Commuter Tracking Sensor Network

"A wireless sensor net for outdoor tracking and localization."

Project Principles:

Seth Hendrick - srh7240@rit.edu
Alessandro Sarra - ags7798@rit.edu
Jared Mistretta - jxm6666@rit.edu

Project Collaborators:

Dr. Jeffrey Wagner - mjwgse@rit.edu

Final Report Date:

December 11, 2014

Table of Contents

Overview

Needs - Quantifying a dollar value for a public good or a public trail, like any public land, can be a difficult task. As the product is being paid for via third party channels, (e.g., state and local taxes), the utility gained by the actual user (who may have not paid any taxes for the good) of the trail cannot be directly correlated with the taxes paid. The actual usage of the trail needs to be tracked to gain a better understanding of the trail value in terms of usage.

Objective - The objective of the project is to capture the usage data for the Lehigh Valley Trail. These data include the mode of transportation and the entry and exit points commuters use. The data will then be accessible via a web interface to a limited set of users.

Description - A wireless sensor network that utilizes computer vision and mesh networking will track the usage of the Lehigh Valley trail. The network is comprised of a series of modules that use image sensors to recognize commuters and their modes of transportation. Additionally these modules communicate with one another to determine what are the entry and exit points commuters are using to access the trail. The network has a gateway node that allows the data to be backhauled to the internet. The data is stored via cloud storage and accessible via a web interface.

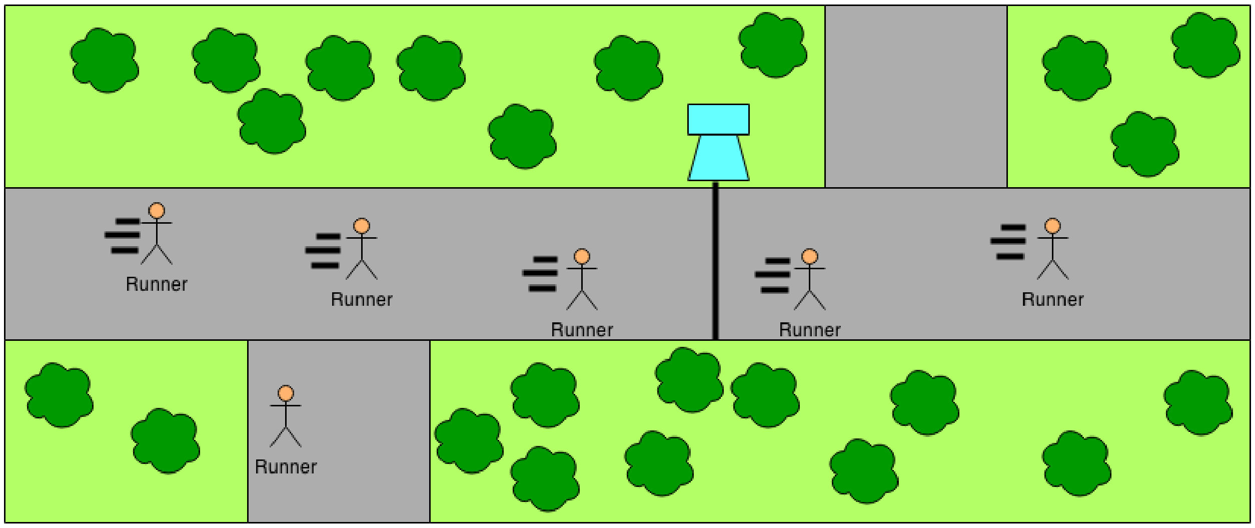


Figure 1 - Marketing Diagram

Figure 1 is a marketing diagram that targets the average user. It shows several subjects passing in front of the blue sensor node located on the Lehigh Valley trail. The sensor node is shown detecting the size of the object passing in front of it, as well as the direction the runner is travelling. The node is also located near two trail entrances, which make it possible to estimate which access points are used to enter the trail.

Requirements Specs -

Customer Needs (Marketing Requirements) -

1. Modules will detect commuters and the path they are taking.
2. Modules will determine the mode of transportation used by the commuter.
3. Modules will be deployed for a week of time without need for maintenance.
4. Trail modules will be able to intercommunicate via a network.
5. Data gathered by the network will be stored via a cloud solution.
6. Data will be accessible via a website interface to a limited set of users, including stakeholders and developers.
7. The website will display the status of each trail node, and provide basic control of the nodes.

Engineering Specs -

Table 1 - Engineering Specifications

Spec #	Marketing Reqs.	Engineering Specification	Justification
1	1,2	The image sensor must be able to capture the image.	The image is needed so that CV can be performed.
2	1,2	The image sensor must be able to perform in a variety of light intensities and directions including overcast weather, dusk and dawn visibility, and midday brightness levels.	The image sensor will be outside and exposed to various weather conditions and light levels throughout the day.
3	3	The image sensor must draw a small amount of power relative to other image sensors on the market, which is 50 µA in standby and 110 mW in active mode on average.	The image sensor is running on a battery, and must draw a minimal amount of current to maintain charge for prolonged periods.
4	1,2	The lens responsible for image acquisition must have a field of view that creates a consistent image size across the various positionings that the modules will have relative to the image subjects.	The image subjects will vary in size and must produce the same size image for increased consistency and accuracy in CV algorithm output.

5	1,2	The processor must be able to acquire images from the sensor and perform CV algorithms on the acquired images.	The commuter’s mode of transportation must be determined.
6	3	The processor must draw a relatively small amount of power, which is 140mA on average while performing CV algorithms.	The processor is running on a battery, and must draw a small amount of current to maintain battery charge for prolonged periods.
7	4	The trail nodes must communicate on a band that is allowable by the FCC.	The project must adhere to IEEE ethical standards, and therefore must be legal in all regards.
8	4	The trail nodes must have a range of at least 3 miles.	The greatest distance between any two nodes is approximately 1.75 miles assuming line of sight is unobstructed.
9	5,6	A module must provide gateway services that will link the network to a database service.	The data from the network must be transferred to a database, so it can be accessed through the Internet.
10	5,6	The cloud storage solution must have enough memory to hold all of the data set that is collected.	The data acquired must not be lost due to insufficient space.
11	6	The data from the database must be accessible via a standard web browser for both mobile and desktop devices.	The data must be easily accessible.
12	6	The data from the database must be viewable in a variety of graphical representations.	The data must be easily readable.
13	6	Web site access must be limited to a select user base.	The data should not be viewed by the public due to privacy concerns.
14	3	The trail nodes should enter a low-power state in the evening.	The image sensors will not be able to capture images in the

			dark, and all power-saving options must be considered.
15	3	The trail nodes must be able to operate in the weather conditions of Rochester, NY.	Replacement of nodes would be costly, and loss of data would also result.
16	3	The modules should be hidden from view when deployed in the field.	Strategically placing the modules will deter theft.
17	3	The trail modules' communication method should draw approximately 120mA while active and 30mA while on standby.	The trail modules are battery operated, and must draw a minimal amount of current in order to maintain charge for prolonged periods.
18	3	A sustainable energy source shall be provided to each trail module.	The module's battery source is recharged, and will last longer.
19	4	Modules will use a wireless mesh protocol.	A mesh protocol provides the needed flexibility.
20	6	The server that hosts the data must be secure to prevent any unauthorized access.	The server's data must be private, and can not be tampered with from outside forces
21	4	Mesh protocol must have encryption services.	Needed to maintain security of data.
22	6	Server must have an uptime of 95%. This means that protections from DDOS and similar attacks must be built in.	With no server, the data is not accessible to stakeholders.
23	6	There should be a "Status" page hosted somewhere other than where the main server is to notify the user if the server is down, or if maintenance is going on.	Users (and developers) should be kept in the loop if the service is down, and why.
24	4	The network will also be configured so that nodes have a common operating picture.	Information will be made redundant and synchronization of data will be maintained

			between all nodes and the server.
25	3	The sensor nodes should not provide an environmental threat of any nature.	It is plausible for the battery unit within the sensor nodes to plate, potentially causing a hazardous situation. The enclosure should prevent an internal hazard from escaping to immediate surroundings.
26	7	The website user interface should allow an admin user to power-cycle a node.	Performing a manual power-cycle on a node is time consuming, as the user must travel to the trail, walk to the node and reset it.
27	7	The admins should be alerted through the web interface or email if a node requires maintenance.	If notifications of the node’s status are automatic, it prevents maintainers from walking out to the trail every day and ensuring all the nodes still work.

Concept Selection

While preliminary research has shown that there are several systems with wireless sensor nets that use video for tracking and localization, one all-inclusive solution that performs this task outdoors has yet to be found. Additionally none of the systems that were researched offer the breadth of services that are being proposed: on board CV, low power operation, diverse deployment environments and configurations, and a web interface for accessing data. None of the systems utilize a trickle power mechanism either. Using the windbelt for trickle charging in this sort of application is a novel use of the technology, and would serve as an identifying factor for the project, allowing it to stand out among others.

Table 2 - Concept Selection Chart

Problem	Solution concept
Detect Commuter	1. Computer Vision - Selected 2. Pressure Sensor 3. Laser Sensor
Module Networking	1. Wireless Mesh - Selected 2. Wi-Fi 3. SDR Ham
Power Charging	1. Windbelt - Selected 2. Solar 3. Turbine 4. None
Data storage and access	1. Cloud Solution 2. Custom web server - Selected 3. Local storage and access
Gateway	1. Raspberry Pi and Ethernet - Selected 2. Wi-Fi 3. 3G/4G 4. Proprietary Radio

CV was selected as the method to gather the data as it is considered the most versatile method to accomplish our tasks. Multiple systems would be required to gather the same data if other technology was used.

Wireless mesh technology was also chosen for its versatility and adaptability. Regular IP networking does not apply well to the wireless arena. By using a mesh protocol there is no need to preconfigure the network topology. It is desired that the user be able to apply the modules and technology being developing for a variety of applications, not just our specific use case.

The windbelt was chosen for its compact design and high efficiency when compared to other systems of this size. It is also considerably cheaper than using solar panels or wind turbines.

Signal Acquisition Method

Power must be provided to the MCU and connected peripherals for an extended period of time. For this reason, use of a sustainable energy resource is required to keep nodes deployed on the trail for longer periods. With the sustainable energy source, the need for manual recharge of the connected energy storage device will occur fewer times over a given period. This is ideal, since manually charging the storage devices would require traveling to each node, disconnecting the batteries, bringing them to the base station and connecting them to one charger, which would have the ability to charge only one battery at a time. If this were the case, it would require extended periods of time with the nodes not being deployed and not collecting valuable data for that time period. On all accounts, the project would be halted for that duration. Several sustainable energy sources were researched for their various positives and negatives. It was determined that the energy source should be efficient, converting a reasonable amount of AC current from a small amount of natural energy, that the source should be unique in nature, that it should provide an adequate amount of energy over time to the storage solution, that it should be relatively low-profile so as not to attract the attention of passersby, and that it should be of relatively low cost.

Table 3 - Windbelt Alternatives Pugh Chart

Alternatives	**Design 1 Windbelt**	**Design 2 Solar**	**Design 3 Turbine**	**Design 4 None**
Efficient	5	5	3	1
Unique	5	3	3	1
Adequate	3	5	4	1
Low-Profile	5	3	1	5
Cost	5	3	1	5
Difficulty	4	2	1	5
Sum	**27**	**21**	**13**	**18**

Table 3 shows the four different sources that were considered, including windbelt energy, solar energy, turbine, and not having a sustainable solution. The windbelt solution was chosen because it was rated highly in all required areas except adequate current. The smaller current supplied by the windbelt was considered a reasonable downside to using the technology. When compared to other sources, such as solar, it is less costly and lower profile. Turbines would drive up the cost, and make low-profile implementation next to impossible, and not having a sustainable solution would be a less costly yet time-consuming alternative.

Node Communication Method

Each node must be able to communicate with other nodes over the entire length of the trail. Figure 2 shows the length of the trail along with the potential node positions.

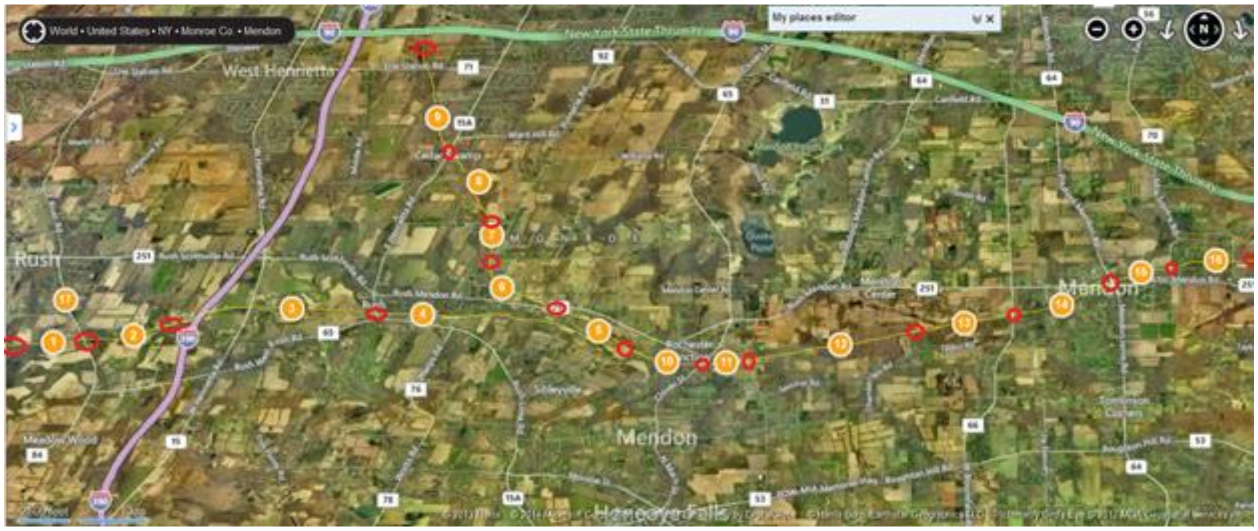


Figure 2: Approximate Trail Node Locations and Trail Entry Points

Each numbered orange circle in Figure 2 is the approximate location of where a node should go. The hollow red circles are trail access points. There is one node in between each access point. The orange circle labeled "17" is Jared's house, and that is where the gateway node will be. For scale, the trail left-to-right stretches about 15 miles, and the vertical distance between the top-most and bottom-most nodes (9 and 11) is roughly 4 miles.

The nodes must be able to send data to each other and to the database. The greatest estimated distance between two nodes is roughly 1.75 miles. Therefore, the selected solution must be able to communicate data at a distance of at least 2 miles, although the greater the distance the better. There were three main methods considered: a giant directional antenna, SMS communication, and ZigBee radio modules.

The first idea considered was to deploy a giant Wi-Fi antenna at Jared's house (position 17 in Figure 2), which is only half a mile from the south-west end of the trail. By pointing 1 or 2 antennas in the direction of the trail, most, if not all, of the trail can be covered. By going with the directional antennas, it means that there is a centralized node at the antenna location that is collecting data from each of the trail nodes, and syncing that data with all the other trail nodes, along with the database. Some of the antennas researched could reach distances of over 28 miles, and were cheaper than buying 17 ZigBee modules. However, upon further investigation, the legality of deploying such devices was unclear. Most directional antennas are designed for point-to-point communications; that is communicating with one and only one other point. For the purposes of the sensor network, the antennas need to do point-to-multipoint communication, or communicate with more than one node at a time. Using antennas designed for point-to-point communication for point-to-multipoint can be illegal depending on the antenna. In addition to the legality issue, the trail is not straight, and therefore a directional antenna might not fan out enough to cover the entire trail. An omnidirectional antenna, or an antenna that goes in all directions, could be used to get a good fan out, but no omnidirectional antennas were found that had a long enough range. The final problem with using a directional antenna is that each of the nodes still need some kind of Wi-Fi dongle. USB Wi-Fi dongles are less than $10, which are cheaper than buying 17 ZigBee modules. However, the MCU may not be compatible with the USB Wi-Fi dongles. Due to the questionable legality, and the fact that the fan-out of the antenna is unknown, this idea was rejected.

Another plan to have the nodes communicate with each other was to connect GSM chips to each node and have the nodes communicate with each other through SMS. GSM chips are designed for low-powered mobile devices, so the power footprint would probably be small. With SMS, the range between each of the nodes is no longer a problem, as they can be deployed anywhere, and still be able to send a text message, assuming a phone signal. Although the GSM chips are cheap, being $10 to $20 apiece, sending SMS messages requires a carrier such as AT&T or T-Mobile, and they are not free. Eventually, sending SMS messages will cost more than using a ZigBee module. Due to the recurring cost, this idea was rejected.

The chosen solution was to use the Digi International XBee 900 HP radio module. The XBee has built in mesh networking, so no mesh networking software needs to be written. It also has a range of 9 to 28 miles, depending on the antenna and line-of-sight. The XBee has a UART interface, which allows it to send data from the MCU. Most importantly, the documentation also clearly states how to use the device legally so the project does not get shut down by the FCC.

Gateway to the internet

The trail nodes need some kind of access to the internet to send its data to a database. One of the concepts considered was to use 3G or 4G attached to one or two of the nodes, and send data to the database over 3G through the cellular network. The good thing about 3G is that range is not a problem. As long as there is a good cell phone signal, data from the trail nodes can go anywhere in the world. The problem with 3G is that it requires a data plan from a wireless carrier, making a recurring cost per month which is less than desirable. Also, 3G seems to use a lot of power, which is a problem since the trail nodes are running on a battery, and power is limited.

Another idea was to purchase a gateway device that would convert the data from the trail nodes to either Ethernet or USB, and connect it to a PC at Jared's house (node 17 in Figure 2). Jared's house has an internet connection which will be the access point to the World Wide Web so the data from the trail nodes can be sent to the database. This idea would make the most sense, but there is a cost problem as gateways would incur an additional expense. There is, however a cheaper alternative.

The choice for the gateway to the internet was to use a Raspberry Pi and an XBee. The XBee will communicate with the trail nodes, and send any data that needs to be written to the database through the Raspberry Pi's UART connection. The Pi has an Ethernet port, so it can send data from the trail directly to the internet. The Pi has an added bonus as well. It can act as both the gateway and the web server at the same time.

Web Server / Cloud storage

The data from the trail nodes needs to be saved to a database somewhere. That data must also be able to be accessible through a web browser via web pages. One idea that was considered was using a third party for the cloud storage of the database and for hosting the web site. There are free hosting services, such as x10Hosting, so cost is not a problem. However, the host might give little control over what can or cannot be installed on the server. Also, additional software needs to be written to send data from the Raspberry Pi to the third party host so the host has the most up-to-date data from the trail. One advantage of having a third party web host is that it can handle a large number of users. However, since heavy traffic is not expected on the site due to the limited number of users that will have access, this advantage does not hold much weight. Another advantage is that a third party site might do automated backups, but this can be easily done on a local server as well.

Another concept would be to use the same Raspberry Pi that will function as a gateway node for the web server to manage the “cloud” storage for the database as well. Using the Pi in this way gives total control over what software can or cannot be installed on the system. The Pi also saves some coding, as there is no need for software to be written that will send the trail node data to some third party host from the Pi; the data from the trail nodes will go directly to the database, which will live in the Pi’s memory. The Pi does come with a few drawbacks however. The first one is that it cannot handle as many users as a third party host. This was not considered an issue due to the low volume of users that will be allowed to access the server. The second downside is if Jared’s house loses power, then the website will go down as well. A “status” webpage page might need to be hosted somewhere else, (e.g., another team member’s house), which will ping the Pi every few minutes and display on the status webpage whether or not the trail data website is down or not.

The selected concept, is to use the Raspberry Pi as the server. The main reason is so there is total control over what is able to be installed on it, and the choices are not limited to whatever the third-party hosting site offers.

Design

The commuter tracking system overview is shown in Figure 3, and is composed of 16 nodes established along the length of the Lehigh Valley Trail that communicate traffic information to a gateway node through use of external antennas. The gateway then stores the data in an associated database. Users have access to the traffic information and the sensor net through a web based graphical user interface.

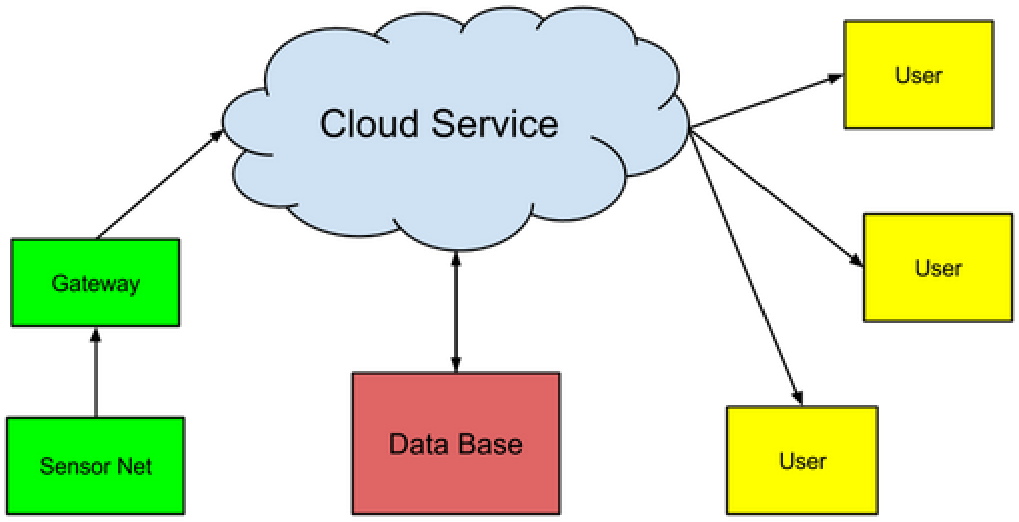


Figure 3 - System Overview

Each sensor node meets a low power requirement while providing the ability to capture and process images. The network is also be configured so that nodes have a common operating picture. In this manner, information is made redundant and synchronization of data is maintained. The approximate radio frequency range of each node is at least 3 miles to accommodate 15 miles of trail with some overlap of neighboring nodes.

Figure 4 shows a hardware block diagram illustrating flow of power and data throughout the system from a high-level perspective. Power is provided via a trickle-charge, sustainable, low output energy resource and fed into a power regulation device, where it is converted from AC to DC. The regulation device also rectifies voltage output prior to supplying power to the image sensor, image processing component, MCU and radio module to ensure they are operating in nominal conditions. A Lithium Ion battery is used to store the harvested energy, which is split using a simple voltage regulation device that provides regulated current to a secondary boost converter and step-down buck converter. The buck converter then provides 3.3V to an XBee radio module. The secondary boost converter provides 5V to two Cortex M processors located on the camera module. An image sensor on the same module provides motion capture information to one of the processors, and the other oversees the transfer of data to and from a radio module connected to an external antenna. An infrared module is connected to the auxiliary processor, allowing the node to operate during low light conditions and waking the image sensor from sleep mode when it senses a commuter.

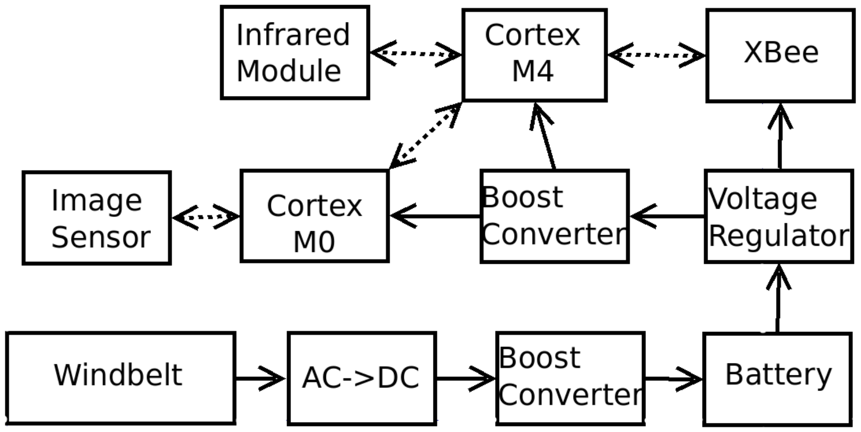


Figure 4 - Hardware Block Diagram

The Power System

Windbelt

A windbelt is used to trickle-charge a lithium ion battery due to the inherent need for sensor nodes to be deployed for longer durations with less manual intervention. Table 3 shows the alternative sustainable energy sources that were considered, along with point values for several different categories of need. It was desired that the chosen design provide sustainable energy efficiently and adequately with a low profile and low cost. Utilization of a unique design was also considered to give the project an interesting touch. The windbelt design met requirements more than other choices, which included solar and turbine, and the choice of not having a sustainable energy element. Due to its potential effect on overall sensor network deployment, the windbelt was considered a high-risk design component. It was later discovered that the energy source is more than capable of providing adequate charge to the battery management system.

