

**Figure 3.** The application interface displaying power and water levels with 20 data points.

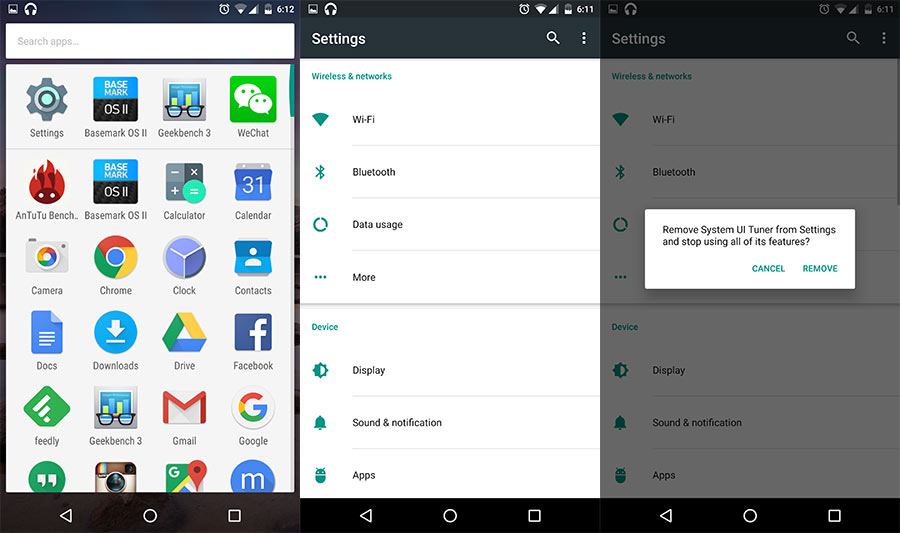
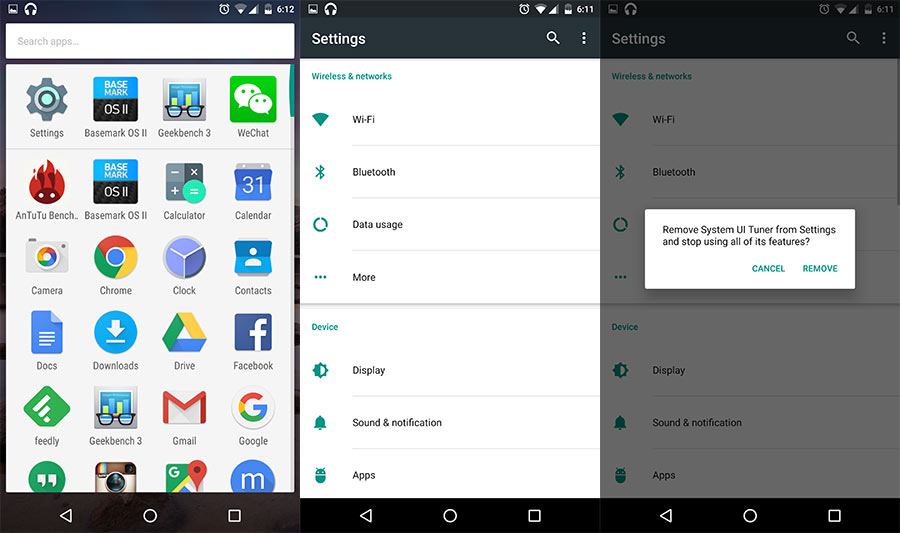
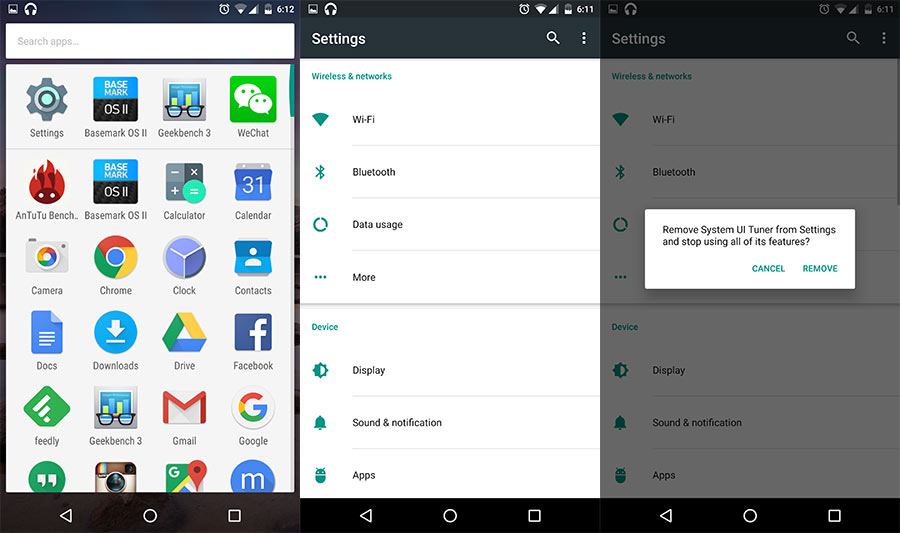
**Figure 4.** The application does not update in real time. However, it can be refreshed by a swipe gesture.