The commuter tracking network is powered using a pre-charged lithium ion battery capable of providing 3.7 - 4.2 V to voltage regulation devices and then to connected peripherals. The windbelt produces a small AC current based on the aero-elastic flutter effect to provide a trickle-charge to the battery. This is done for a fraction of the cost that would be incurred if using turbine or solar sources, and can provide exponentially more power for any given wind speed than turbine technology. As shown in Figure 5, this can be done on a specific model windbelt for wind speeds as low as 3.5 m/s, supplying 0.2 mW of AC power across a 390-Ω load at 70 Hz. This power output improves exponentially as wind speed increases, taking full advantage of variability of environmental conditions.

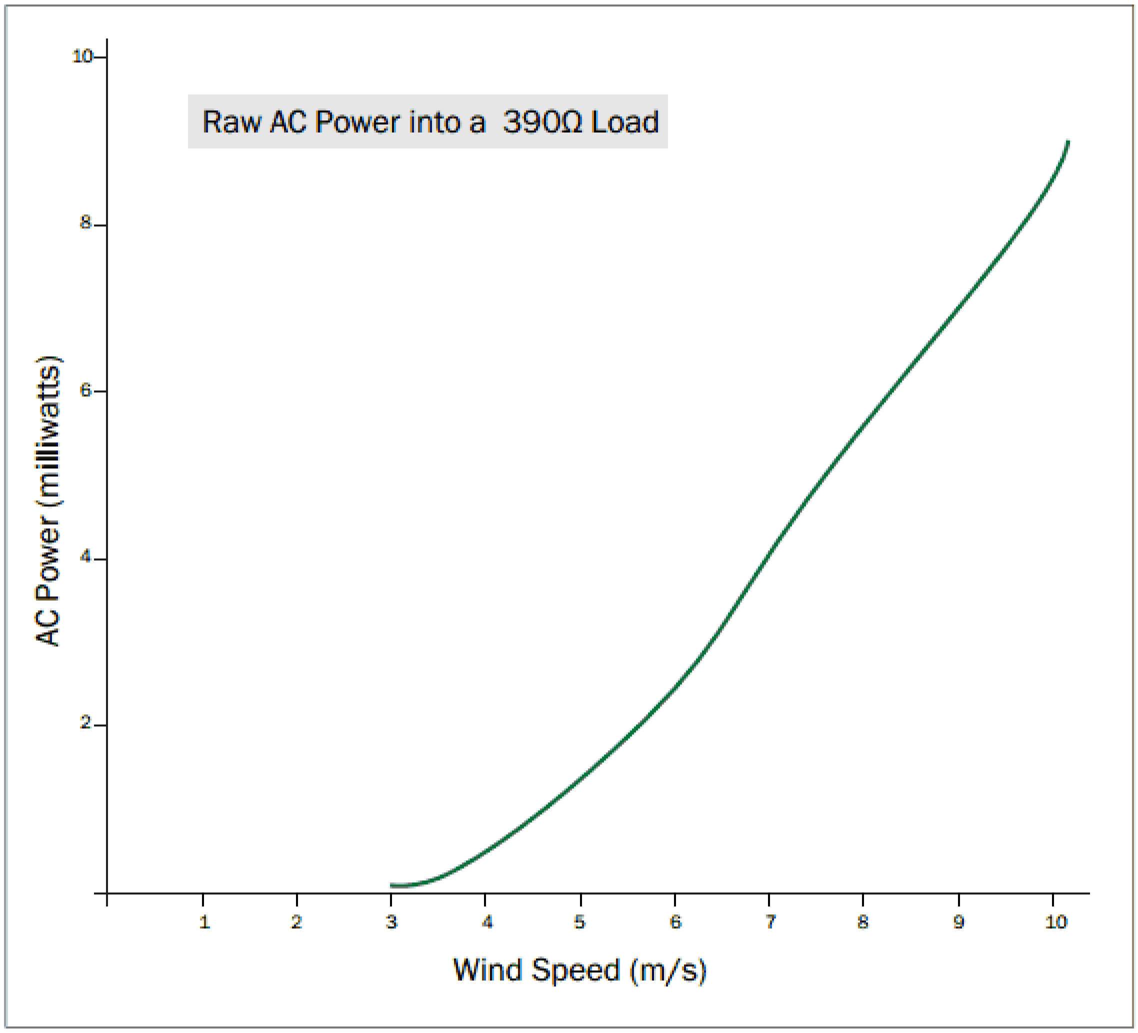


Figure 5 - Wind Speed (m/s) vs. AC Power (mW)

Likewise, power is provided by the Li-Ion battery during time periods where wind is not present. The flutter effect produces power from wind by vibrating a band pulled taut across a small opening. The band repeatedly moves a small magnet across one or more coils, inducing current upon connected wires. As an example, the µicroWindbelt shown in Figure 6 is able to power wireless sensors through the attached 3-V DC buffered supply. This model exhibited characteristics similar to those we wished to provide in our own custom implementation of windbelt technology. The design was developed in an evolutionary manner, generating working prototypes that were adequately tested and implemented through use of RIT's 3-D printing facilities.

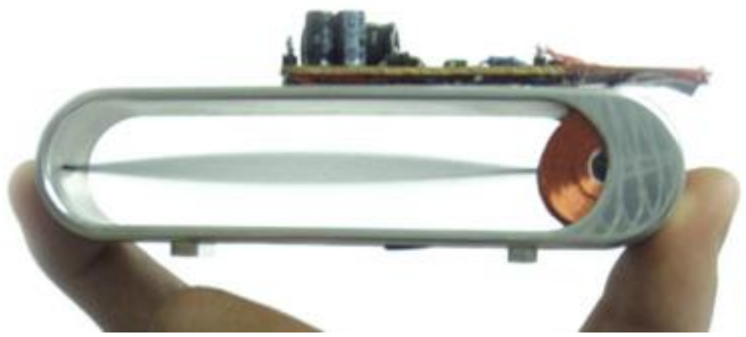
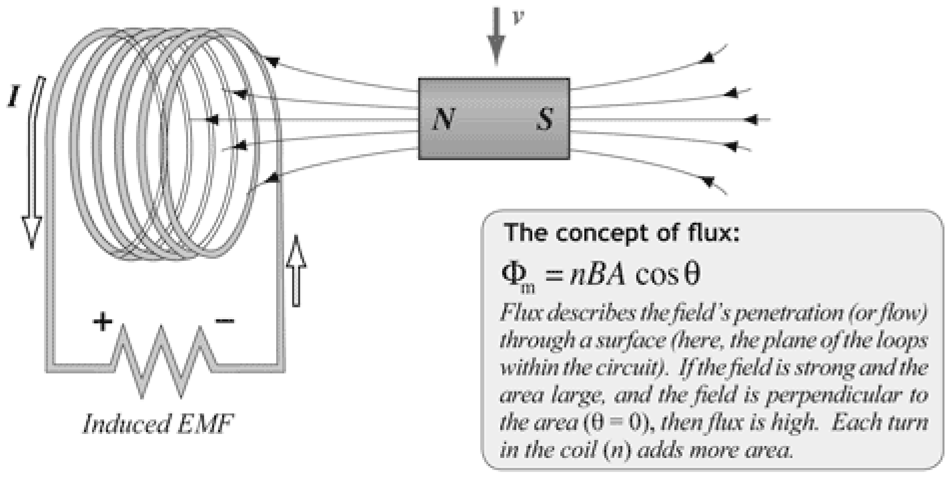


Figure 6 - The Humdinger µicroWindbelt

Dimensions of the design shown in Figure 6 are 13 cm by 3 cm by 2.5 cm. The design was developed by Shawn Fraye of Humdinger LLC, and was specifically derived for use with microcontrollers, providing 0.2 mW at 3.5 m/s, 2.0 mW at 5.5 m/s, and 5.0 mW at 7.5 m/s at 70 Hz. It can provide anywhere from 50-200+ Wh at the listed dimensions, and charges large onboard capacitors.

Since the battery used in the application was of much larger capacity than a coin-cell, a larger version of the windbelt was prototyped. Several iterations of design were implemented and tested. The first consisted of a simple wooden structure of about two feet in length. A band was bolted between the two ends of the structure with two large hard-drive magnets attached to it. To test the device, wind was blown across the band. This prototype was unsuccessful, as the magnets were too heavy, and the inductor coil was not positioned properly according to Faraday's Law, shown in Figure 7. In order to increase the amount of electromagnetic flux induced, the pole of the magnet should have been positioned perpendicular to the end of the coil. This orientation would increase the rate of change of flux through the circuit, which increases proportionally to induce EMF. Also, due to the fixed nature of the band, the frequency at which the belt vibrated could not be adjusted. This made characterization of the device fairly linear, and uninformative. A limited number of windings on the coil was yet another limiting factor, as area of the coil increases with each winding.



The concept of flux:

$$\Phi_m = nBA\cos\theta$$

Flux describes the field's penetration (or flow) through a surface (here, the plane of the loops within the circuit). If the field is strong and the area large, and the field is perpendicular to the area ($\theta = 0$), then flux is high. Each turn in the coil (n) adds more area.

Figure 7 - Faraday's Law

The second design shown in Figure 8 was longer, measuring at about 30", and utilizing a pulley structure to hold the belt perpendicular to the cardboard body of the windbelt. The material allowed for several holes to be drilled along the length of the belt, where the pulley structures could be repositioned for testing of different belt lengths. Problems with this design were related to the pulleys, as they could not hold the belt taut enough to provide consistent frequencies of 60 Hz when vibrating. Nuts used to keep the pulleys fixed continuously came loose, and were hard to manage. The cardboard was not thick enough to prevent flexing due to stress applied by the components, and the distance of the belt from the tube created a space restriction, so the coil was improperly positioned underneath the magnet to account for lack of space.

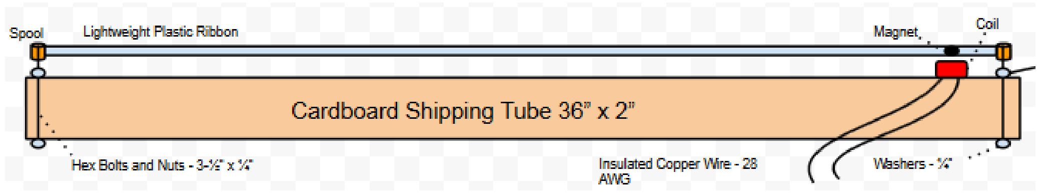

Spool
Lightweight Plastic Ribbon
Magnet
Coil
Cardboard Shipping Tube 36" x 2"
Hex Bolts and Nuts - 3-½" x ¼"
Insulated Copper Wire - 28 AWG
Washers - ¼"

Figure 8a - Windbelt Prototype 2 Side View

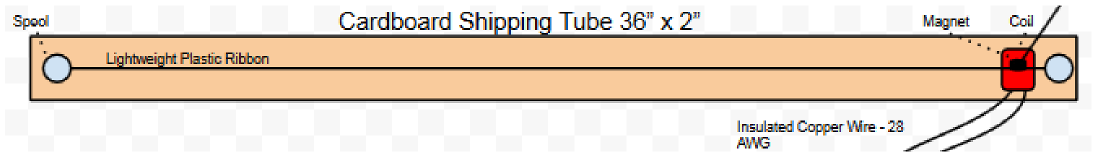

Spool
Cardboard Shipping Tube 36" x 2"
Magnet
Coil
Lightweight Plastic Ribbon
Insulated Copper Wire - 28 AWG

Figure 8b - Windbelt Prototype 2 Top View

An industrial sized fan was used to provide a 15 m/s wind source to the second prototype, and due to the position of the coil in relation to the magnet, consistent AC voltage could not be produced. The belt would, on occasion, produce some AC voltage when the coil moved into range of the magnet, but amounts were not adequate for the application.

The third windbelt prototype shown in Figure 9 was constructed from 2” diameter PVC pipe cut at a 30” length. Pulley assemblies like those used in prototype two were more effective in this design, since the added friction from the PVC material kept them from moving. Bolts could also be tightened as much as desired without flexing the pipe. Although the belt could produce AC voltages in excess of 1.5 V at times, performance was once again inconsistent due to the orientation of the coil.

Several options were considered to fit the coil in the correct position for Faraday’s Law. Toroid inductors were considered to allow for a lower profile coil, and the ability to position the band lower alongside the coil. This would bring the coil very close to the magnet, allowing for stronger induced flux. An alternative was to change the oscillation so that the magnet was vibrating vertically alongside the end of the coil.

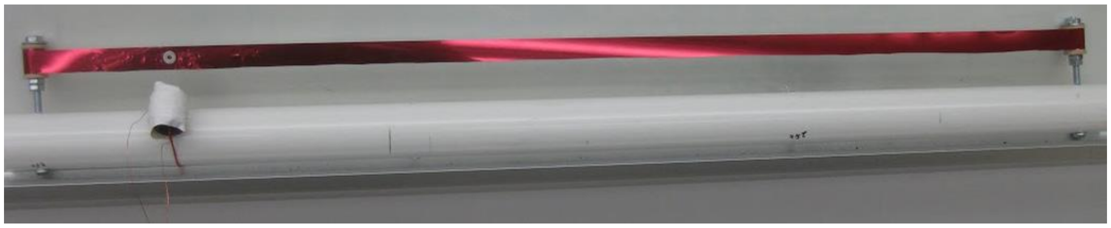


Figure 9 - Windbelt Prototype 3

The final prototype of the windbelt shown in Figure 10 changed the orientation of the magnet. In addition, a second coil was added to allow for twice the induced flux. Each coil had 2000 windings, increasing the area of the magnetic field and the rate of disruption. A brace screwed into the end of the fixture was used to vary the tension of the belt so that a 60-70 Hz vibration could be achieved. Magnets were fixed in place on the belt with magic tape, and coil assemblies were held to the fixture with duct tape.

Creating the coils involved epoxying a ferrite core to two Plexiglas plates. This allowed for more efficient coil winding with more control over placement of the wire. Once the coil assemblies were created, they were mounted and raised up or lowered using wooden shims. This allowed for very close proximity to the magnet, and more EMF.

Using the fan shown in Figure 10, varying wind speeds were tested on the device at a variety of angles and distances from the wind source. On average, the windbelt produced approximately 2.0 V_{AC} peak to peak for a 15 m/s wind source at about 60 Hz. Further optimizing the signal involved moving the magnet closer to the end of the band, and tightening the band to produce frequencies higher than 60 Hz. This resulted in about a 0.5 V AC improvement at about 65 - 70 Hz. Tightening the band beyond a certain point actually had adverse effects on the signal. Similarly, using larger magnets and positioning them too close to the end of the belt also produced weaker signals. Once the windbelt prototype was optimized, it was prepared for outdoor testing. A small breadboard with a half-wave rectifier circuit was fixed to the top, and the inductors were placed in series with the rectifier to produce pulsating DC.

Outdoor testing was accomplished by attaching the windbelt to a lamp post on RIT campus. The belt was left in 25 m/s winds, and gathered data via a BeagleBone Black ADC attached to the output of the half-wave rectifier. The ADC data was then transferred via Wi-Fi to a database connected to the project website. Data gathered showed that the ADC value was capped during wind gusts, and enough power was generated to keep the boost converter running for extended periods of time.

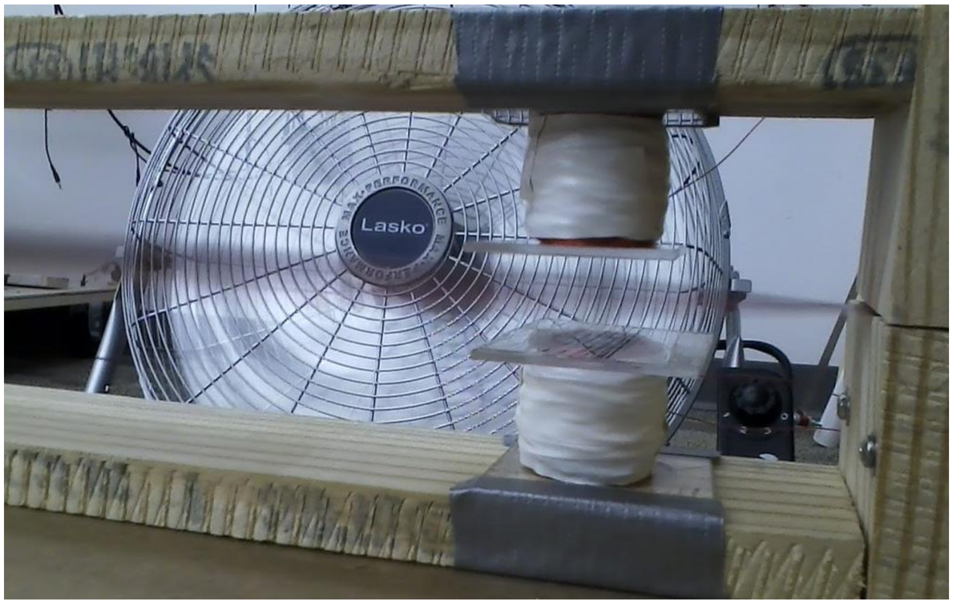


Figure 10 - Windbelt Prototype 4

The Power Circuit

The trickle-charge supplied to the Li-Ion battery was of a low enough current to keep a fully charged battery at full capacity while not causing damage to the battery. This was ensured through use of a rectification circuit positioned after the windbelt that would cut charge of the battery to at most 7 C, or 7 charge hours to full battery capacity. This limit was imposed for safety reasons, since a Li-Ion battery could potentially overcharge, causing damage to the cells and possibly plating out the metal on the battery. Other safety features of the battery include reverse polarity protection, a charge temperature limiter that prohibits charge when the battery is out of range of a certain threshold, and discharge current protection to prohibit reverse flow of current from inhibiting the generation of inductive current.

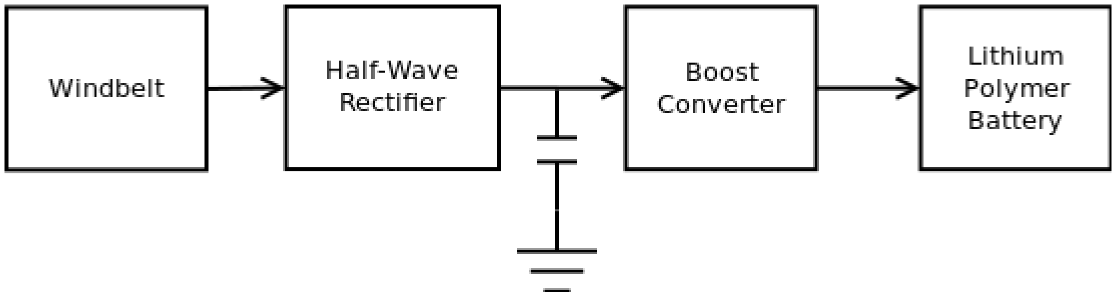


Figure 11 - Signal Conditioning Block Diagram

Figure 11 shows a block diagram of the input signal as it passes from the windbelt to the rectification circuit, which is composed of a simple Schottky diode. The diode will function to keep the AC signal from going into negative values, essentially cutting off the lower portion of the waveform and maintaining an adequate DC range for the boost converter to exploit. A capacitor is added after the rectifier to smooth its output, also keeping signal levels from dropping too far between periods and raising the average voltage. The complete half-wave rectifier along with smoothing capacitor design is shown in Figure 12.

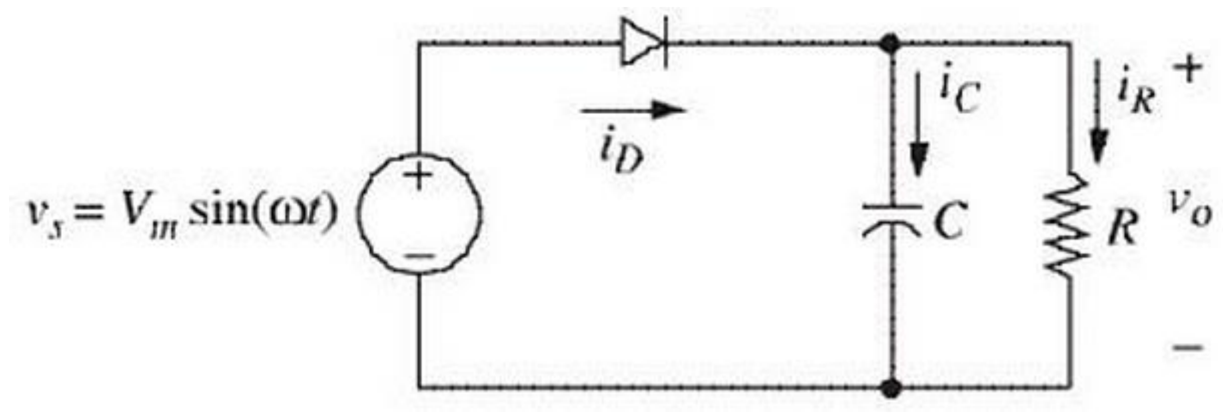


Figure 12 - Half-Wave Rectifier with Smoothing Capacitor

As shown, the current I_D will pass from the 60 - 70 Hz AC source provided by the windbelt through the diode and be divided into I_C and I_R. The voltage levels across the resistor are represented by the waveform shown in Figure 13.

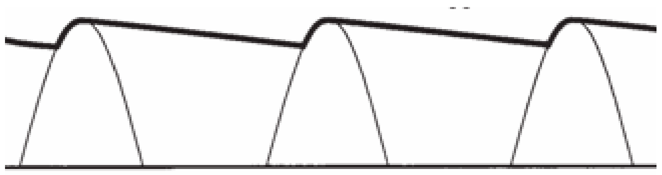


Figure 13 - Waveform Result of Half-Wave Filter

The AC signal is kept at a high enough value for the remainder of the signal conditioning circuit to work effectively. Negative AC values are eliminated due to reverse current protection of the diode, and risk associated with applying reverse current to the battery is also reduced.

Testing the voltage produced by the windbelt prior to leaving the signal conditioning circuit showed that it was generating voltages of about 1.25 V_{DC} on average at the output of the rectifier, with 25 m/s gusts of wind producing higher than 1.8 V_{DC} maximum voltages. This was considered a reliable source for the battery management system.

To further reduce risks associated with providing an unstable source to the battery, a TI bq25504 boost converter is connected to the output of the rectification circuit via the VIN_DC port. Initial testing showed that the half-wave rectifier was sufficient in providing a DC source to the converter, so a full-wave rectifier alternative was considered, but not used. Documentation for the battery management system actually states that the device prefers pulsating DC, and implementation of the full-wave rectifier would increase the amount of voltage drop prior to entering the device as well as increase costs. This was considered an unnecessary compromise for relatively minimal power gains.

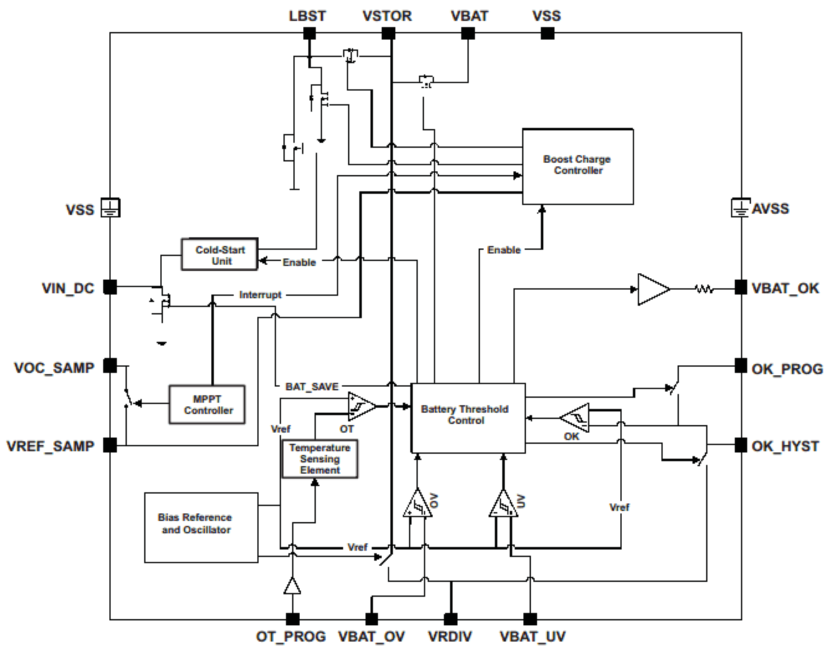


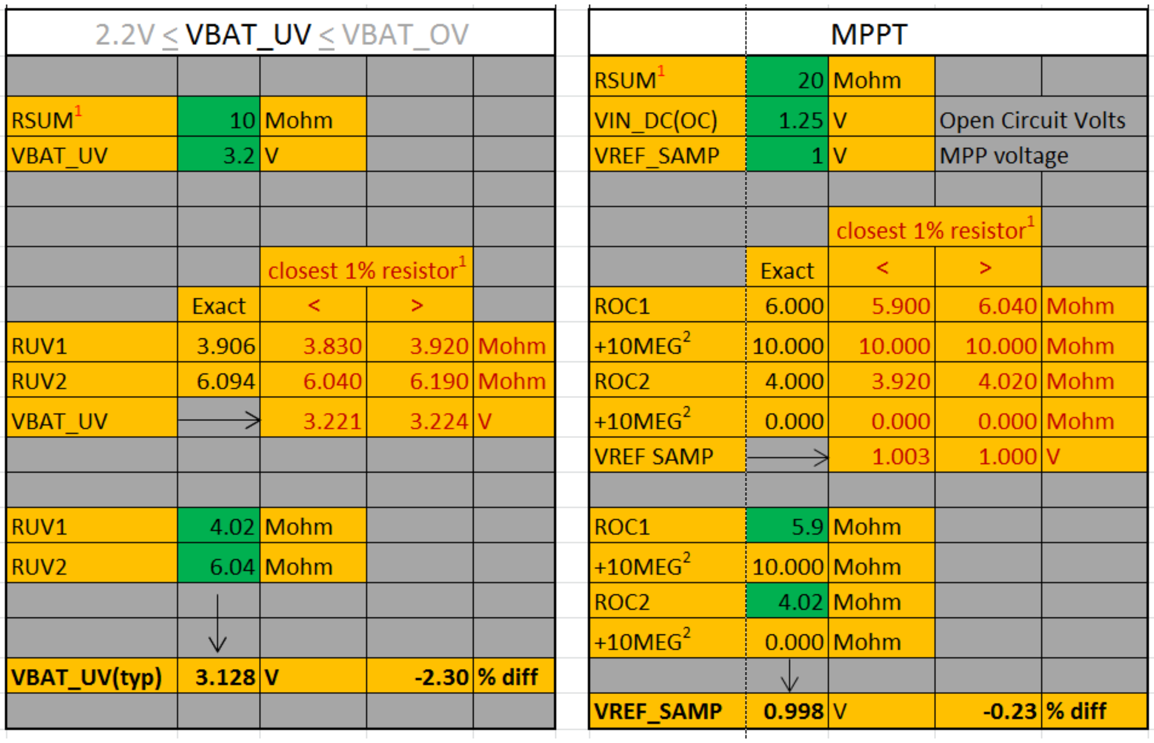
Figure 14 - Texas Instruments bq25504

As shown in Figure 14, the bq25504 boost converter takes the DC signal provided by the rectifier and drives it up to a voltage specified by the overvoltage threshold configuration, which can then be used to safely charge the Li-Ion battery without worry that levels will exceed safe charging voltage. The TI device was chosen due to its ultra-intelligent, nano-power design that is ideal for implementation in a wireless sensor node application. It provides safety features to the Li-Ion battery such as programmable overvoltage and under-voltage protection, and thermal shutdown protection. The converter has programmable hysteresis features that allow it to retain past voltage values for power level tracking (MPPT), can turn on and remain running with minimal power-on voltage, and can even warn attached microcontrollers of pending loss of power. A low voltage cold-start of 330 mV or higher is needed to start the device, along with a V_{IN} of at least 80 mV to allow it to continue harvesting, allowing the battery to safely charge for longer time periods with minimal power consumption by the device. When the device is not connected to an analog load and is not amplifying, it consumes only 330 nA.

MPPT, or maximum power point tracking, is a technique used in energy conversion systems that allows for continuous sampling of the output of a power source, and the dynamic application of resistive load that will allow the circuit to obtain maximum power during changes in environmental conditions and variations of input voltage. Internal potentiometers are automatically adjusted to divide the voltage differently when it reaches a certain threshold value. An external reference voltage can also be set by MCU to keep voltage levels at a specified minimum. All threshold values are fully programmable.

The boost converter also has a “battery good” port that can be programmed to monitor voltage of the storage device, and send a signal when it reaches a certain level. If signals exceed a programmable overvoltage or under-voltage threshold, the connected MCU can send a signal to shut off all connected peripherals. As reassurance that the boost converter works with most modern day cell chemistries, Texas Instruments provides documentation confirming compatibility with Lithium Polymer and Lithium Ion batteries. The boost converter is of great importance in an energy acquisition scenario, as changes in the environment, specifically wind speed and temperatures, would otherwise have a much greater effect on the power being provided to the Li-Ion battery, and the health of the battery as well.

Prototyping the BQ25504 was quite involved. The process began with a spreadsheet provided by Texas Instruments that allows the user to enter desired thresholds for the VBAT_OV, VBAT_UV, VBAT_OK_HYST, and MPPT signals and outputs the required resistor values to obtain those signals. The MPPT, or maximum power point threshold, is usually 0.7 - 0.8 of the windbelt’s open circuit voltage. Since this was found to be approximately 1.25 V on average from windbelt testing, the VREF_SAMP signal that provides the MPPT pin with the appropriate divider reference was set to 1 V, or 80% of the average input voltage. Figure 15 shows that the resulting resistor values for ROC1 and ROC2 are 5.9 MΩ and 4.02 MΩ, respectively.



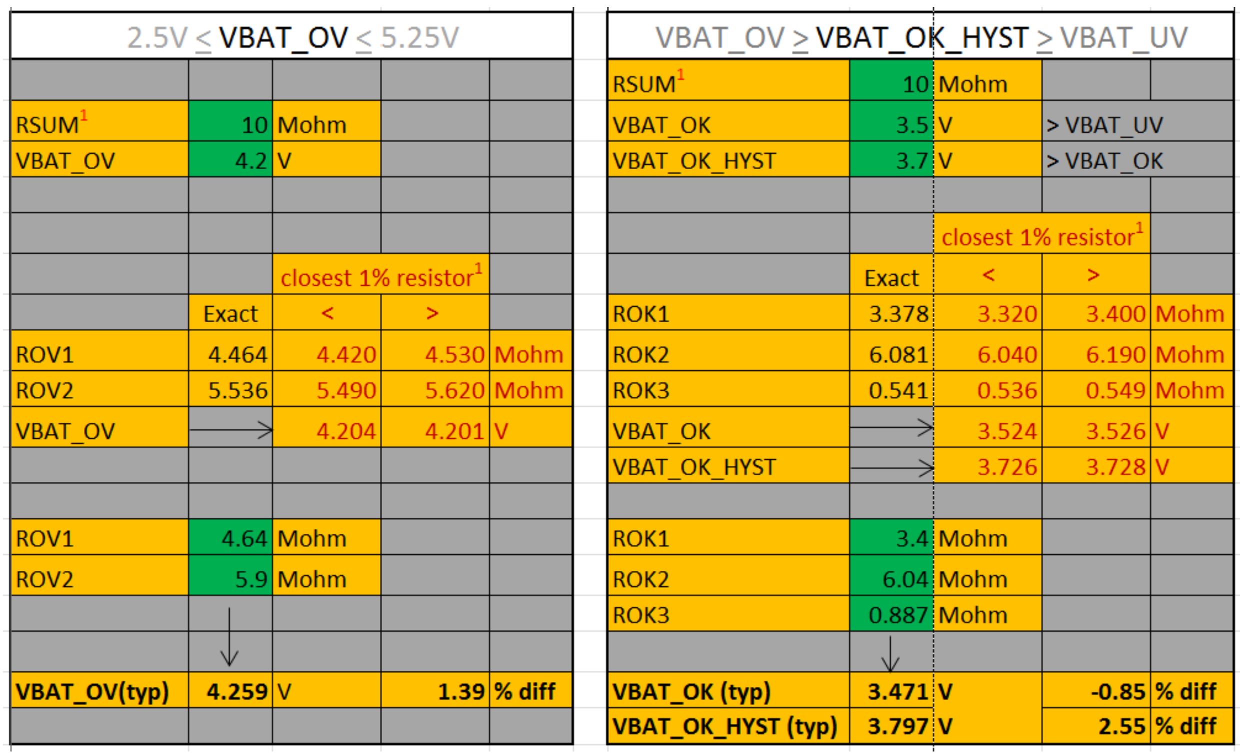
2.2V ≤ VBAT_UV ≤ VBAT_OV				
RSUM[1]	10	Mohm		
VBAT_UV	3.2	V		
		closest 1% resistor[1]		
	Exact	<	>	
RUV1	3.906	3.830	3.920	Mohm
RUV2	6.094	6.040	6.190	Mohm
VBAT_UV	→	3.221	3.224	V
RUV1	4.02	Mohm		
RUV2	6.04	Mohm		
	↓			
VBAT_UV(typ)	**3.128**	**V**	**-2.30**	**% diff**

MPPT				
RSUM[1]	20	Mohm		
VIN_DC(OC)	1.25	V	Open Circuit Volts	
VREF_SAMP	1	V	MPP voltage	
		closest 1% resistor[1]		
	Exact	<	>	
ROC1	6.000	5.900	6.040	Mohm
+10MEG[2]	10.000	10.000	10.000	Mohm
ROC2	4.000	3.920	4.020	Mohm
+10MEG[2]	0.000	0.000	0.000	Mohm
VREF SAMP	→	1.003	1.000	V
ROC1	5.9	Mohm		
+10MEG[2]	10.000	Mohm		
ROC2	4.02	Mohm		
+10MEG[2]	0.000	Mohm		
	↓			
VREF_SAMP	**0.998**	V	**-0.23**	**% diff**

Figure 15 - Spreadsheet Calculations of VBAT_UV and VREF_SAMP

The spreadsheet returns the closest 1% resistor values that can be purchased. As shown in Figure 15, entering the desired values into the two bottom green cells results in the realized VREF_SAMP signal for the chosen hardware, which is about 0.998V. The -0.23% difference was considered acceptable deviation from ideal operating conditions.

Figure 15 also shows the calculation for VBAT_UV, or the under-voltage threshold for the VBAT signal that will cause the boost converter to cut current to the VBAT pin. This was set to 3.2 V, since charging a Lithium Ion battery at voltages lower than that has practically no effect on the battery. Values of 4.02Mohm and 6.04Mohm were chosen to get an actual VBAT_UV of 3.128 V.



2.5V ≤ VBAT_OV ≤ 5.25V				
RSUM[1]	10	Mohm		
VBAT_OV	4.2	V		
		closest 1% resistor[1]		
	Exact	<	>	
ROV1	4.464	4.420	4.530	Mohm
ROV2	5.536	5.490	5.620	Mohm
VBAT_OV	→	4.204	4.201	V
ROV1	4.64	Mohm		
ROV2	5.9	Mohm		
	↓			
VBAT_OV(typ)	**4.259**	V	**1.39**	**% diff**

VBAT_OV ≥ VBAT_OK_HYST ≥ VBAT_UV					
RSUM[1]	10	Mohm			
VBAT_OK	3.5	V	> VBAT_UV		
VBAT_OK_HYST	3.7	V	> VBAT_OK		
		closest 1% resistor[1]			
	Exact	<	>		
ROK1	3.378	3.320	3.400	Mohm	
ROK2	6.081	6.040	6.190	Mohm	
ROK3	0.541	0.536	0.549	Mohm	
VBAT_OK	→	3.524	3.526	V	
VBAT_OK_HYST	→	3.726	3.728	V	
ROK1	3.4	Mohm			
ROK2	6.04	Mohm			
ROK3	0.887	Mohm			
	↓				
VBAT_OK (typ)	**3.471**	**V**	**-0.85**	**% diff**	
VBAT_OK_HYST (typ)	**3.797**	**V**	**2.55**	**% diff**	

Figure 16 - Spreadsheet Calculations of VBAT_OV and VBAT_OK

Figure 16 shows how the overvoltage and “battery ok” resistor nets were configured. Since Li-Ion batteries have maximum charging voltage of 4.2 V, this was entered as the overvoltage threshold. Resistor values of 4.64Mohm and 5.9Mohm were used to realize a VBAT_OV of 4.259 V. Due to the fact that when load is added to the circuit, some voltage is lost in favor of current, the slight overshoot was considered acceptable. The microcontroller signal VBAT_OK was set to trigger if it was either less than 3.471 V or greater than 3.797 V. These values were chosen after load was connected to the circuit, and long term testing was completed to determine average operating range of the circuit.

After determining the correct values to place within each resistor net, the Cadence schematic in Figure 17 was generated using those values, along with capacitance and inductance values noted in the data sheet for the device. The inductor of the circuit was set to 22 uH, and load capacitance was set to 4.7 uF. To further reduce noise in the circuit at the input and output nodes of the circuit, a 0.1 uF capacitor was placed in parallel, and to reduce the runtime of the simulation in Cadence, a 0.47 uF capacitor was used to model the battery at the VBAT node.

Transient analysis was run with the windbelt input set to 2.5 V_{p-p} at 60 Hz. As shown in Figure 18, this produced signal levels that were close to those calculated within the spreadsheet. In the first graph voltage at the load pin, VSTOR, was approximately 4.2 V. In the second graph, the input voltage is shown as pulsating DC, oscillating between 1.0 - 1.2 V when the device exits cold start and enters steady state. This is shown to take about 70 ms with no load, and a capacitance of 0.47 uF at the VBAT pin. The third graph shows that the VBAT voltage matches that of the VSTOR pin. In ideal conditions, with no load attached, this is the case. However, it was expected that voltage would drop at both nodes depending on the current draw of the load connected. This is because the battery is expected to provide a majority of the power to the circuit. The battery management system was only put in place to reduce the rate of discharge, not to dramatically increase the charge of the battery or maintain its current charge.

A QFN (quad flat no leads) to DIP (dual inline package) circuit board was purchased in order to prototype the circuit on a breadboard with the resistor configurations tested. When choosing the PCB, care was taken to choose a design with a 0.5 mm pitch

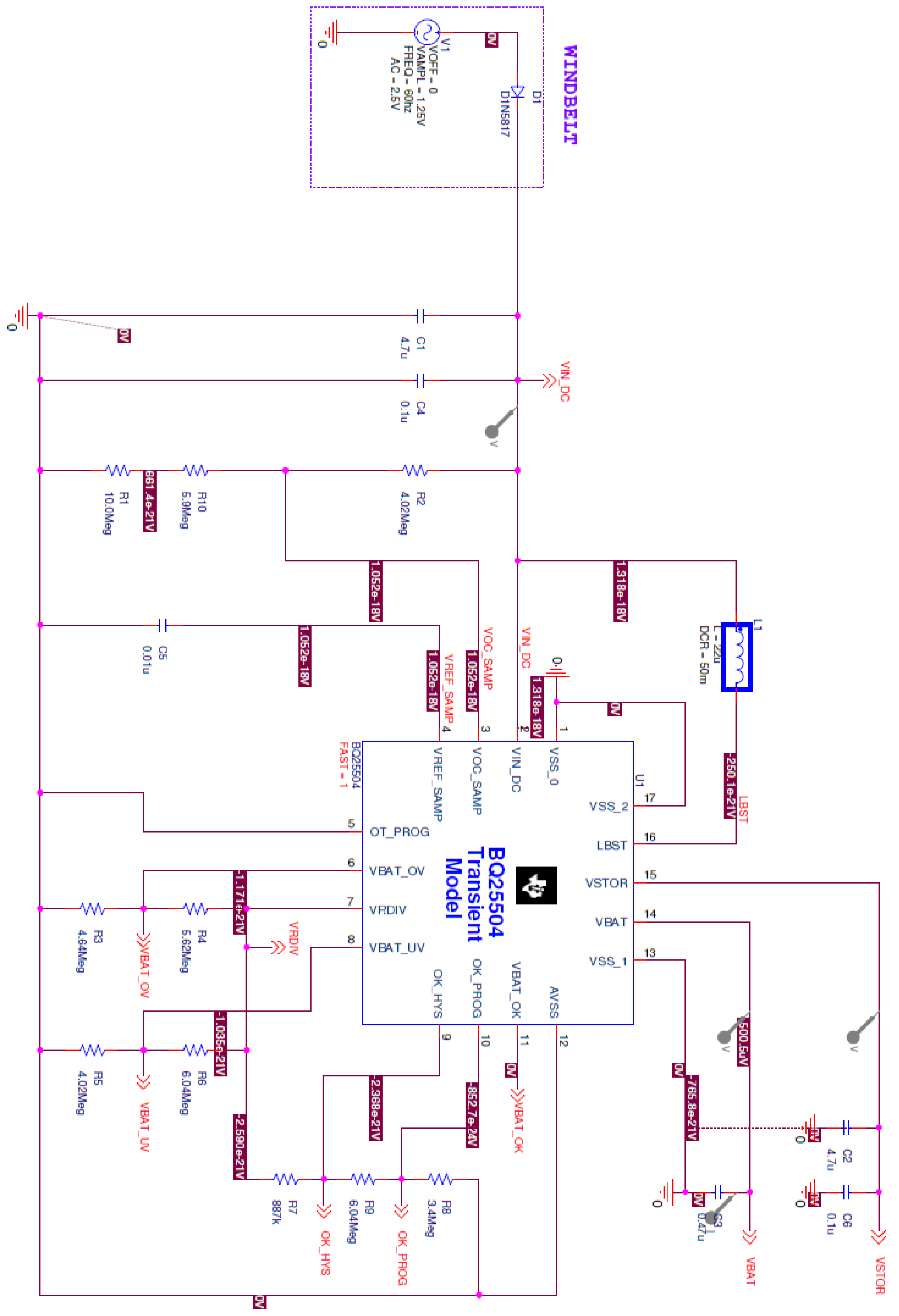


Figure 17 - TI BQ25504 Cadence Schematic

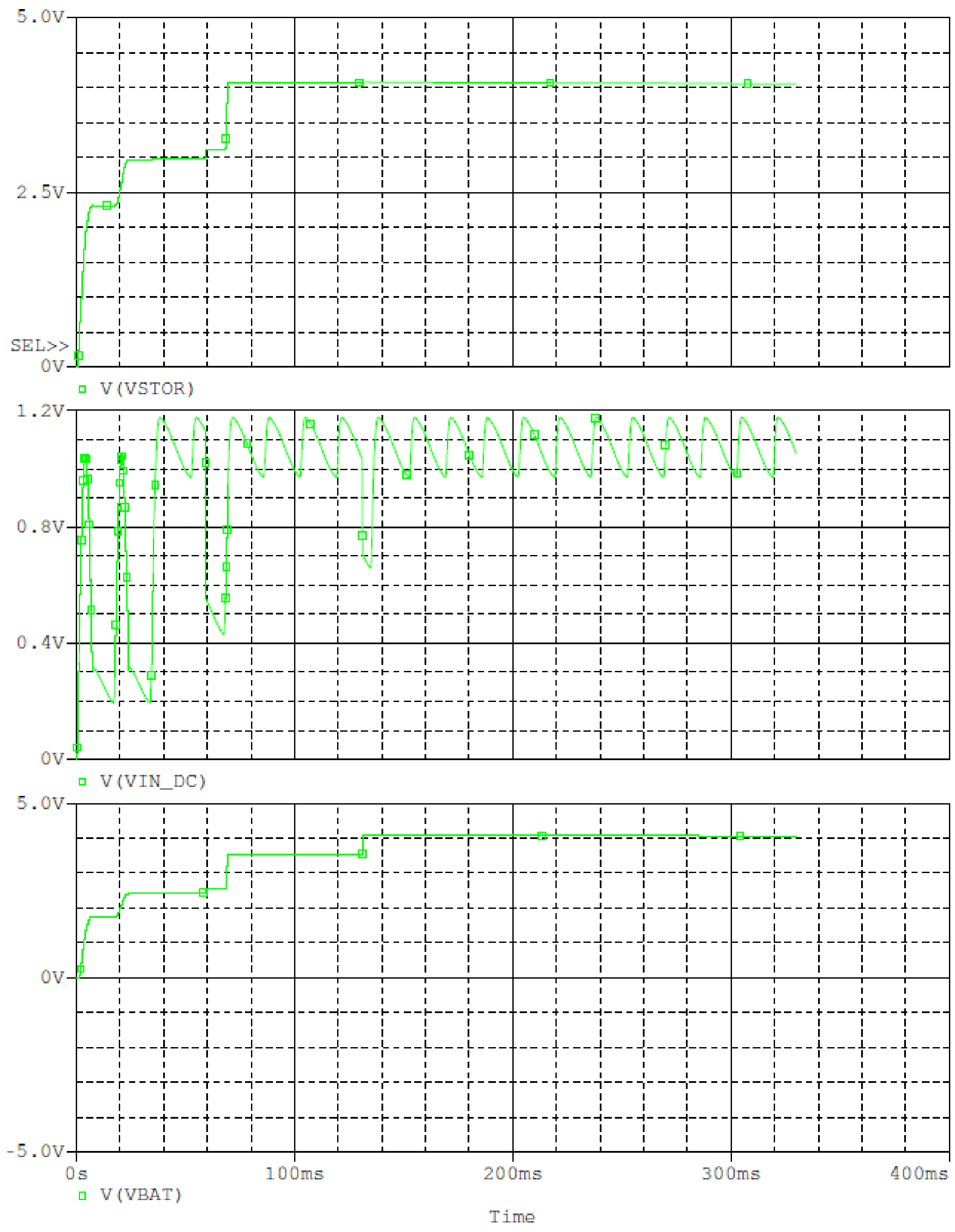


Figure 18 - Cadence Transient Analysis Results

between pins and a 3 x 3 mm body. Figure 19 shows the PCB with and without the boost converter soldered to it. Break-away headers are clearly visible in the image furthest to the right, allowing for prototyping on a breadboard.

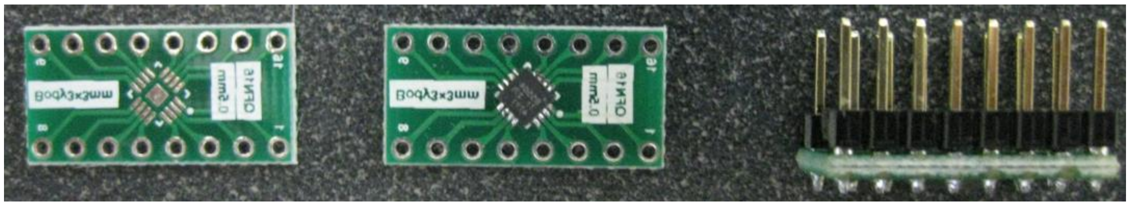


Figure 19 - QFN to DIP Converter

Since the QFN has recessed leads, conventional soldering methods could not be used to adhere the chip to the board. Rather, a reflow soldering technique was used to melt pre-applied solder paste and allow it to cool. The heating and cooling process allowed the solder to attract to the leads on the underside of the chip, forming a strong bond. To ensure that the solder paste was applied correctly, the stencil in Figure 20 was created out of thin aluminum pieces using a cad drawing tool and a printer. The marked sections were then etched out of the aluminum with an acid wash.

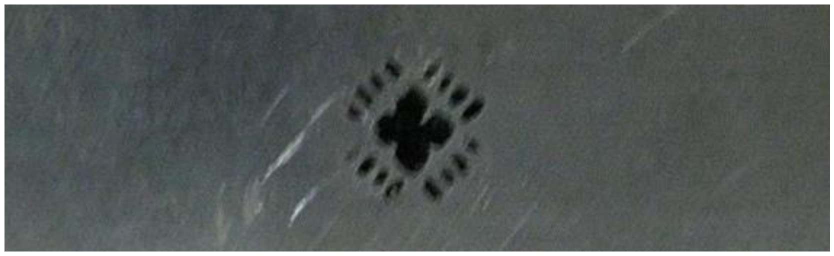


Figure 20 - Solder Paste Stencil

When soldering, care was taken to not apply too much paste to the pads, and to align pin one with the indicator on the PCB. The reflow process was handled by a modified toaster oven with an MSP430 microcontroller that was programmed to allow for the correct heating and cooling process over a specified period of time. Once the solder paste had been melted and cooled, headers were soldered on and it was placed on a breadboard along with the components specified in Figure 17.