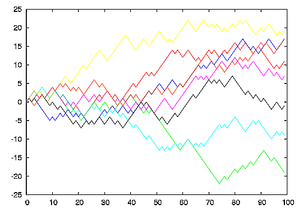
# Challenges in implementation

Several challenges were encountered during implementation of this project. First of all, there is the issue that it is infeasible to buy actual RC cars on which to mount the Raspberry Pi devices. In order to reduce cost, stationary Pi devices were used, and the joystick on SenseHat was used to simulate the gradual decrease of water and power levels over time. Additionally, problems of connectivity was encountered when trying to make the Android application. The initial plan was to create a private channel on ThingSpeak since security is a concern for users of the project. However in the end, I opted for a public channel instead since it was easier to set up and the public channel had already been tested. Another issue which was encountered was that since the router bot was set up as a wifi router, it had to be physically connected to an ethernet cable. It was not possible for the route to simultaneously connect to a wifi network AND act as a wifi provider. This problem may be solved by a change in the project architecture where the router bot acts as a stationary hotspot and does not actually perform any rover duties. The downside is that this may increase the cost of the project.

# Possible future development

The Android application may be expanded to include all the following features for better ease of use:





Rover (4)

Rover (3)

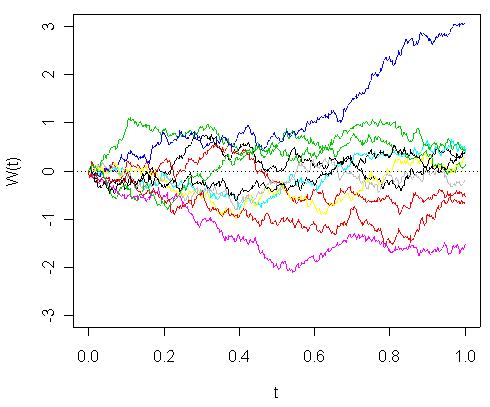
Dock

Router (1)

Real Time Data

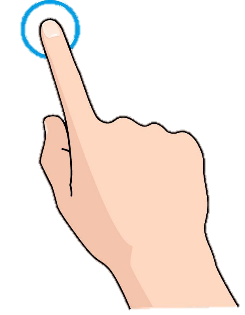
Preferences

Rover (2)



Set Patrol Area

Set Thresholds



Add New Bot

Select Router

Rover ( )

(fig 1.) (fig. 2) (fig. 3)

(Figure 1): The starting interface of the Android application on the user’s end. Includes the following settings (at minimum): *Preferences*: User preferences, including ThingSpeak account details, channel IDs etc.  
*Real Time Data*: Real time ThingSpeak data pulled from the cloud via ThingSpeak API. Includes temperature, humidity, moisture, battery level and water level data from every device as well as their averages. (Figure 2)  
*Set Thresholds*: Allows users to present temperature, humidity and moisture threshold levels for individual rovers or the RSS fleet as a whole such that whenever a rover detects levels beyond the threshold its dispenser system will activate.  
*Set Patrol Area*: For the sake of simplicity, the RSS app will regard the entire field as a rectangle, with exact dimensions defined by the user. The fields will then be divided automatically into sections depending on the number of rovers in the fleet. This ensures that rovers do not collide during patrol. Users will have the freedom to adjust the sizes of different patrol areas and to determine the placement of individual rovers and the dock. (Figure 3)  
*Select Router*: Users can use this functionality to determine which rover acts as the router device.  
*Add New Bot*: Whenever a new rover is introduced to the fleet, the user will need to manually connect it to the router.

# References

Macro Yau’s ThingSpeak Android API: https://github.com/MacroYau/ThingSpeakAndroid