Figure 21 shows the completed breadboard prototype. All resistor nets are clearly labelled in green, while the orange wires are labelled as the outputs of the circuit. The red wire at the bottom of the prototype carries the input signal coming from the half-wave rectifier, and the windbelt.

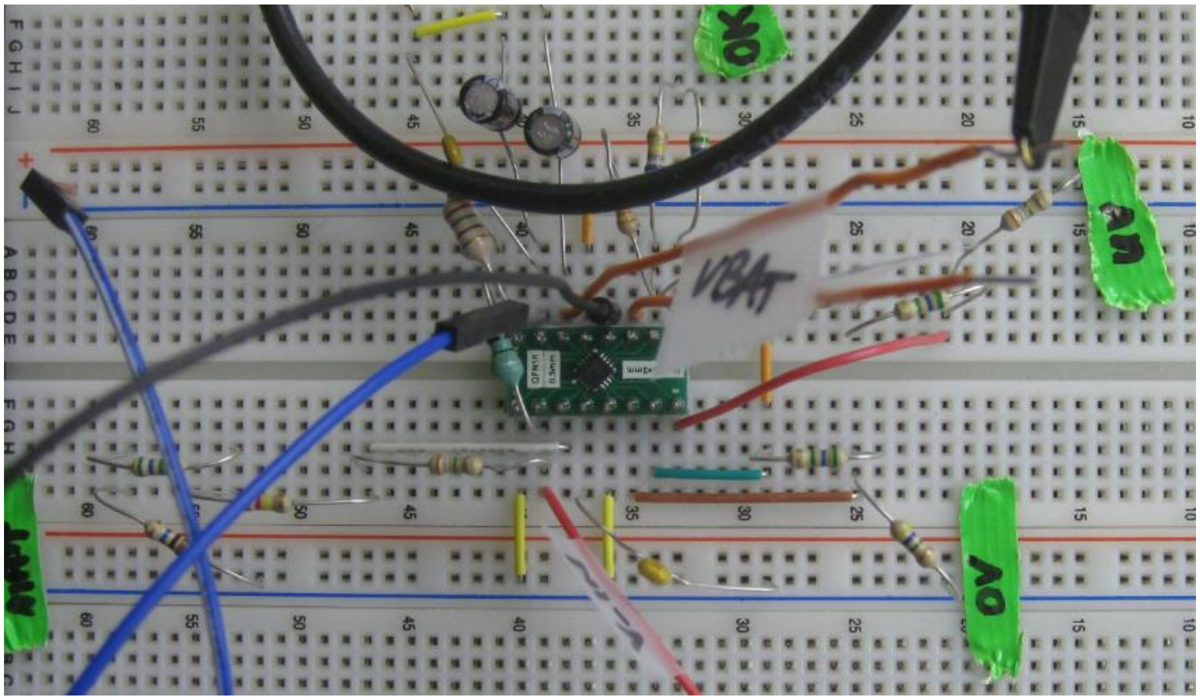


Figure 21 - BQ25504 Breadboard Prototyping

To characterize the circuit, a maximum AC voltage of 1 V was applied gradually. At about 480 mV, the boost converter powered on, reaching the overvoltage threshold at about 4.15 V in approximately 200 ms as shown in Figure 22. Both the VBAT and VSTOR pins match in voltage since no load was applied. To test dynamic voltage response, the input voltage was dropped to a level lower than the current one. The device responded by dropping output voltage slightly, but MPPT functionality raised it back up to a level within the VBAT_OK range.

Next, the circuit was characterized with the battery attached to the VBAT pin, and produced output very similar to that in Figure 22. To emulate the current that would eventually be drawn by peripherals attached to the VSTOR pin, a 22 ohm and 10 ohm resistor were each placed in parallel at that node. This would account for a 182 mA current draw in parallel with a 222 mA one. Figure 23 shows the resulting output with the load attached.

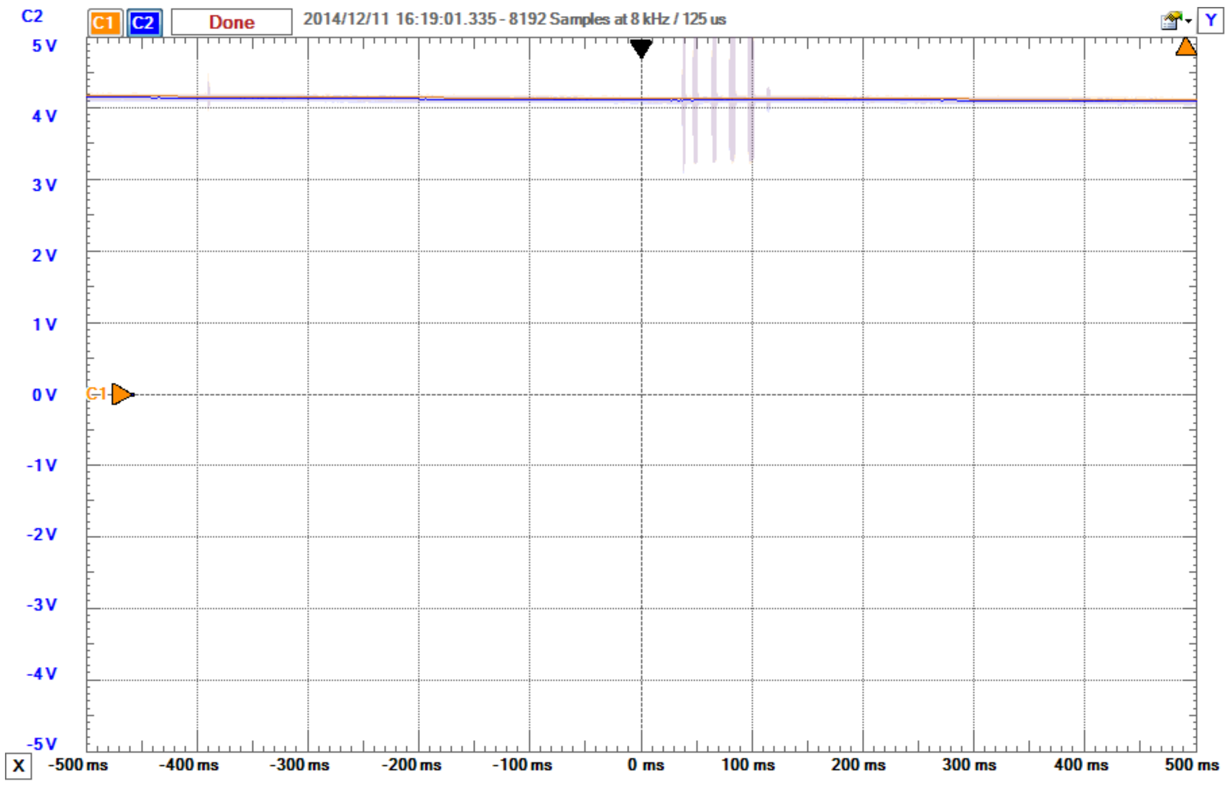


Figure 22 - BQ25504 Prototype VSTOR and VBAT Voltage

Voltage levels drop to about 3.7 V on the VSTOR pin and 4.05 V on the VBAT pin. This new VSTOR level was used to determine how to tune the VBAT_OK signal that would be used to determine if the battery was in a good state.

To test varied input voltages, the VIN signal was changed, and then dropped to 0 V. The effect this had on the output was negligible, since the attached battery took over powering the circuit once VIN dropped below 80 mV.

After characterizing the prototype with and without load, the EAGLE PCB design program was used to create a custom board for the BQ25504. First, the schematic was created as shown in Figure 24. EAGLE design libraries were downloaded from a variety

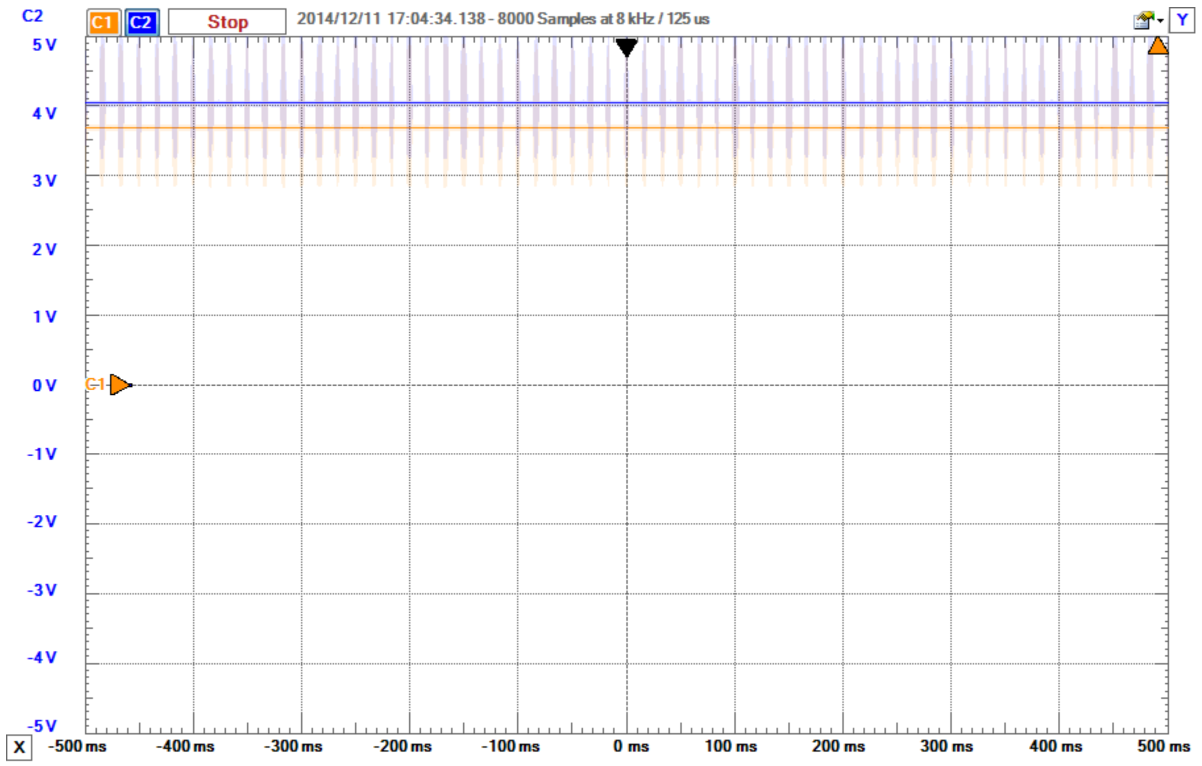


Figure 23 - Prototype VSTOR and VBAT with Load

of vendors, all of which were available through TI's documentation and online libraries. The design passed all electrical rule checks, and was converted to the layout shown in Figure 25, where the polygon tool was used to create power and ground planes, and routes between components were manually created.

Since the design would eventually be sent out to OSH Park for fabrication, their design rule check file was imported, and Gerber and Excellon files were created using their standardizations, as shown in Figure 26.

Surface Mount Technology facilities on RIT campus were utilized in order to solder surface mount components to the PCB. This involved a combination of reflow and traditional soldering techniques. Mainly, an air gun was used to shoot compressed air

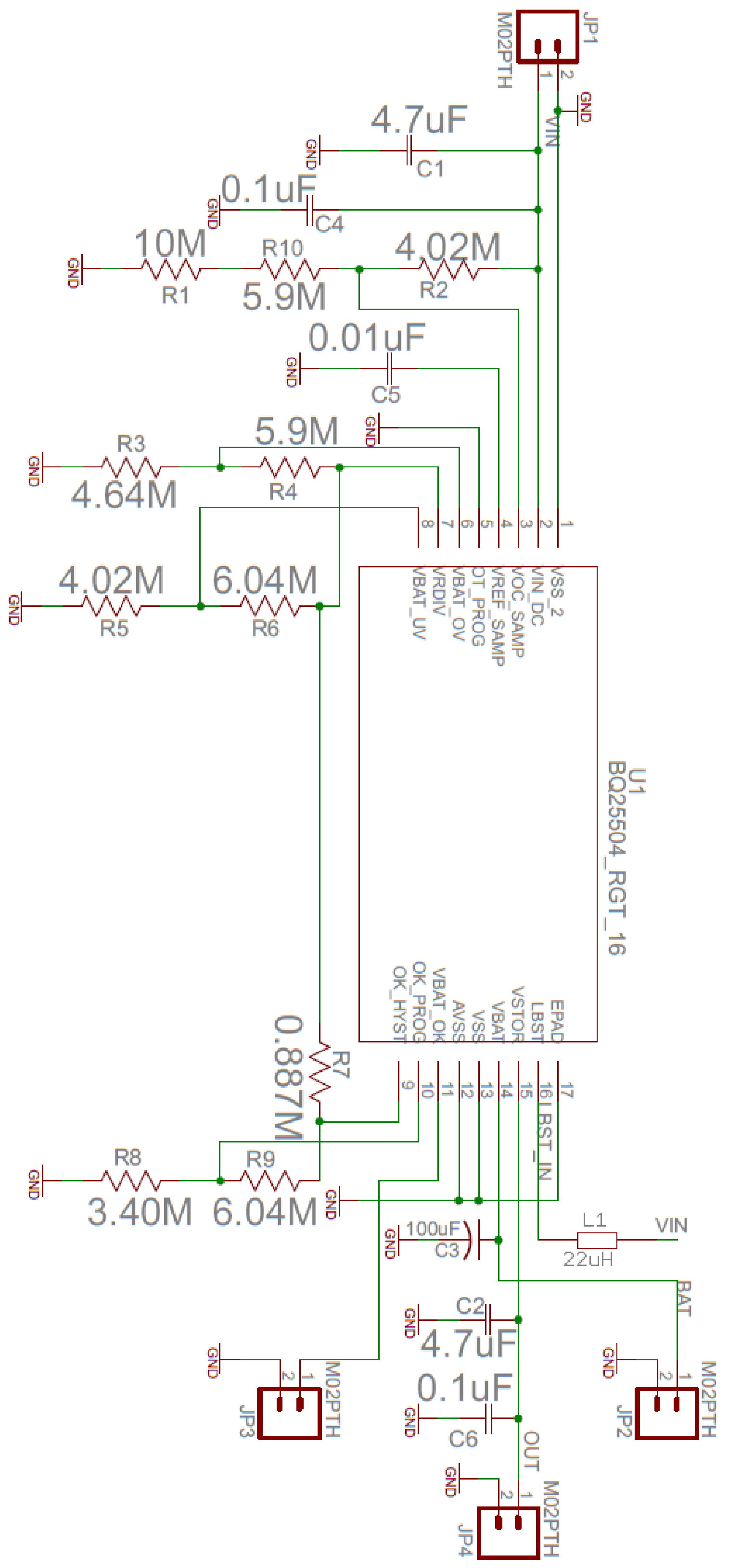


Figure 24 - EAGLE Design Schematic

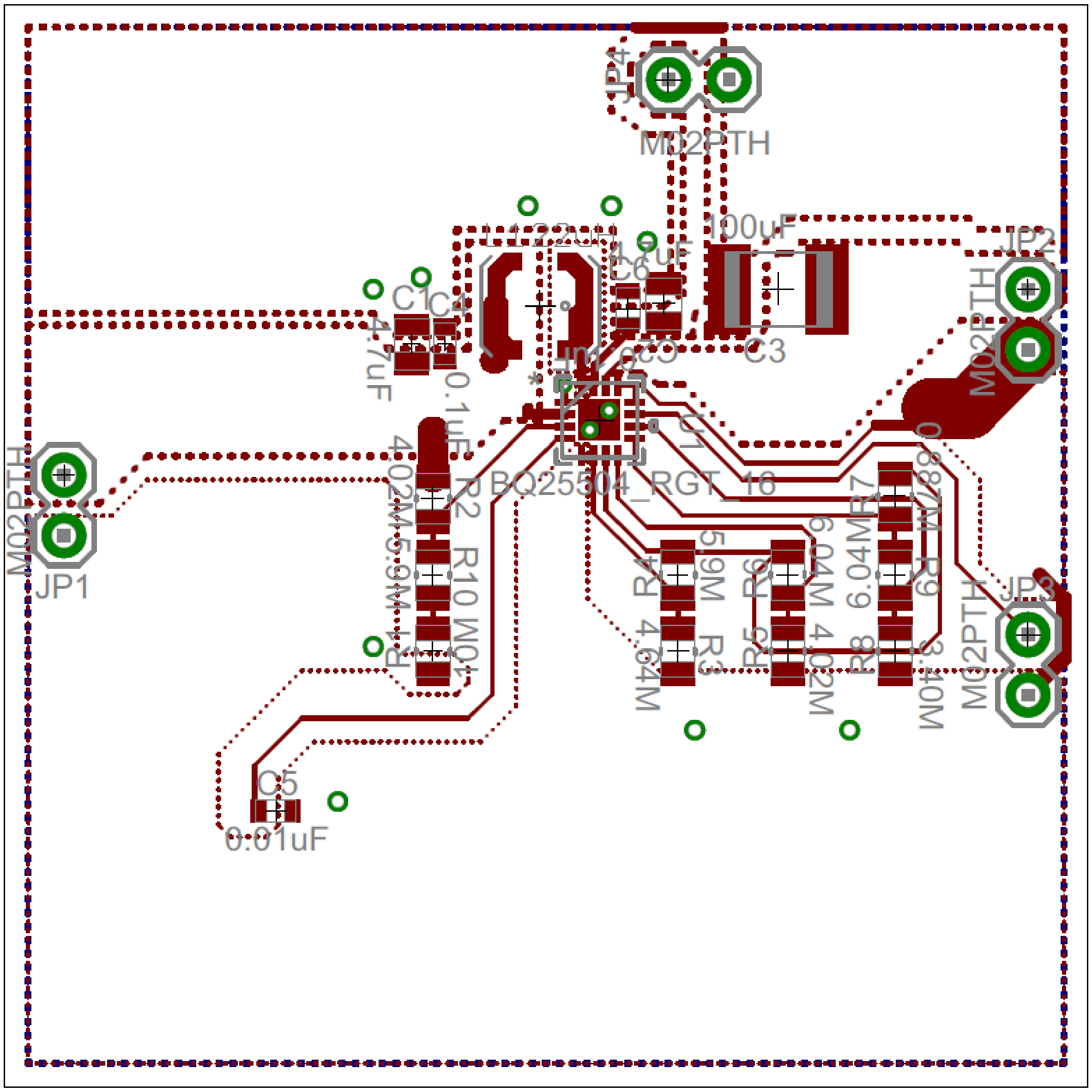


Figure 25 - EAGLE Design Layout

into a syringe filled with solder paste. The tool was actuated with a foot switch, and the board itself was watched closely with a microscope while carefully placing solder paste, and then each chip over the paste. A heat gun was used to reflow the solder, and the board was cooled prior to testing. An x-ray microscope was used to check for bridges in the solder after cooling.

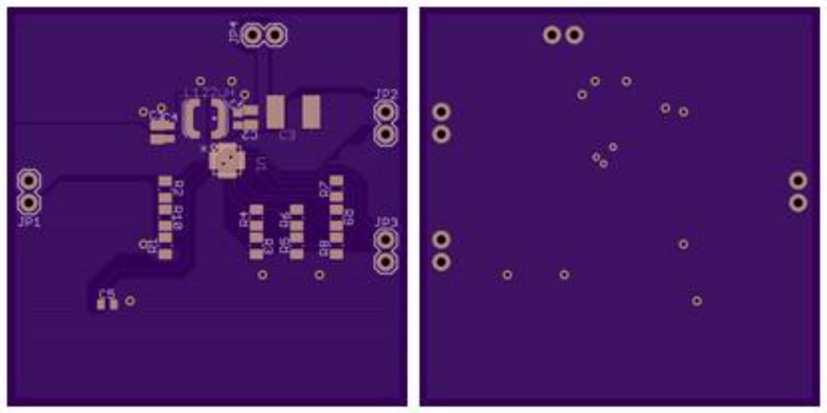


Figure 26 - BQ25504 Custom PCB

Figure 27 shows the completed PCB. To characterize the board, JP1 was connected to VIN. JP4 was labelled as VSTOR, and JP2 was connected to the Li-Ion battery. Voltage levels at VSTOR and VBAT pins are shown in Figure 28. Although similar to the readings from the prototype, the design benefitted greatly from being placed on a custom PCB, as there was much less signal noise at the output.

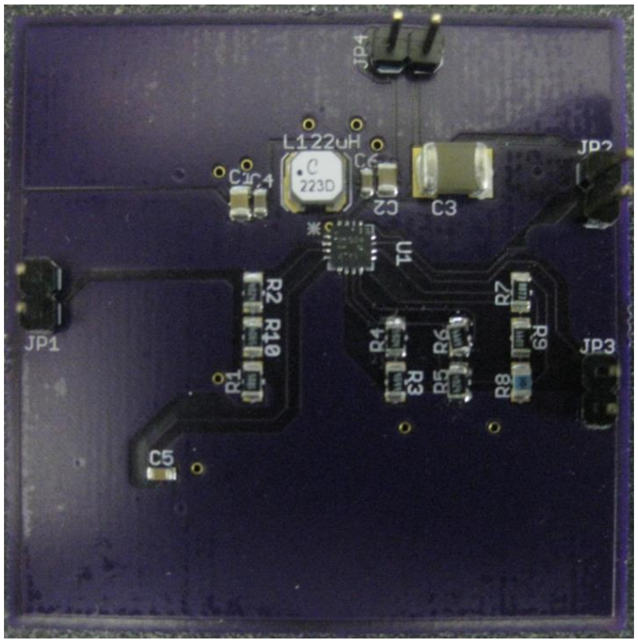


Figure 27 - Custom PCB with SMT Components

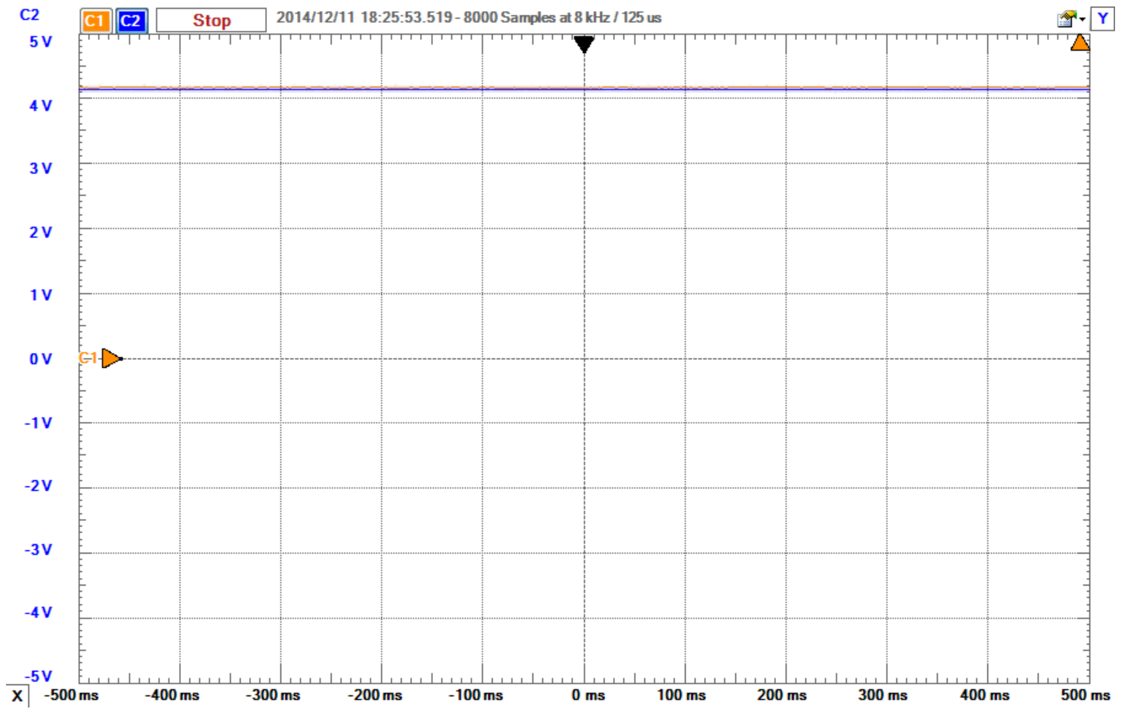


Figure 28 - Custom PCB VSTOR and VBAT

Due to the modularity of the design, and a split in voltage levels at the node after the boost converter in the overall system, the 3.7 V produced by the battery is passed into a secondary boost converter to be ramped up to a level that is acceptable by the Pixy camera module. The second line that is split off of the battery node is attenuated to provide levels acceptable by the XBee radio. These connections are made according to the block diagram in Figure 29. Components that will be implemented in future iterations of design, are also present in the diagram, and are described further in the retrospective.

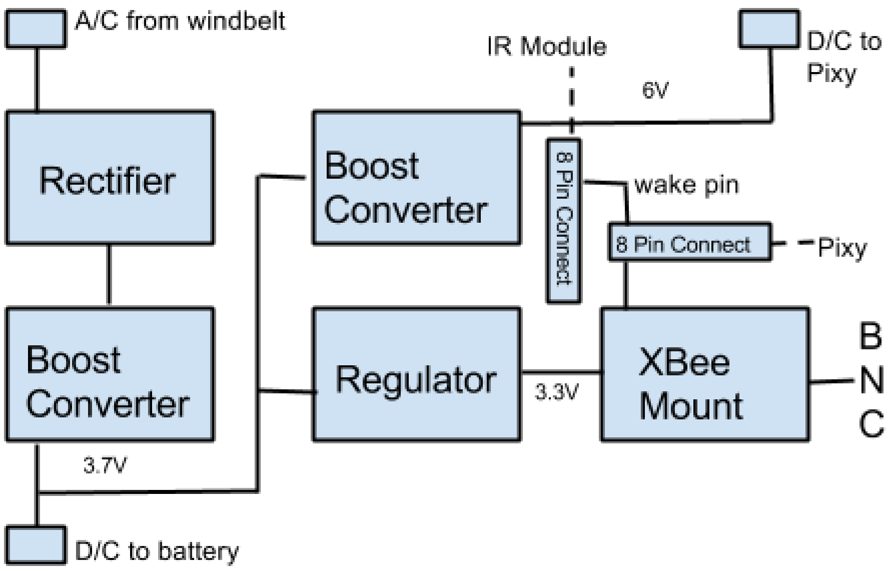


Figure 29 - Custom PCB Design

To step voltage down to a level acceptable by the XBee radios, a Texas Instruments TPS62740 was used. The EVM provided by TI was configured for our application, as it has a selectable voltage that can be changed by setting jumpers JP3-JP6 as shown in Figure 30. Each jumper was set to ‘1’ to get 3.3 V at the output of the regulator.

The buck converter is able to output up to 300 mA of current, which meets our specifications for the XBee radio. It also dynamically adjusts the gain to achieve 3.3 V at the output, regardless of the input voltage. This is important for when battery levels drop, and stage two must continue to drive the load.

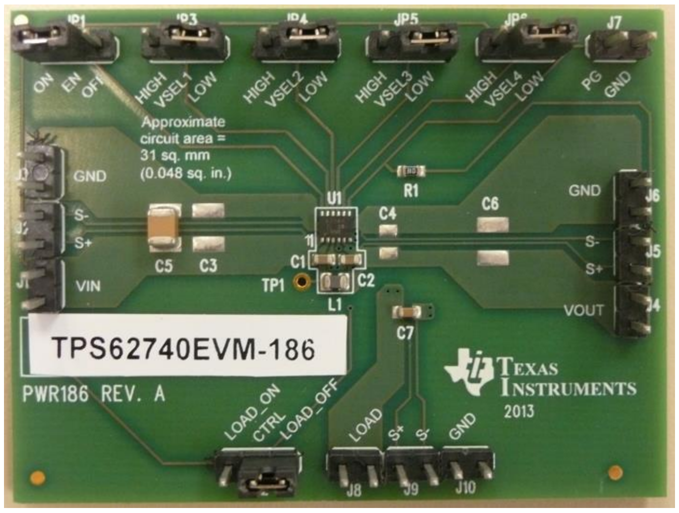


Figure 30 - Stage 2 Buck Converter

Figure 30 shows the stage two boost converter, the TPS61032. Texas Instruments provides an evaluation module to customers that is able to meet our application's specifications, so it was used to prototype the design.

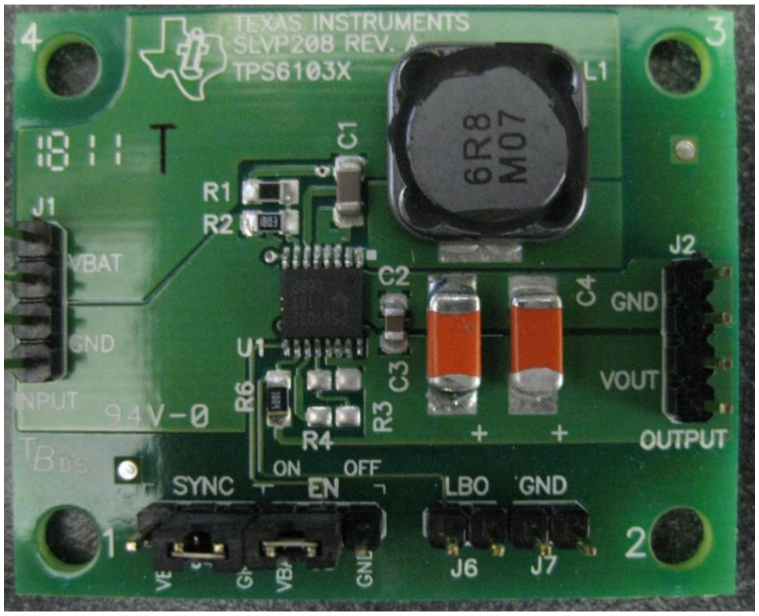


Figure 31 - Stage 2 Boost Converter

The boost converter generates 5 V at the output, regardless of the input to the module, which is enough to power the embedded camera. This once again helps in situations where the battery is discharged, but stage 2 must still be running. The dynamic gain adjust can be disabled, and replaced with a low power mode by changing the SYNC jumper to ground. This was disabled for characterization purposes, but may be used in future iterations of design to enable a low power state on the attached camera.

As shown in Figure 32, voltage characterization is similar to that emulated with resistors in the prototype testing. The voltage at the load pin, VSTOR, drops to about 3.67 V, since 80 mA is being drawn by the XBee radio, and about 150 mA by the camera. VBAT drops to about 4.08 V since more current is being drawn from the battery.

This means that since the battery voltage without load is about 4.15 V, the battery will not be charged to full capacity by the battery management solution. Since this was not the goal of the system, this is acceptable. It was found through extensive voltage level testing that the boost converter did slow the rate of discharge by several hours over a

24 hour testing period. This was calculated by tracking voltage over time. When the sustainable source was applied, the battery discharged to 4.4 V instead of 4.2V.

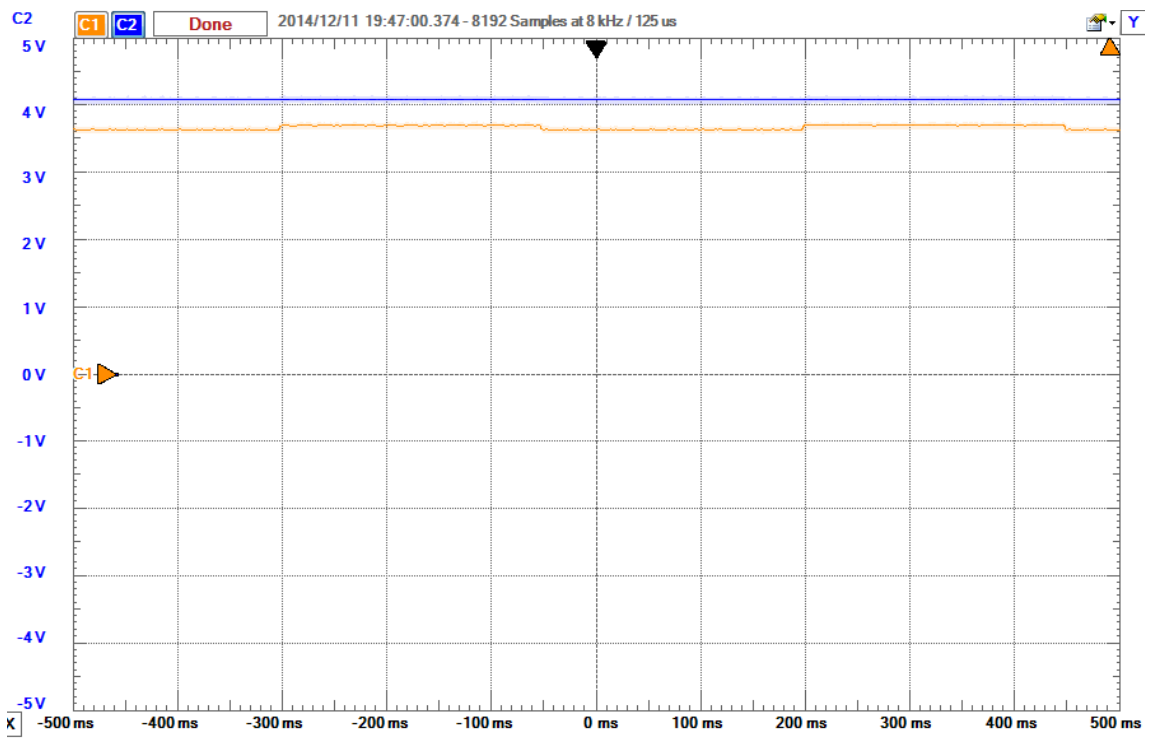


Figure 32 - PCB VSTOR and VBAT with Load

Battery

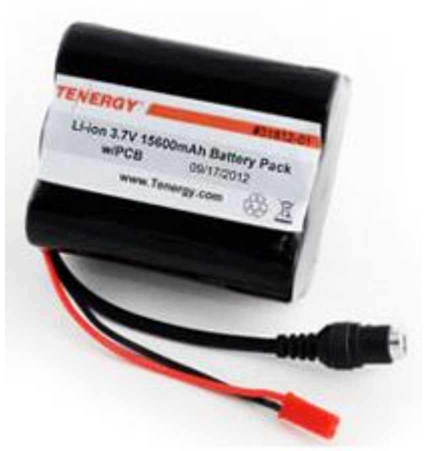


Figure 33 - Tenergy Li-Ion Battery Pack

The battery chosen for the design was the Tenergy Lithium Ion 3.7V 15600 mAh Battery Pack (model number 31812). A picture of the battery can be seen in Figure 32. This model was chosen for its relatively stable cell-chemistry, large capacity, relatively small dimensions and weight, and larger cycle life and charge rates. The battery is capable of charging at a rate of 15600 mA for 1 C. At 80% charging efficiency at 10 A, the battery would reach full capacity in approximately 1 hour and 11 minutes. Reducing the charge rate to 1 A at about 50% efficiency increases the time to full charge by about 14 hours. Discharge rates are similar, which means that the method used for charging the battery should be able to keep up with about half a day’s worth of battery consumption at about 1 A.

The battery can be charged up to 500 times before full capacity begins to degrade. Dimensions are roughly 20 cm by 7 cm by 1 cm, which can fit a slightly larger windbelt enclosure than the micro enclosure previously specified. The Tenergy battery will be able to provide a 4.2 V nominal signal with a cut-off voltage of 2.75 V at 0.2 C discharge rate and requires 3.7 V or more to accumulate charge.

Image Capture and Processing

Original Idea

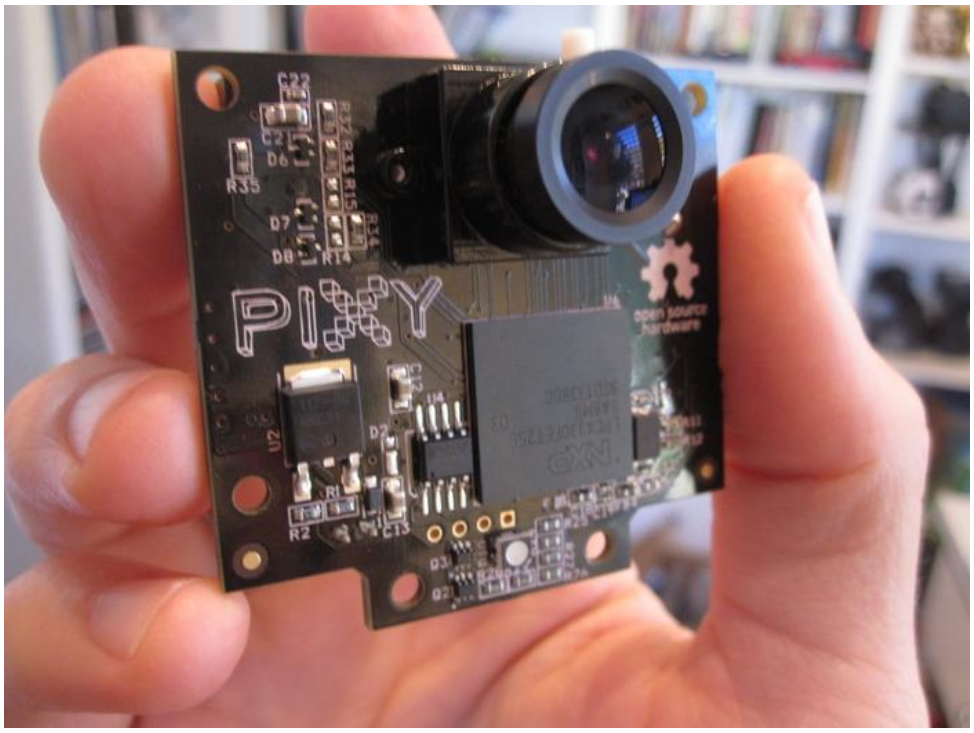


Figure 34 - Pixy Board Image Processing Module

To accomplish the task of capturing and processing images the CMUCam 5, or Pixy, was originally selected. A picture of the Pixy board is shown in Figure 34. Pixy is a sensor module that includes an image sensor and on board image processing. The Pixy utilizes an Omnivision OV9715 image sensor capable of a 1280x800 pixel resolution. A lens with a 70-degree FOV comes standard with the Pixy. This lens may need to be replaced depending on the specifics of the module deployment. The NXP LPC4330 processor is the onboard MCU that is used to perform the onboard image processing. This powerful dual core processor can handle the image processing without too high of a power consumption. The power consumption for the unit averages 140mA. A diagram of the LPC4330's architecture can be seen in Figure 35.

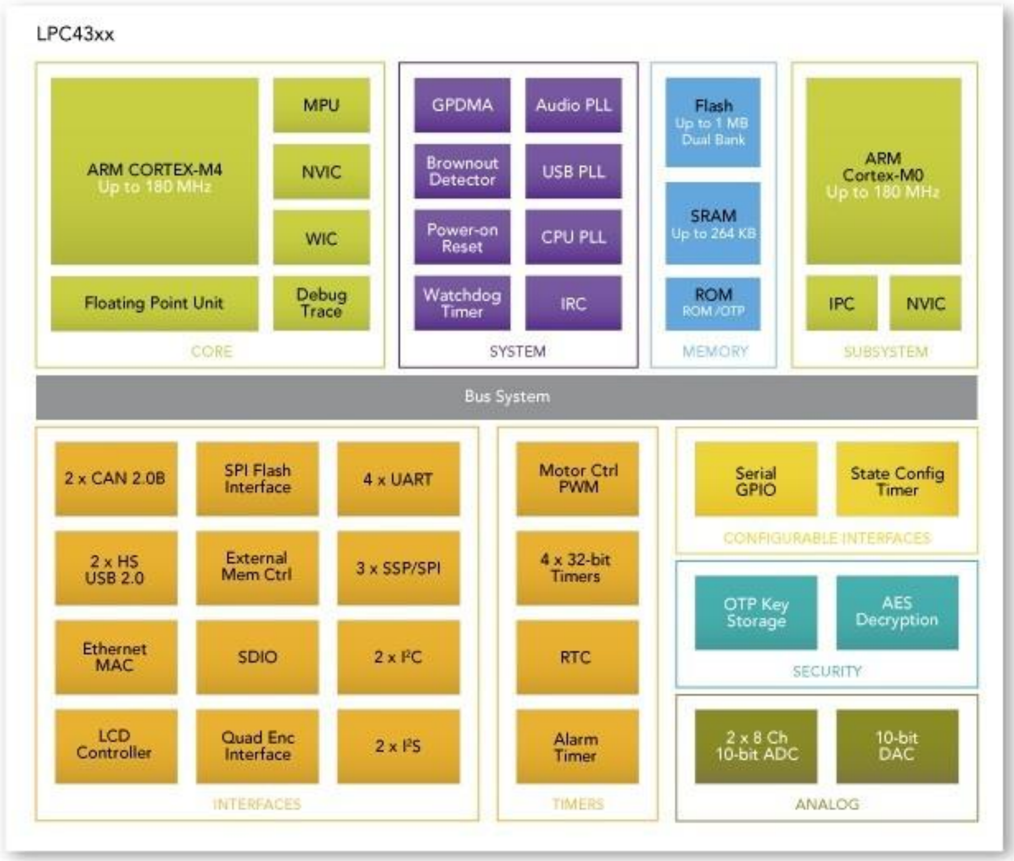


Figure 35 - LPC43XX Architecture Diagram

The NXP LPC4330 MCU that is onboard the Pixy would have been used to control the modules. The processor is a dual core ARM Cortex-M architecture, and includes an M4 and M0 core that can intercommunicate between a common bus system. This processor is fully capable of doing the image processing that is needed and any additional control that may be needed as well. It would simply need to send communications over UART to the XBee modules. The XBee modules operate on their own without any need for control, and the power system also operates without the need for a control system. Since all of the image processing is done onboard, the Pixy simply sends a small amount of statistical data. For example, data that can be sent by the Pixy could be the type of pedestrian that passed in front of the camera. This is accomplished at a rate of 50Hz.

The Pixy also provides a diverse set of communication ports and protocols to interface with a variety of devices, as shown in Figure 36. This will be used to communicate with the XBee modules.

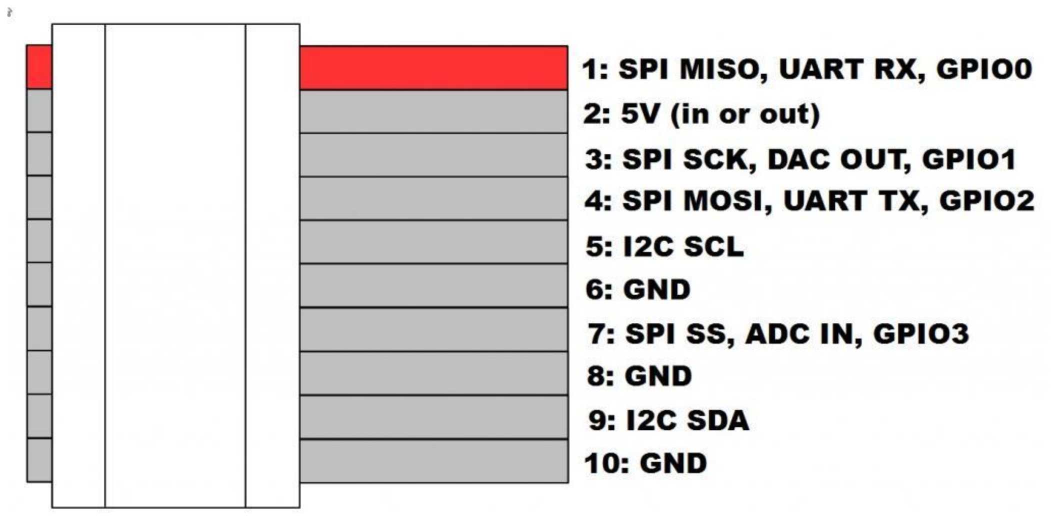

1: SPI MISO, UART RX, GPIO0
2: 5V (in or out)
3: SPI SCK, DAC OUT, GPIO1
4: SPI MOSI, UART TX, GPIO2
5: I2C SCL
6: GND
7: SPI SS, ADC IN, GPIO3
8: GND
9: I2C SDA
10: GND

Figure 36 - Pixy Module Pin-out

Sensor data (e.g. XY coordinates, object speed, etc.) will be sent to the XBees via UART. This data will then be routed through the network to the cloud storage solution.

Currently, the Pixy includes software only for color detection. While new facilities are being added to the Pixy constantly we will probably need to write some application specific software. We will need descriptors for a person walking, riding a bike and riding a horse. We should be able to adapt existing open source software to our purposes.

The Pixy is also ideal for our purposes due to its physical specifications. With dimensions of only 2.1” x 2.0” x 1.4” the module can be hidden in an inconspicuous manner. The light weight of 27 grams also aids in the versatility of its deployment.

However, the Pixy cam had several roadblocks on the way. For one, the onboard memory is too small to hold the CV library we wanted to use. The code was also very difficult to read and to the firmware was tough to alter. A backup plan was chosen for iteration 1.

Backup plan

With the Pixy Cam tabled, the new idea was to use a Raspberry Pi and a Pi Cam to capture the data and send it over to the XBee. This is not intended to be a permanent solution, just as a proof of concept until the Pixy Cam can be figured out. The reason for this is unlike the Pixy Cam, the Pi will draw too much power, and doesn't have enough CPU power to do the CV algorithms within a reasonable time. In addition, the CV library used would not compile for the Pi. Figure 37 shows the Pi Cam, along with a Raspberry Pi attached to an XBee and an infrared sensor.

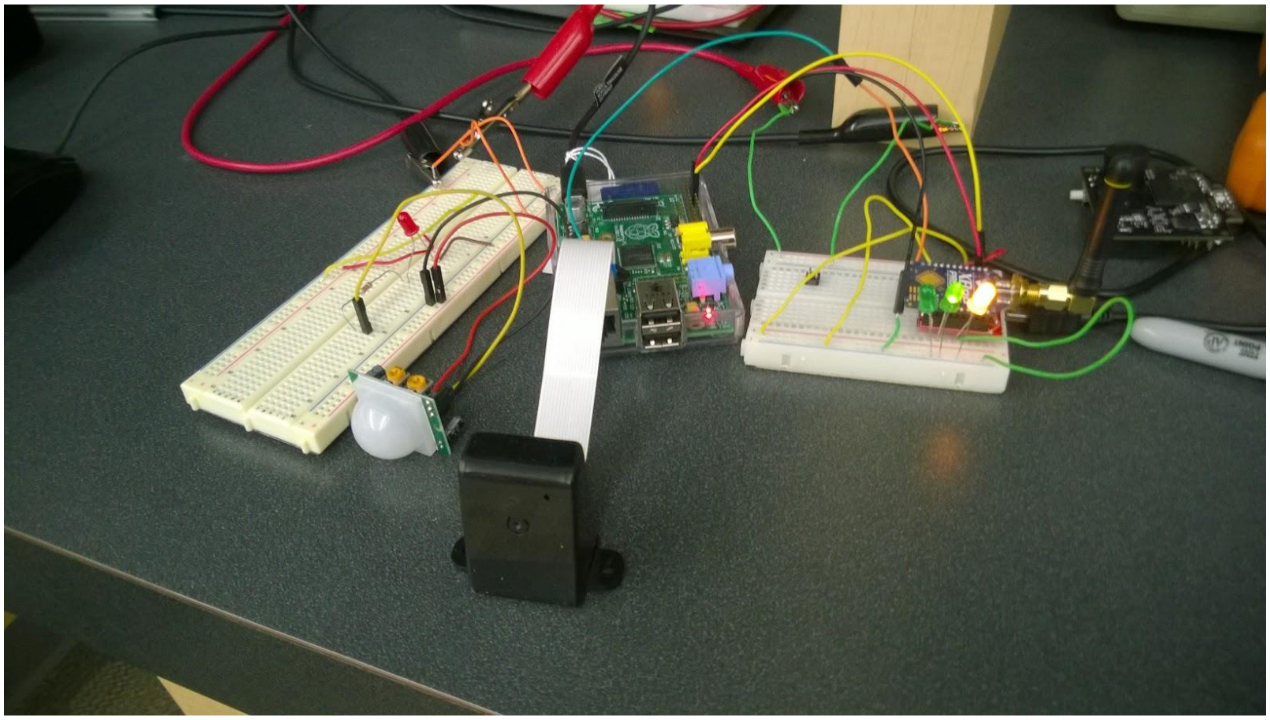


Figure 37 - The Raspberry Pi and the Camera attached.

Since the Pi could not run the CV algorithms on board, the images needed to be sent out to the gateway node, and processed on a computer with more power. Figure 38 shows the process the captured image needed to go through to be processed, and the result written to the database.

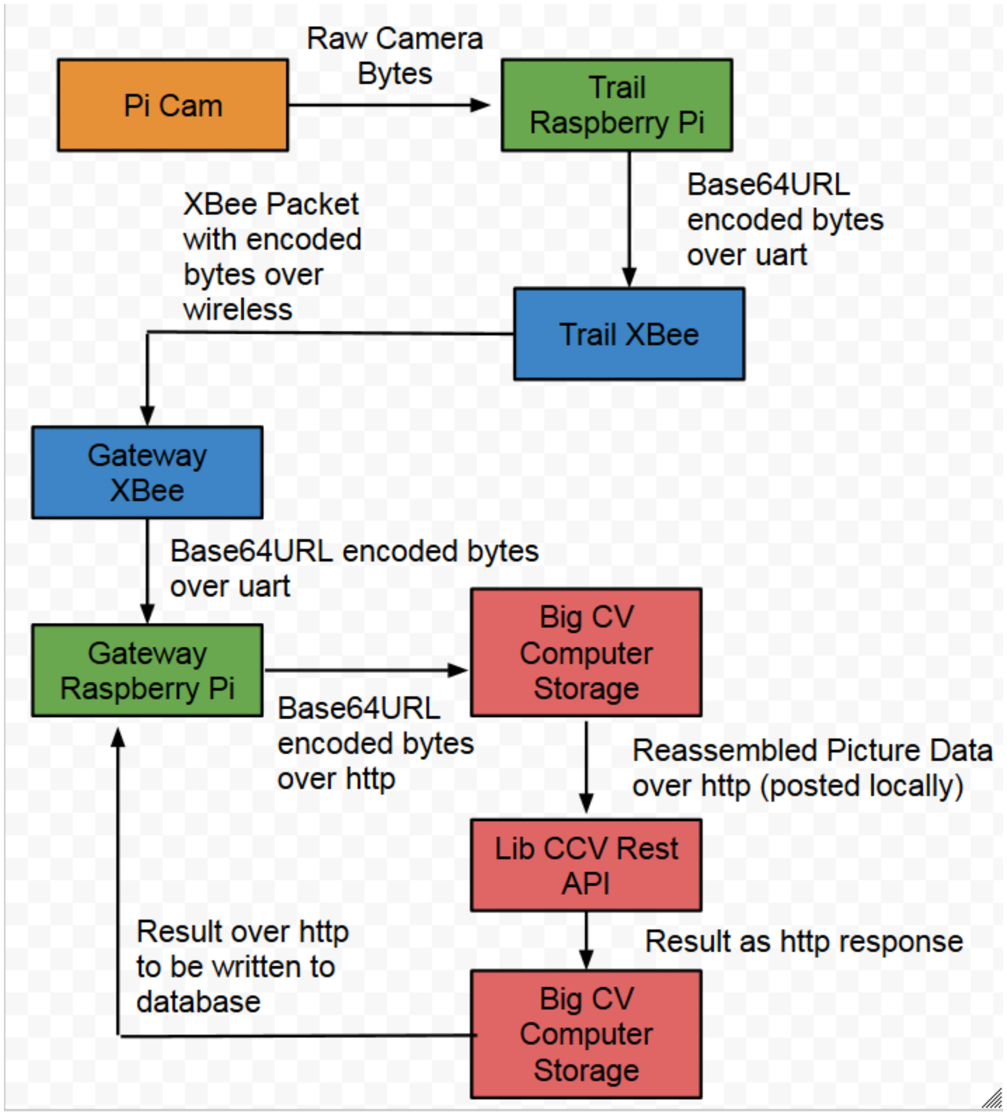

Pi Cam
Raw Camera Bytes
Trail Raspberry Pi
Base64URL encoded bytes over uart
XBee Packet with encoded bytes over wireless
Trail XBee
Gateway XBee
Base64URL encoded bytes over uart
Gateway Raspberry Pi
Big CV Computer Storage
Base64URL encoded bytes over http
Reassembled Picture Data over http (posted locally)
Lib CCV Rest API
Result as http response
Result over http to be written to database
Big CV Computer Storage

Figure 38- The image path

From the Pi Cam, the raw image was saved on the Raspberry Pi. The image needed to be encoded in such a way that the XBee could send it over. Some bytes can cause the XBee to stop sending data, which would be disastrous. The images were encoded to Base64URL, which converts all bytes to either capital or lowercase alphanumeric characters and '-' and '_' characters. All of these characters are safe to send over the XBee. The XBee can also only send 255 characters at a time in its payload. Therefore, the encoded image was send in pieces over the wireless network to the gateway node. The gateway node then sent the data directly to a more powerful computer, which decoded the image data and saved it. Once the entire image was sent over the network, the trail node signaled to the gateway node, who then told the big computer to start the CV algorithm on the reassembled picture. The picture was then send to LibCCV's rest API, over http, also running on the big computer. The CV algorithm would run, and the result would be returned back to the gateway node, who would then update the database accordingly.

Although this process was slow, and a lot different from the original idea, it did work. For future iterations, the CV must be done on the trail nodes directly. Sending that much data over XBee is not ideal and very unstable. However, this does show that if the XBee protocol is followed, anything that can connect to an XBee can become a trail node.

Infrared Sensor

Both the Pi and the Pixy cam has one weakness. When one is powered, it will be consistently be filming the trail, even if no one is walking past. This will waste power as the camera is active and running image processing on nothing. To help mitigate this, an infrared sensor can be put in place that will “wake up” the camera whenever someone walks past the trail.

The infrared sensor that was chosen was the Zilog ZMotion Detection Module (Z8FS040), which is shown below in Figure 39.

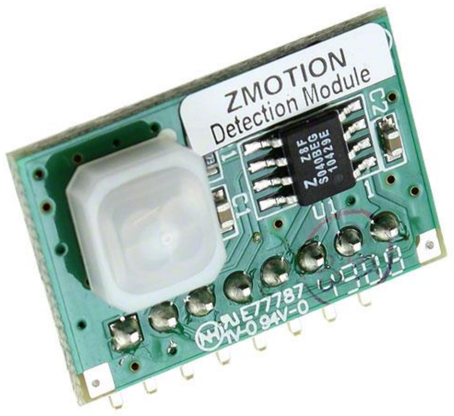


Figure 39 - The ZMotion Detection Module

The ZMotion module comes with a variety of pins that can be used. The pinout is below in Figure 40.

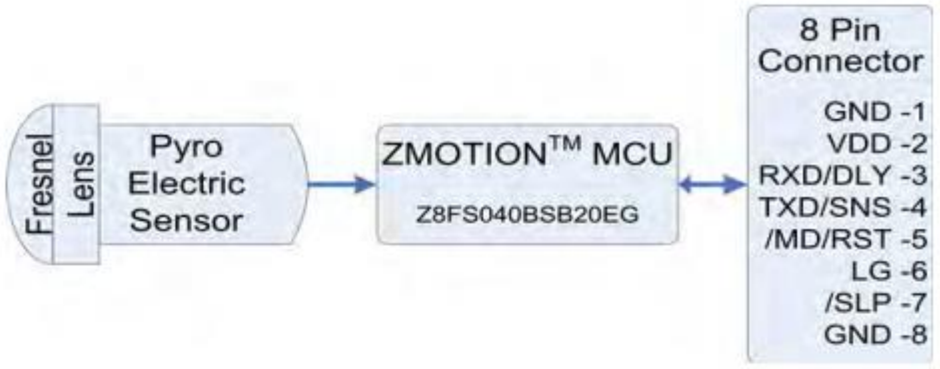


Figure 40 - ZMotion pinout

The ZMotion module can operated in two different modes: hardware interface and serial interface. For the purposes of this project, the ZMotion module is used in hardware interface mode. When in hardware interface mode, pin 5 of the ZMotion module will be set to logic ‘0’ when motion is detected. This pin can be tied to one of the MCU’s GPIO ports, and the camera will be woken up via an interrupt whenever the logic of the pin is ‘0’. When the camera is woken up, it can do image processing before going back to sleep.

Another advantage of the ZMotion is that is has a low-power mode, which is activated whenever pin 7 is low. During this time, the motion sensor will not detect any motion, as it is basically asleep. The pin that controls low power mode can be connected to the XBee’s “module status pin,” which is set to low whenever the XBee is asleep. At night, when the gateway node tells all of the XBees to go to sleep, not only will the XBees go to sleep, but the infrared sensor will also go to sleep. When the infrared sensor goes to sleep, the camera will stay asleep. This technique reduces the total power consumption during the night, when the camera’s vision is naturally limited.

Network Architecture

Digi XBee RF modules will be used to coordinate data among the sensor nodes, utilizing the 900 MHz experimental band along with 2-3 dB gain antennas to intercommunicate. An approximate RF range of 3 miles is necessary to allow overlap of each node with at least two other nodes along the length of the trail.

The XBee name refers to a family of form factor compatible radio modules that utilize the 802.15.4 network protocol. A network of this IEEE standard specifies the physical layer and access control for low-rate LR-WPANs, as shown in Figure 41.

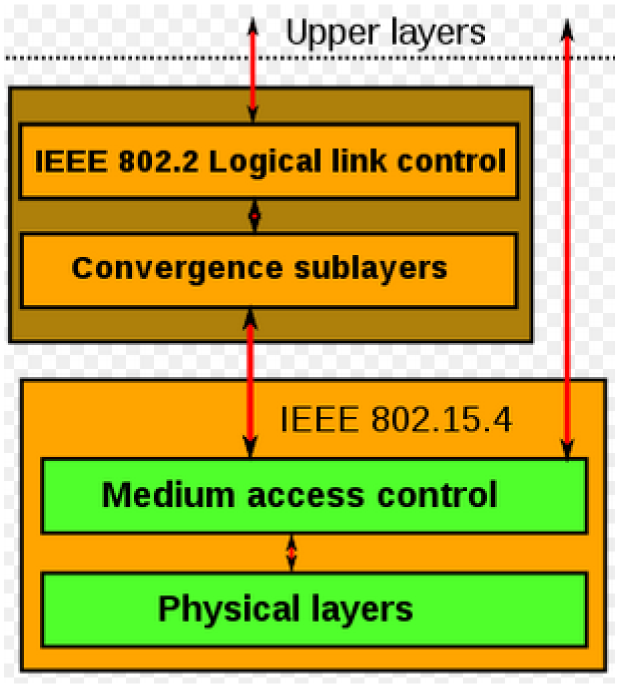


Figure 41 - IEEE 802.15.4 Protocol Stack

The protocol architecture is conceptually simple, based on the OSI model. Only the two layers shown in Figure 41 are defined by the protocol. In North America, the physical layer utilizes the 902 to 928 MHz band, including up to 30 channels. Data rates range from 100 kbps at 868 MHz to 250 kbps at 915 MHz.

The MAC layer allows for network beaconing, controlling frame validation, guaranteeing time slots and handling node associations. Frame size is limited to 127 bytes for most

802.15.4 applications, and fragmentation schemes are available to support larger network layer packets.

The ZigBee protocol makes up the remainder of the stack, adding four main components: the network layer, application layer, ZigBee device objects (ZDOs) and manufacturer-defined application objects. The application objects allow for customization and complete integration of the network. ZigBee allows for low power consumption and secure networking in personal area networks. A small form factor is characteristic of modules that use the protocol, along with the ability to transmit data over long distances while passing data through intermediate devices. This is accomplished while attempting to reach more distant nodes, creating a highly reliable mesh network topology and eliminating the need for a centralized control for certain applications. ZigBee devices utilize 128-bit symmetric encryption for security purposes.

The Digi XBee-Pro 900HP S3B is the specific 900 MHz module that will be used, coupled with a 2.1-dB antenna. The radio can operate with a range of up to 9 miles provided that LOS (line of sight) should not be an issue. This will allow for creation of a true mesh network, since all project nodes are within a 15-mile range limit of each other along the trail. The radio utilizes an ADF7023 transceiver with a Cortex-M3 operating at 2.8 MHz. It can transmit at the 900 MHz experimental band that is desired, and transfers at 10 Kbps worst case range. Speeds of up to 200 Kbps are possible at minimum range. The most efficient transmittable range has been proven to be 4 miles with 2.1-dB dipole antennas. The radio will be connected to the MCU via UART, and operates at 2.1 to 3.6 VDC with a transmit current of 215 mA, a receive current of 29 mA, and a sleep current of 2.5 µA.

Computer Vision Algorithm

There were several potential options for computer vision (CV) libraries that we could use for the project. The major feature that we were looking for was a lack of further library dependencies that many libraries required. Also many of the CV libraries required OS level services that would not be reasonable to have running on the PixyCam. For these reasons we chose to use a library called LibCCV. LibCCV can be compiles with zero dependencies and has a very small footprint for a CV library.

Initially the plan was to utilize the Scale Invariant Feature Transform (SIFT) algorithm to run directly on the PixyCam. SIFT would have provided the ability to create feature descriptors for objects that appeared.

The SIFT algorithm provides the two major CV processes that we need to accomplish our task: feature detection and description. As its namesake states SIFT selects features that are scale invariant. Changes in scale can be a serious issue when on is attempting object detection. By selecting features of this nature SIFT performs well under images that exhibit both rotation and scale changes.

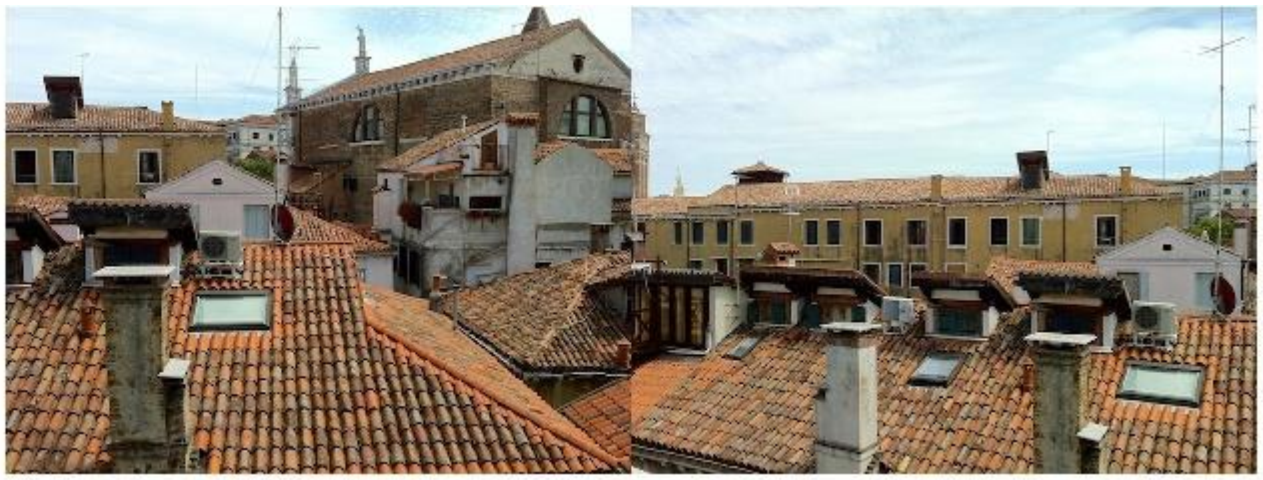


Figure 42 - SIFT Test Images

Figure 42 is two test images that were run through the SIFT program. You can see below in Figure 43 how SIFT locates the same locations in each of the images below.

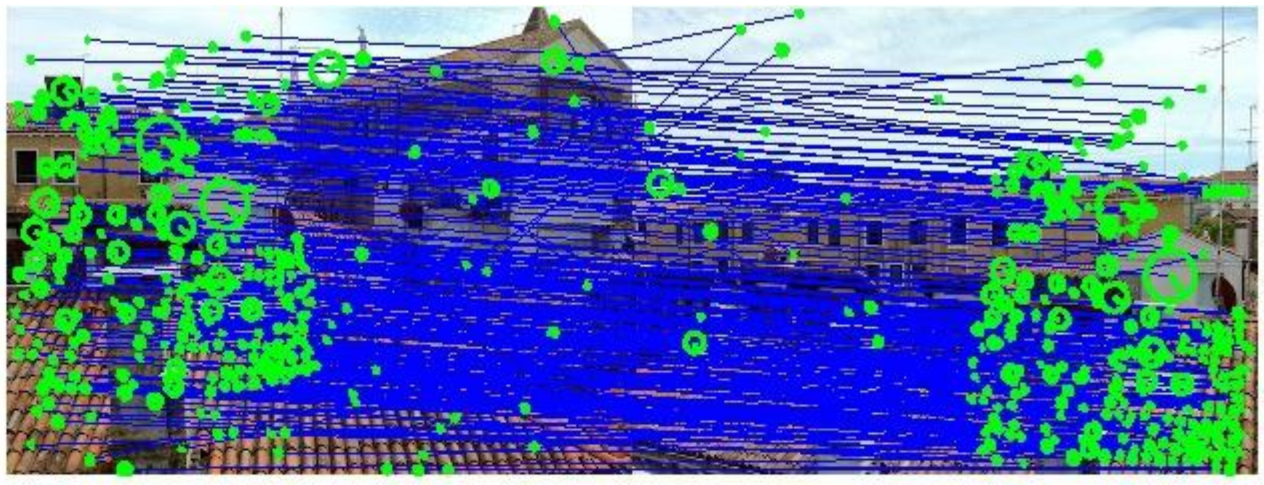


Figure 43 - SIFT in action

A couple of false negatives can be seen but this is a very small percentage of the matches.

These feature descriptors are relatively small data that could easily be sent across our XBee network. The feature descriptors would be used to track objects across other node locations & be used to identify the kind of object that was being tracked. These feature descriptors would be ultimately passed through the gateway node to a powerful desktop/server for the higher level CV processing; object detection, cross network tracking.

The memory available on the PixyCam proved to be too small for us to flash the compiled code to. As such a change in hardware and process was required. As described above, a RaspberryPi and PiCam were chosen to implement the image capture system. The captured image was then serialized and sent to a CV server.

An additional reason that LibCCV was chosen was because it offered a REST Http interface by which we could interface with the CV server. This allowed us to continue to use the current library with the change in configuration.

As we were putting all of the CV load onto the CV server with the image processing we choose a different CV data flow and used the Integral Channel Features (ICF) algorithm. ICF is a very strong (accurate) rigid object detector that LibCCV includes a training set for pedestrian detection.

ICF analyzes the input image on a number of image channels. These channels are typically linear and non-linear transformations, such as edge detection. Features are then selected depending on local rectangular sums. By utilizing this simple but diverse method, information rich features are produced in a computationally efficient manner.

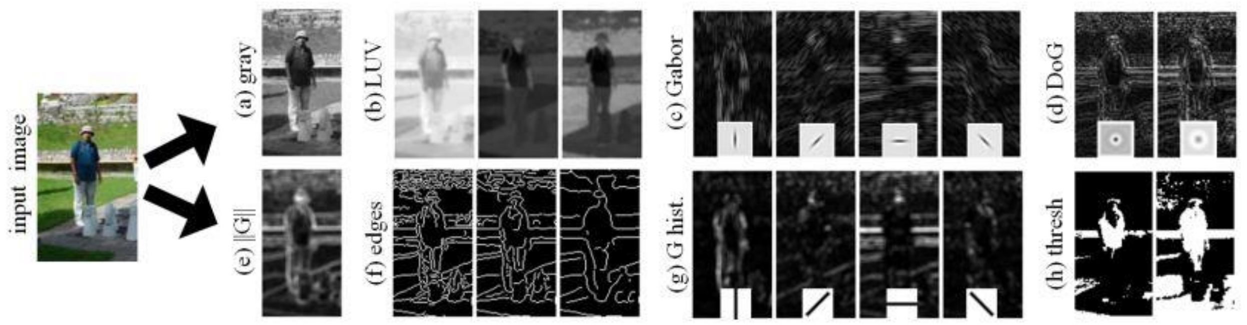


Figure 44 - ICF channel configuration example

Figure 44 describes a potential channel configuration. Several image classification channels are running simultaneously. This level of information density is part of what makes ICF as strong as a detector.

This configuration allows us to fully put the CV responsibilities onto the CV server. Because of this we can potentially integrate any image capture system into our system. This really opens up the potential areas of application and integration into existing systems.

Enclosure

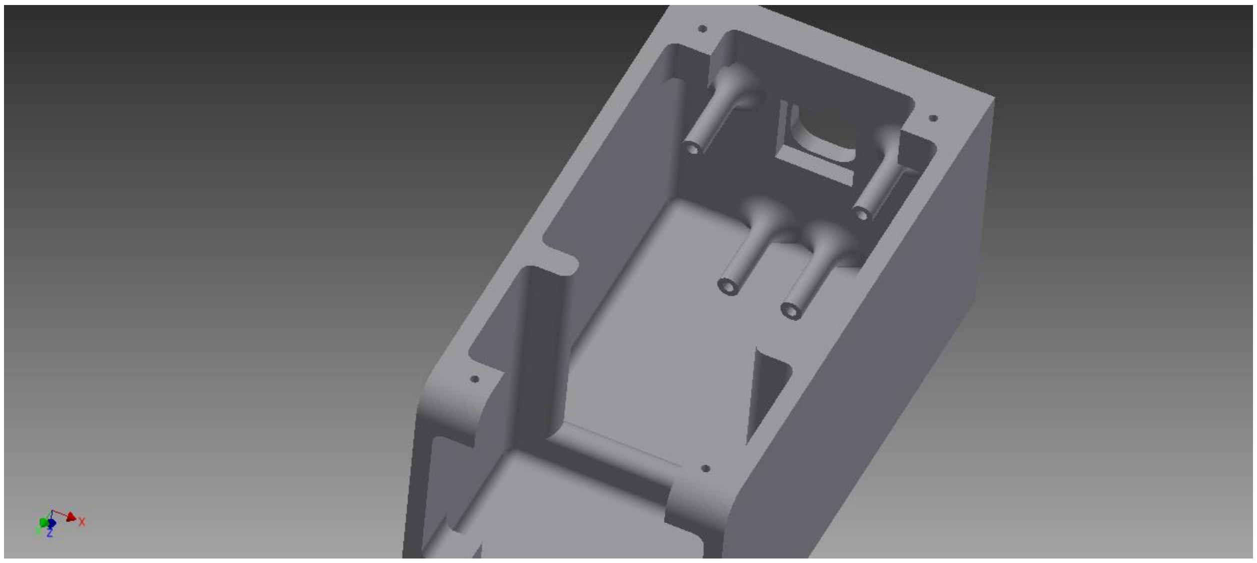


Figure 45 - Enclosure Overview

The enclosure system had several requirements for our project. As the nodes were to be outside for a prolonged period of time the enclosure had to be robust and able to withstand a variety of conditions. Because of these requirements we decided that a custom enclosure design would be needed to suit our needs.

Using the AutoDesk AutoCAD Inventor software package a design for the enclosure was produced and iterated on, as shown in figures 45 and 46. A post system was utilized to offset the PixyCam sensor while it was securely fixed to the enclosure. A rear compartment was created both to house the large and bulky battery unit but also to add rigidity to the design of the enclosure. All interior edges were filleted to provide additional rigidity and strength. Without the filleting, forces would be focused into the edge where the two faces meet. This would encourage cracking and ultimately the failure of the enclosure.

A hole was kept in the front of the enclosure for the image sensor to “look” through. This hole has an inner lip that a piece of clear Lexan would be epoxied to. The top of the enclosure adheres to the main body of the enclosure with a simple screw system. An additional sealing agent of some form would need to be added to the surfaces that meet between the two pieces to ensure waterproofness.

The enclosure was produced using an additive manufacturing process; 3D printing. The 3D printer utilized a filament extruding process. We specifically used PLA as the printing material. It was approximately a 14hour process to print the enclosure.
The enclosure ended up being very rigid and strong despite also being very light; ~250 grams.

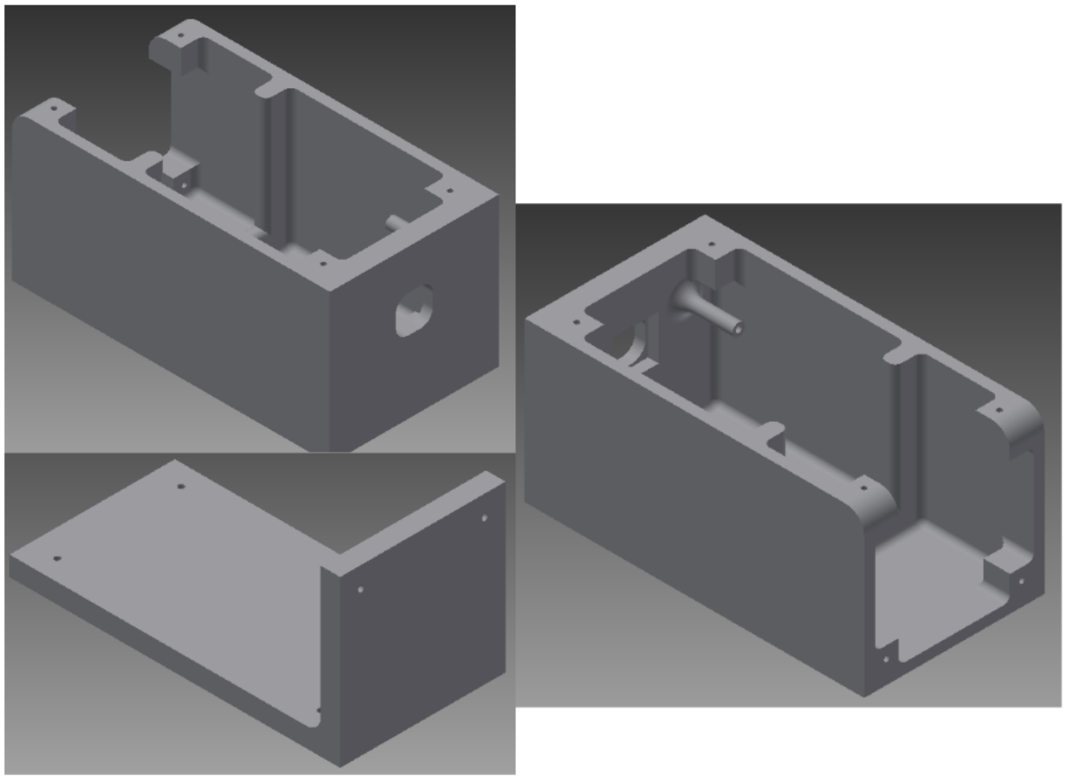


Figure 46 - Enclosure from Different Views

Figure 46 gives several views of the enclosure. The method by which the top would be affixed to the body of the enclosure can also be seen. Ultimately the entire system was able to be fit into a small enclosure with the approximate dimensions of 6”X3”X2.5”. We were happy to be able to fit both the image capture system, power system and power source (battery) in a relatively small package.

Cloud Server and Gateway Node

The Digi XBee 900 HP allows communication from each of the trail nodes, but there needs to be some way to save the data to a web-accessible database through some kind of gateway node. This can be achieved by using a Raspberry Pi. The Pi can be used to both read data from the trail nodes, write these data to a database, and then serve web pages using the data from the database to users from the internet. Figure 47 shows a block diagram on how this will work.

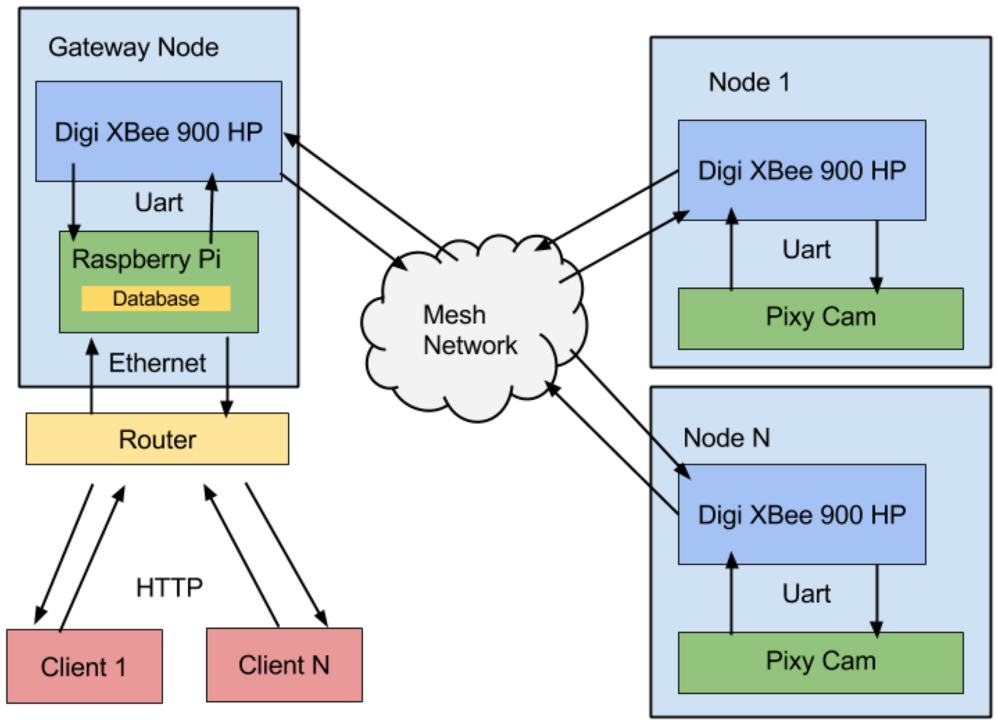


Figure 47 - Network Communication Diagram

Each of the trail nodes have the MCU connected to the Digi XBee module via UART. The XBees, meanwhile, form a mesh network. Meanwhile, the gateway node has the Raspberry Pi also connected to a Digi XBee through the Pi's UART interface. If one of the trail nodes needs information to be sent to the database, it communicates with the Raspberry Pi, which will then write the information to the database that is in the Pi's memory. The Pi will also act as the web server as well. The Pi has a built-in Ethernet jack that is used to connect with the World-Wide-Web. The Pi runs the server software

Apache and the Python web framework Django, to handle http requests and send web pages with data from the database to clients.

There is a “Status” webpage hosted on another Raspberry Pi at Seth’s apartment. Every minute, an HTTP get request is sent to the gateway node to get the server status. If the server returns “Ok,” the status page will display that everything is up and running. If the server returns “maintenance,” the status page will display that the server is undergoing maintenance. If a 404 is returned, then the server is down, and the status page will display “down.” The purpose of this is so if a user cannot access the server, they can go to this status page and see if there is maintenance going on, or if the site is down completely.

Gateway Software

There are several pieces of software running on the gateway node to ensure all the data gets moved around correctly. Figure 48 shows a block diagram of all the processes running on the gateway node.

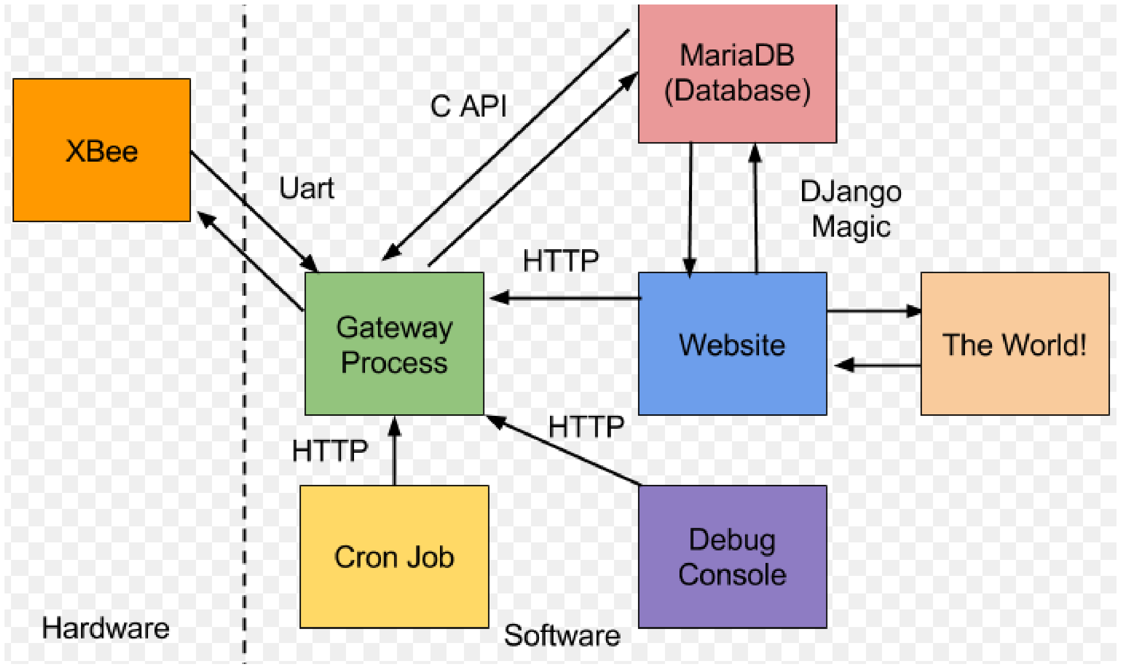
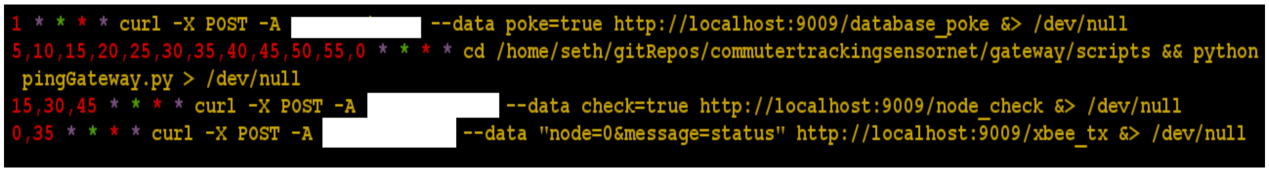


Figure 48 - Gateway Software

On the Raspberry Pi Gateway Node, there is a custom piece of software that handles getting everything to communicate with each other (labeled “Gateway Process” in figure 48). This includes getting the incoming data from the XBee to be written to the database, sending out messages to the trail nodes, and sending emails to admins in case something bad happens. This process runs a web server using the C++ Poco Library. This allows other processes, including itself, to send commands to it via HTTP.

The gateway process will read from and write to the database through a C API. Some examples of data read from the database include admin emails and phone numbers, and the nodes on the trail. Examples of data the gateway process writes to the database includes any entities found on the trail and any error messages that come from the trail nodes on the trail.

Sometimes, commands need to be run every few minutes, or at a certain time. One example of this is every 15 minutes, the gateway process sends a command to each of the trail nodes asking for their status. This is done via cron jobs. A cron job is a command that is run at a certain time. The command can be run minutely, hourly, monthly, etc. Figure 49 below shows the gateway's current cron configuration.



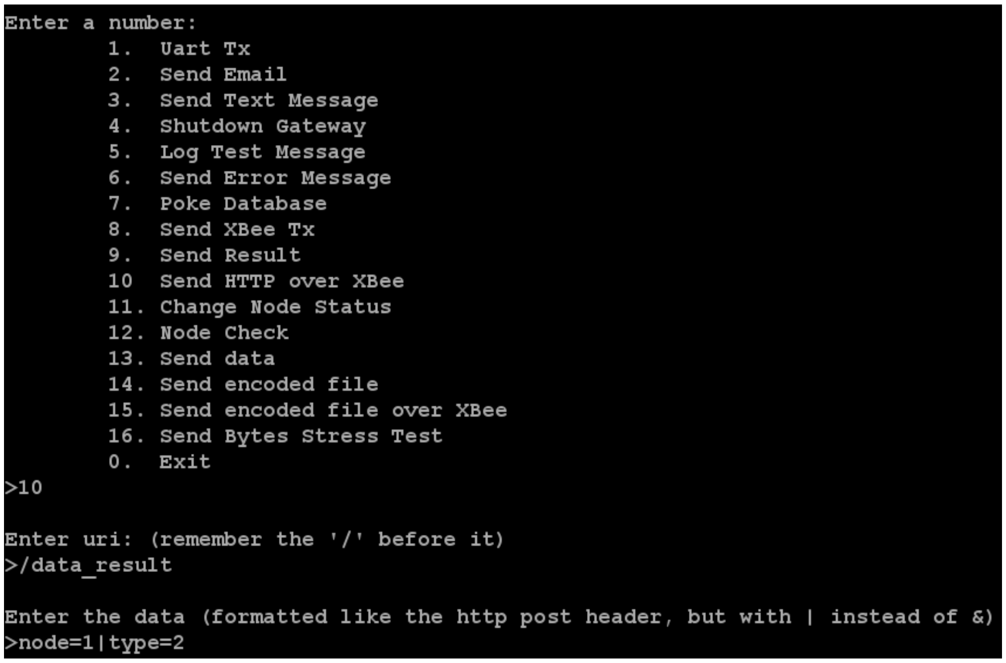
```
1 * * * * curl -X POST -A [redacted] --data poke=true http://localhost:9009/database_poke &> /dev/null
5,10,15,20,25,30,35,40,45,50,55,0 * * * * cd /home/seth/gitRepos/commutertrackingsensornet/gateway/scripts && python
 pingGateway.py > /dev/null
15,30,45 * * * * curl -X POST -A [redacted] --data check=true http://localhost:9009/node_check &> /dev/null
0,35 * * * * curl -X POST -A [redacted] --data "node=0&message=status" http://localhost:9009/xbee_tx &> /dev/null
```

Figure 49 - Gateway Node's Crontab

This figure shows the gateway node's crontab. Some sensitive information is whited out. The first command is set to run every hour when the minute reaches 1 (so 12:01, 1:01, 2:01, etc.), and it tells the gateway process to talk to the database, otherwise the connection will time out. It does this by sending an http post request via curl to the gateway process, which runs on port 9009. The second command runs every five minutes. It runs a script that checks to see if the gateway process is still running, and if it's not, it writes to the database that the process is down, which is later displayed on the website. The third command runs every hour at minute 15, 30, and 45. This sends an http post request via curl that tells the gateway process to look through the database for any nodes that did not update their status within an hour. The gateway process will then update the database accordingly, and send any emails to the admins. The last command runs on every hour, and on every 35 minute mark. It sends a command to all trail nodes (node 0 is a special node that means messages get sent to all nodes in the network) to send the gateway node their status.

In its current form, the website does not directly talk to the gateway process, contrary to what Figure 48 implies. The idea was to allow admin users to power cycle a node from the web page, which requires the website to post to the gateway process. This did not make it in this iteration of the project.

The debug console is mainly used for testing purposes. When the Raspberry Pi is started up, the gateway process runs automatically. This means that a standard user cannot access this process to send commands over stdin. Therefore, the debug console was written so users can send commands to the Gateway node over http. Figure 50 shows a picture of the simple debug console, along with all the features it can do.



```
Enter a number:
        1.  Uart Tx
        2.  Send Email
        3.  Send Text Message
        4.  Shutdown Gateway
        5.  Log Test Message
        6.  Send Error Message
        7.  Poke Database
        8.  Send XBee Tx
        9.  Send Result
        10  Send HTTP over XBee
        11. Change Node Status
        12. Node Check
        13. Send data
        14. Send encoded file
        15. Send encoded file over XBee
        16. Send Bytes Stress Test
        0.  Exit
>10

Enter uri: (remember the '/' before it)
>/data_result

Enter the data (formatted like the http post header, but with | instead of &)
>node=1|type=2
```

Figure 50: Debug Console

The debug console allows for users to send commands for test purposes to the gateway process. In the example above, the gateway process will send an http command out over XBee. In this case, it is sending to the gateway node a message that node 1 detected a pedestrian.

The gateway process also communicates with the XBee over UART. When it needs to send a message out over XBee, it formats the packet and sends the packet out over UART. Upon receiving a packet, an interrupt occurs that tells the process data is ready to be read off of UART, and the data is added to a queue for processing. The payload from the packet consists of two parts. The first part is an URI to post to, and the second part is the data to send. Basically, the gateway node is sending an http post request to itself, and then the command gets executed. For example, if a trail node wants to update its status to “okay”, it will send the packet:

```
/node_status\tnode=2|status=1
```

The gateway process will then separate the URI to post to and the data from the ‘\t’ (tab) character. It will then convert all ‘|’ characters in the data section to an ‘&’ character, as that is what HTTP requires. The reason why an ‘&’ character was not sent over XBee in the first place is because for some reason, ‘&’ stops the XBee from completing the message. The gateway process then calls the following command:

```
curl -X POST -A thisIsASecret --data ”node=2&status=1” http://localhost:9009/node_status
```

The gateway process will then treat the command exactly like it received it from a cron job or the debug console. The sequence diagram in figure 51 shows this process.

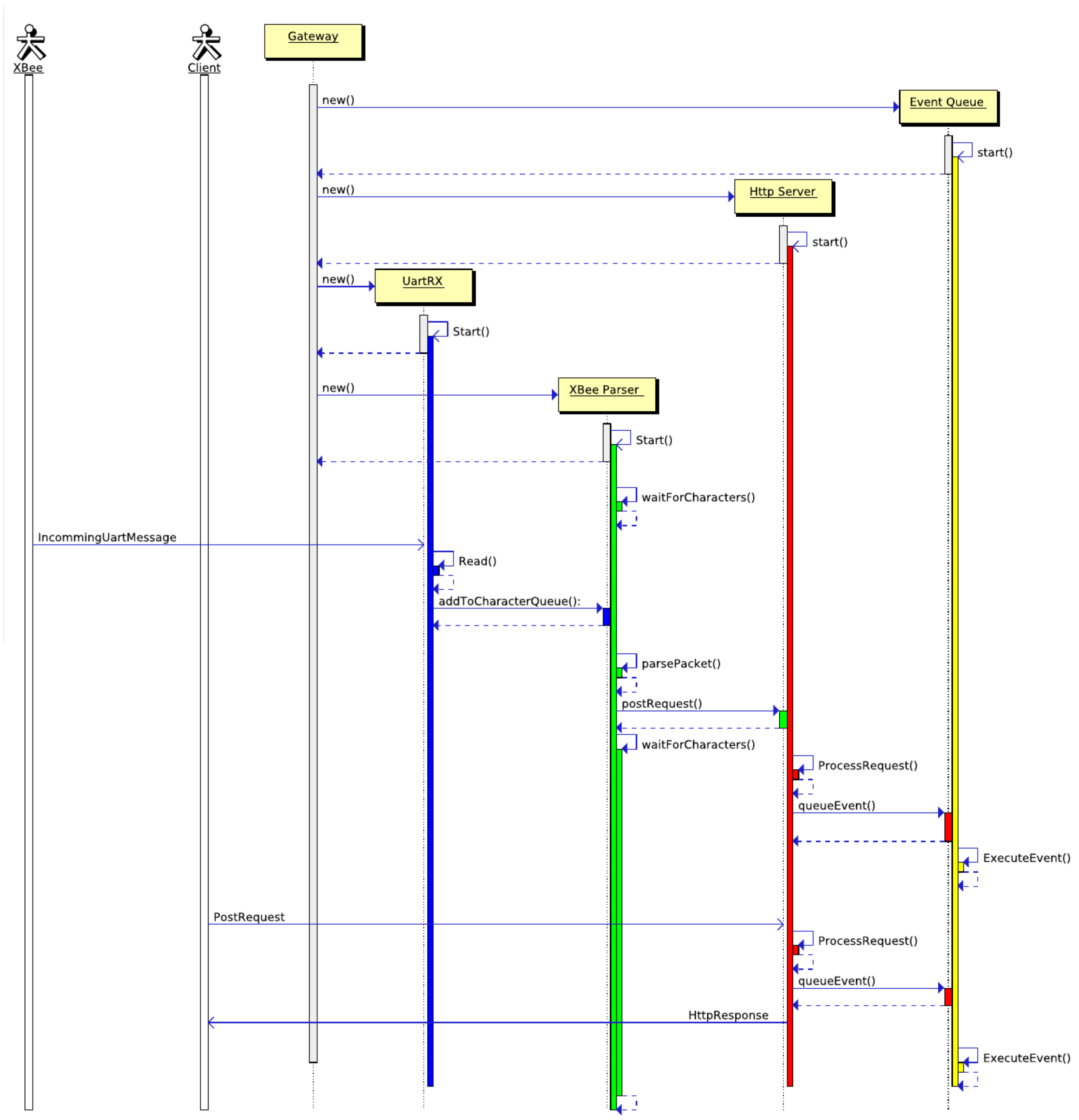


Figure 51- Gateway Process Sequence Diagram

The Gateway Process is made up of 5 threads. The first thread, labeled Gateway, is the main thread. Its job is to allocate and start everything, wait for the other threads to exit, and clean everything up. UartRX's only job is to wait until an interrupt occurs, and then it reads characters from UART, and sends them to the XBee Parser thread, and continues to do this until no characters are coming from the UART. The XBee parser takes the characters from the UART and parses them to get the payload. It will then *perform an HTTP Post request based on the payload data to the HTTP Server thread. The HTTP Server thread will then ensure the data is correct, and if it is, create an event based on the data and add it to the event queue thread. The event queue will then execute the event. In the event that a client (such as a cron job) posts directly to the http server, the same process happens, but nothing occurs with the XBee parser or UART threads.*

Database

The database is implemented using MariaDB, which is a drop-and-replace of MySQL. There are three main parts of the database. The first part is stuff that Django automatically creates for it to function. This is created automatically when Django is synced with the database. The schema for it can seen below in Figure 52**.** The only table that was created custom was the ctsn_user table, which stores data the Django user table does not have, but must be tied to a user, such as a phone number.

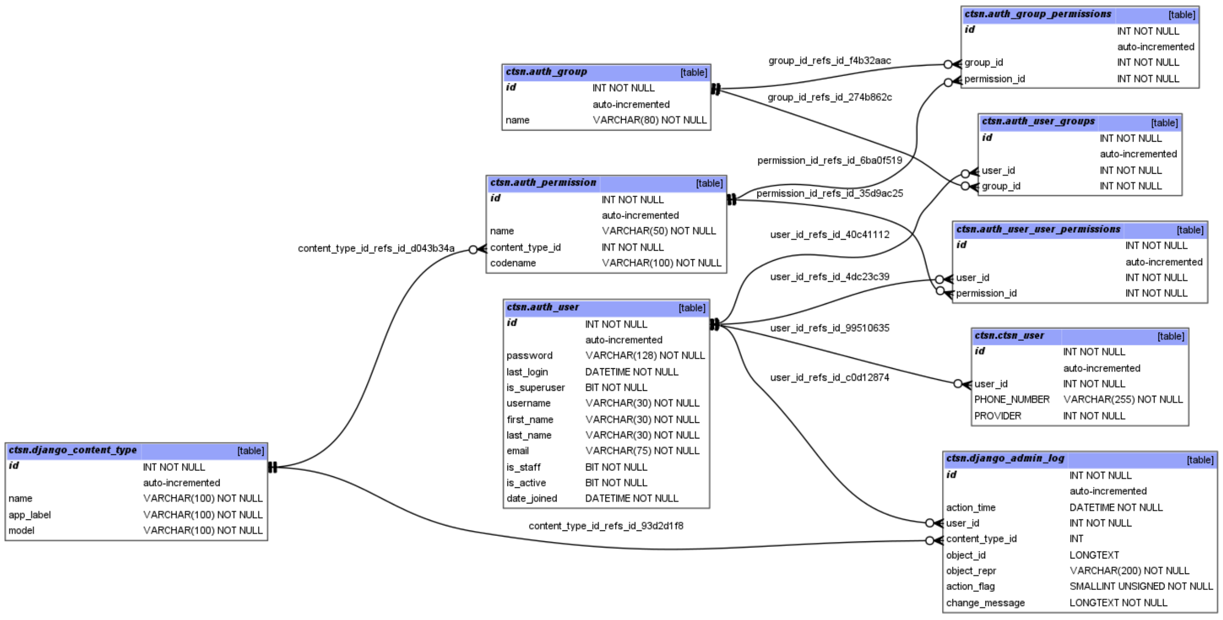

ctsn.auth_group_permissions [table]
id INT NOT NULL auto-incremented
group_id INT NOT NULL
permission_id INT NOT NULL
ctsn.auth_group [table]
id INT NOT NULL auto-incremented
name VARCHAR(80) NOT NULL
group_id_refs_id_f4b32aac
group_id_refs_id_274b862c
ctsn.auth_user_groups [table]
id INT NOT NULL auto-incremented
user_id INT NOT NULL
group_id INT NOT NULL
permission_id_refs_id_6ba0f519
permission_id_refs_id_35d9ac25
ctsn.auth_permission [table]
id INT NOT NULL auto-incremented
name VARCHAR(50) NOT NULL
content_type_id INT NOT NULL
codename VARCHAR(100) NOT NULL
content_type_id_refs_id_d043b34a
ctsn.auth_user_user_permissions [table]
id INT NOT NULL auto-incremented
user_id INT NOT NULL
permission_id INT NOT NULL
user_id_refs_id_40c41112
user_id_refs_id_4dc23c39
ctsn.auth_user [table]
id INT NOT NULL auto-incremented
password VARCHAR(128) NOT NULL
last_login DATETIME NOT NULL
is_superuser BIT NOT NULL
username VARCHAR(30) NOT NULL
first_name VARCHAR(30) NOT NULL
last_name VARCHAR(30) NOT NULL
email VARCHAR(75) NOT NULL
is_staff BIT NOT NULL
is_active BIT NOT NULL
date_joined DATETIME NOT NULL
user_id_refs_id_99510635
user_id_refs_id_c0d12874
ctsn.ctsn_user [table]
id INT NOT NULL auto-incremented
user_id INT NOT NULL
PHONE_NUMBER VARCHAR(255) NOT NULL
PROVIDER INT NOT NULL
ctsn.django_content_type [table]
id INT NOT NULL auto-incremented
name VARCHAR(100) NOT NULL
app_label VARCHAR(100) NOT NULL
model VARCHAR(100) NOT NULL
content_type_id_refs_id_93d2d1f8
ctsn.django_admin_log [table]
id INT NOT NULL auto-incremented
action_time DATETIME NOT NULL
user_id INT NOT NULL
content_type_id INT
object_id LONGTEXT
object_repr VARCHAR(200) NOT NULL
action_flag SMALLINT UNSIGNED NOT NULL
change_message LONGTEXT NOT NULL

Figure 52 - Django Database Schema

The second part of the database was for Django plugins. Plugins were added to the Django framework for security purposes. Some plugins include a CAPTCHA so bots can't brute force their way in, a plugin that edits robots.txt to block incoming bots, and a way to block an IP address after a certain number of failed login attempts. These database tables were also created automatically, and are shown in figure 53**.**

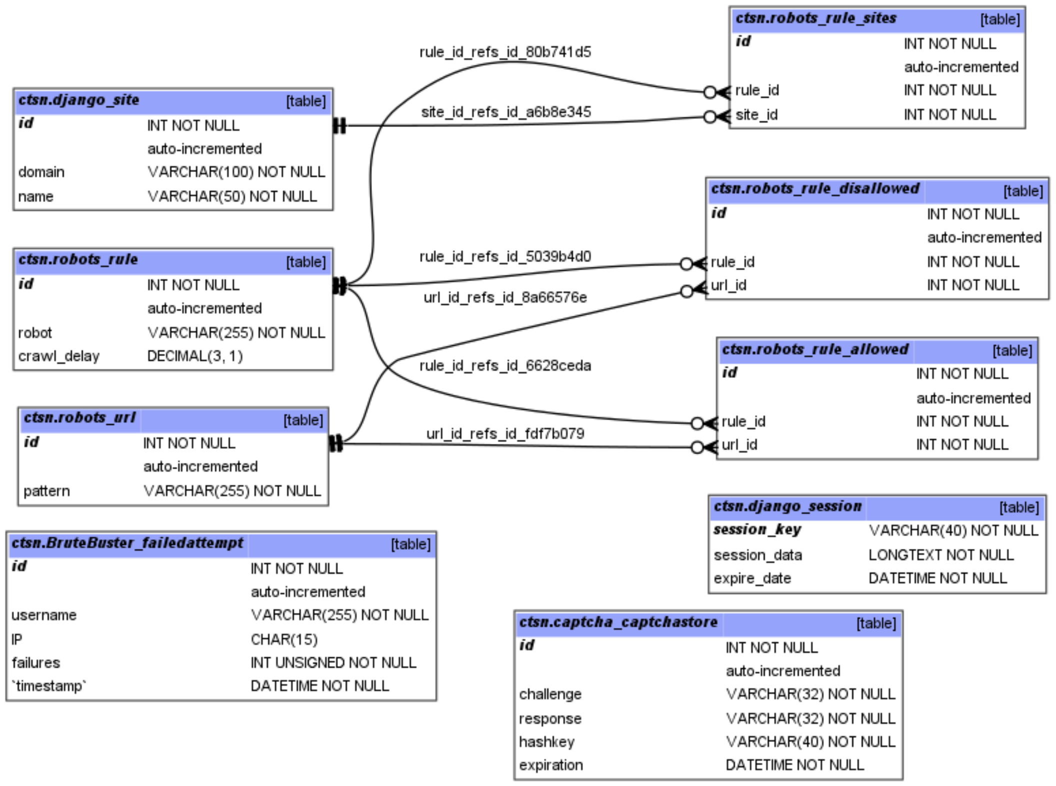

ctsn.robots_rule_sites [table]
id INT NOT NULL auto-incremented
rule_id INT NOT NULL
site_id INT NOT NULL
rule_id_refs_id_80b741d5
site_id_refs_id_a6b8e345
ctsn.django_site [table]
id INT NOT NULL auto-incremented
domain VARCHAR(100) NOT NULL
name VARCHAR(50) NOT NULL
ctsn.robots_rule_disallowed [table]
id INT NOT NULL auto-incremented
rule_id INT NOT NULL
url_id INT NOT NULL
rule_id_refs_id_5039b4d0
url_id_refs_id_8a66576e
ctsn.robots_rule [table]
id INT NOT NULL auto-incremented
robot VARCHAR(255) NOT NULL
crawl_delay DECIMAL(3, 1)
ctsn.robots_rule_allowed [table]
id INT NOT NULL auto-incremented
rule_id INT NOT NULL
url_id INT NOT NULL
rule_id_refs_id_6628ceda
url_id_refs_id_fdf7b079
ctsn.robots_url [table]
id INT NOT NULL auto-incremented
pattern VARCHAR(255) NOT NULL
ctsn.django_session [table]
session_key VARCHAR(40) NOT NULL
session_data LONGTEXT NOT NULL
expire_date DATETIME NOT NULL
ctsn.BruteBuster_failedattempt [table]
id INT NOT NULL auto-incremented
username VARCHAR(255) NOT NULL
IP CHAR(15)
failures INT UNSIGNED NOT NULL
`timestamp` DATETIME NOT NULL
ctsn.captcha_captchastore [table]
id INT NOT NULL auto-incremented
challenge VARCHAR(32) NOT NULL
response VARCHAR(32) NOT NULL
hashkey VARCHAR(40) NOT NULL
expiration DATETIME NOT NULL

Figure 53- Django Plugin Database Schema

The last piece of the database were tables that handle the trail data. The schema is shown below in figure 54, and what each table represents is in table 4.

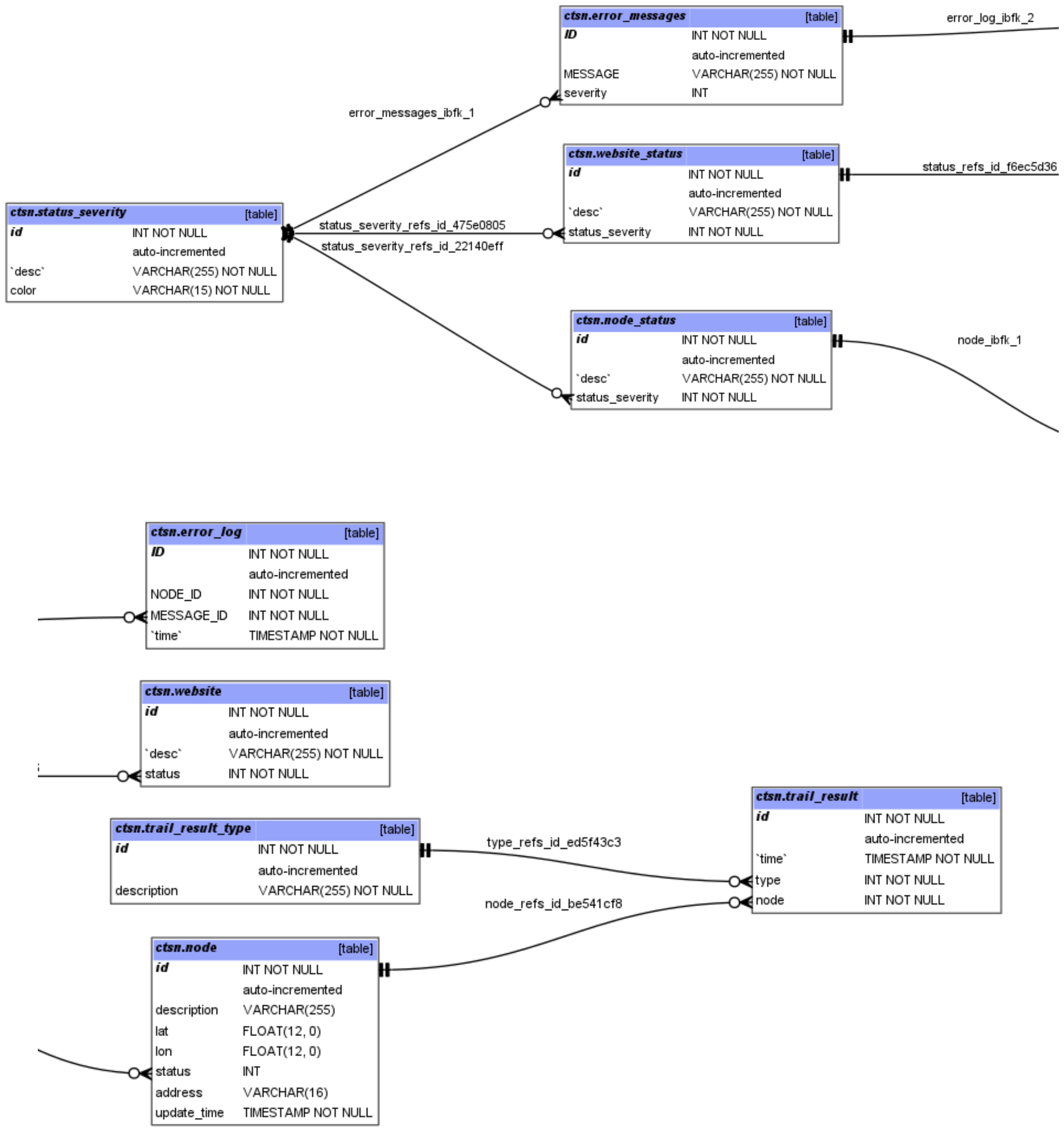

ctsn.error_messages [table]
ID INT NOT NULL auto-incremented
MESSAGE VARCHAR(255) NOT NULL
severity INT
error_log_ibfk_2
error_messages_ibfk_1
ctsn.website_status [table]
id INT NOT NULL auto-incremented
`desc` VARCHAR(255) NOT NULL
status_severity INT NOT NULL
status_refs_id_f6ec5d36
ctsn.status_severity [table]
id INT NOT NULL auto-incremented
`desc` VARCHAR(255) NOT NULL
color VARCHAR(15) NOT NULL
status_severity_refs_id_475e0805
status_severity_refs_id_22140eff
ctsn.node_status [table]
id INT NOT NULL auto-incremented
`desc` VARCHAR(255) NOT NULL
status_severity INT NOT NULL
node_ibfk_1
ctsn.error_log [table]
ID INT NOT NULL auto-incremented
NODE_ID INT NOT NULL
MESSAGE_ID INT NOT NULL
`time` TIMESTAMP NOT NULL
ctsn.website [table]
id INT NOT NULL auto-incremented
`desc` VARCHAR(255) NOT NULL
status INT NOT NULL
ctsn.trail_result_type [table]
id INT NOT NULL auto-incremented
description VARCHAR(255) NOT NULL
type_refs_id_ed5f43c3
ctsn.trail_result [table]
id INT NOT NULL auto-incremented
`time` TIMESTAMP NOT NULL
type INT NOT NULL
node INT NOT NULL
node_refs_id_be541cf8
ctsn.node [table]
id INT NOT NULL auto-incremented
description VARCHAR(255)
lat FLOAT(12, 0)
lon FLOAT(12, 0)
status INT
address VARCHAR(16)
update_time TIMESTAMP NOT NULL

Figure 54- CTSN Trail Schema

Note, for space reasons, the schema picture was cut in half and placed in two rows.

Table 4 - CTSN Trail Database Table Description

Table Name	**Description**
Trail Result Type	The different types of pedestrian targets the trail can detect. For example, “horse”, “walker”, and “biker” are stored as rows in here.
Trail Result	When a trail detects a pedestrian, a row gets added here. Each row contains the type of pedestrian, the time at which the detection occurred, and the node from which the result came from.
Node	Each row in this table represents a node on the trail. Each row contains the latitude and longitude of the node, its status, the mac address, and a timestamp of when the last status update occurred.
Node Status	A list of possible node statuses. Some statuses include “okay”, “offline”, or “battery critical”.
Error Messages	A list of possible error messages that can occur.
Error Log	This table stores all of the errors that occurred in the system and when.
Website Status	Possible statuses of the website. These include “okay”, “maintenance”, or “down”.
Website	A list of all the websites in the system and their status. Useful in case there is more than one frontend.
Status Severity	For each status, there is a corresponding severity such as “Okay”, “Minor”, or “Critical”. This table lists these.

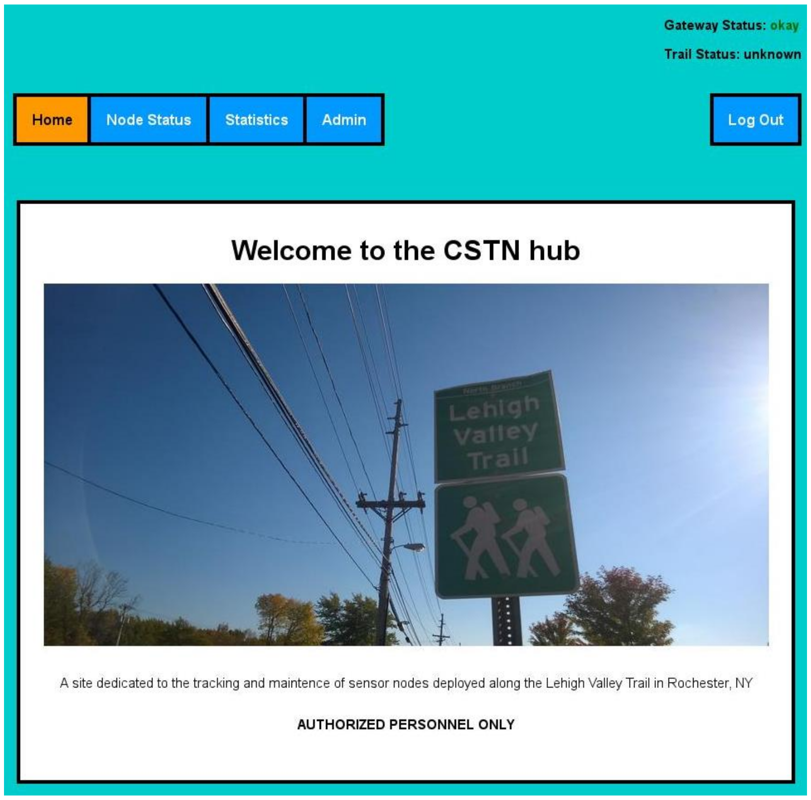
User Interface and Controls

The Django web framework is used to implement the sensor network website. Django uses the Python programming language along with an HTML/Python hybrid templating language, and allows for the user to import CSS, external and internal media, and JavaScript. A user interface was developed that allow users to access the database end of the sensor network through the Raspberry Pi's web server. Django allows for an Apache server connection, and also interfaces with several database formats, such as MariaDB, which provides all of the necessary features for our project within a relatively small footprint.

The web interface allows access to two user groups, user and admin. The standard user has the ability to view sensor node information, and to view accumulated data as it is received. This information can be filtered on a per node basis. Accumulated data from the entire network can also be viewed all at once. The standard user will also be able to view each node's status such as if the node is down or needs a battery change.

Admins can view everything a standard use can view. In addition, admins can add or remove users and nodes from the database, turn nodes off, put the website in maintenance mode, or view error messages from the trail. When a node is not "online" or a sensor is processing an alert, such as a battery getting low, the node sends a message to the gateway node. These messages are accumulated in a message center for the admin to view when desired. Messages will also be sent via email and text to the administrator's provided account. The admin will eventually also have the option to power down a node remotely. All information from the database (within reason) can be removed or modified by the admin.

Figures 55 - 60 are screenshots of the website:



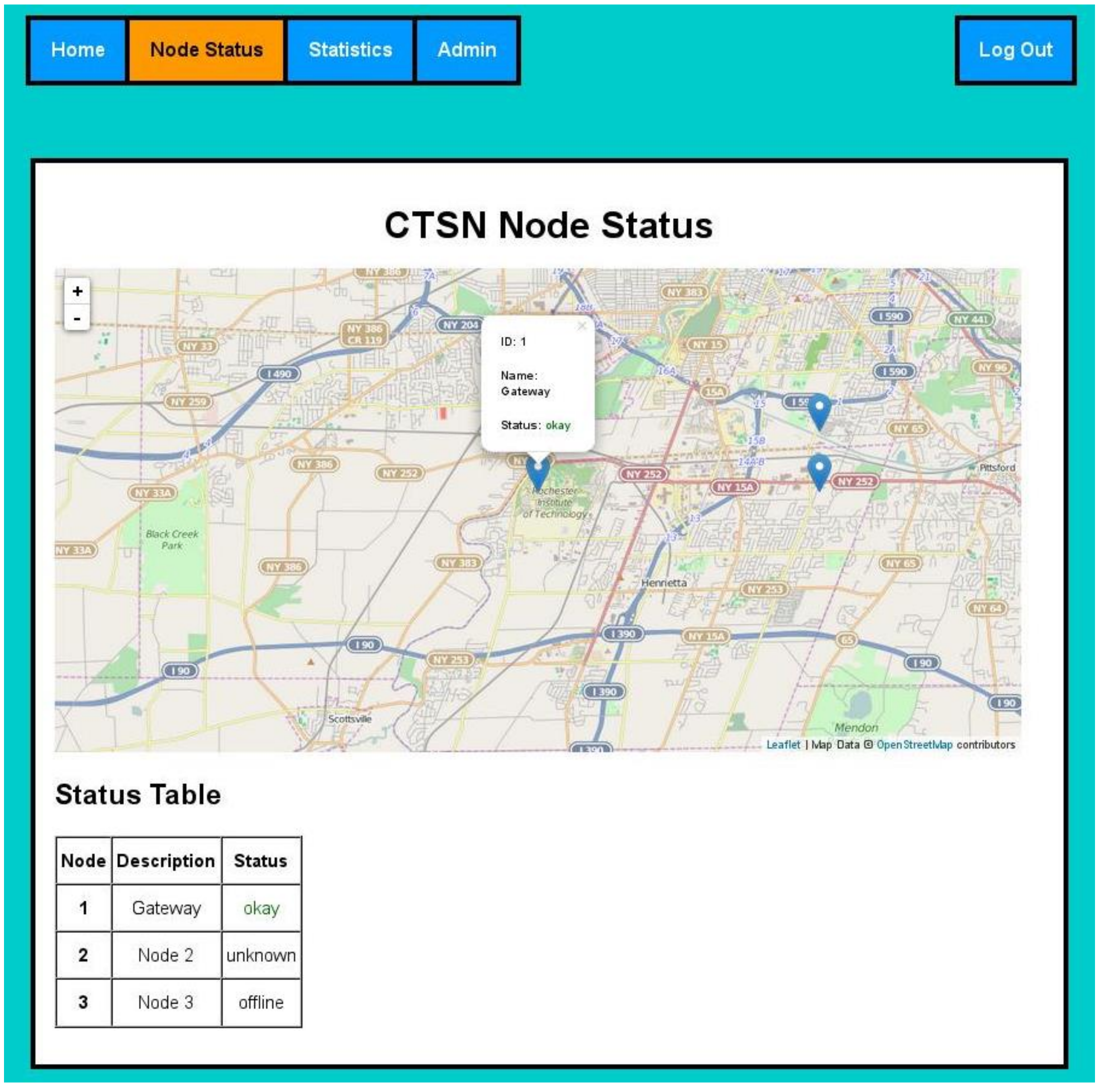
Welcome to the CSTN hub

A site dedicated to the tracking and maintence of sensor nodes deployed along the Lehigh Valley Trail in Rochester, NY

AUTHORIZED PERSONNEL ONLY

Figure 55 - Home page

Figure 55 shows the homepage of the website. It will show a welcome message, and has buttons that go to the other pages. In the upper right corner, there is a status of the gateway node, and the overall trail status. The overall trail status shows the "worst" status of all the trail nodes. In this case, one trail node has an "unknown" status, so it displays "unknown". The admin link is only viewable and accessible to users with admin level access.



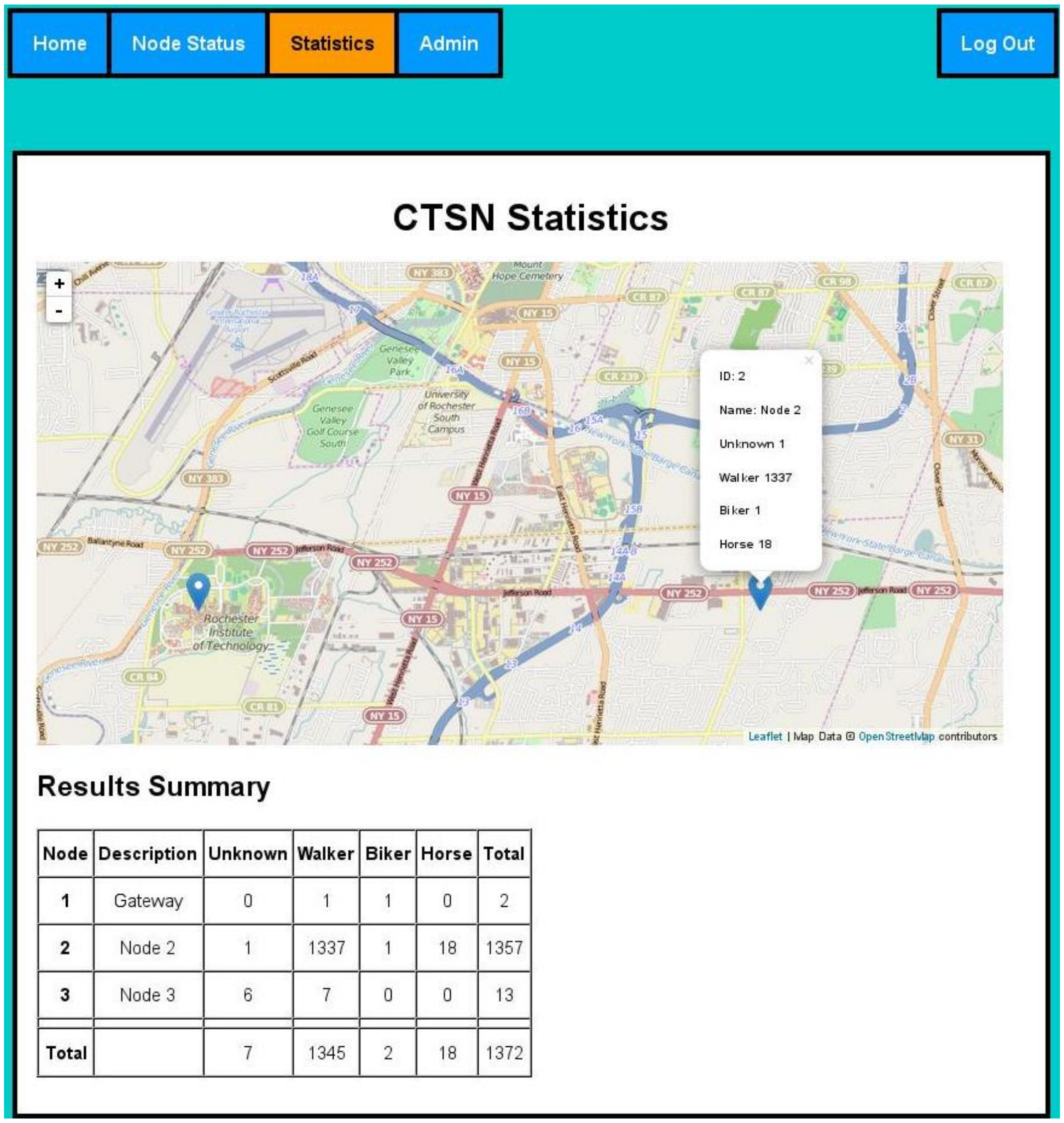
CTSN Node Status

Status Table

Node	Description	Status
1	Gateway	okay
2	Node 2	unknown
3	Node 3	offline

Figure 56 - Status page

Figure 56 shows the page that tracks the status of each of the nodes, with each node being a blue marker on an interactive map. Clicking on one of these markers will produce more specific information of the node in a popup bubble. For example, node 1 is selected in the interactive map, and it shows the Node ID, name, and the stats. Below the map is a table of each of the nodes. The table is here just in case the user does not have a JavaScript-enabled browser, they can still view the data. The Gateway and Trail node status shown on the homepage is still on this page, but cut off so the screenshot could fit.



CTSN Statistics

ID: 2

Name: Node 2

Unknown 1

Walker 1337

Biker 1

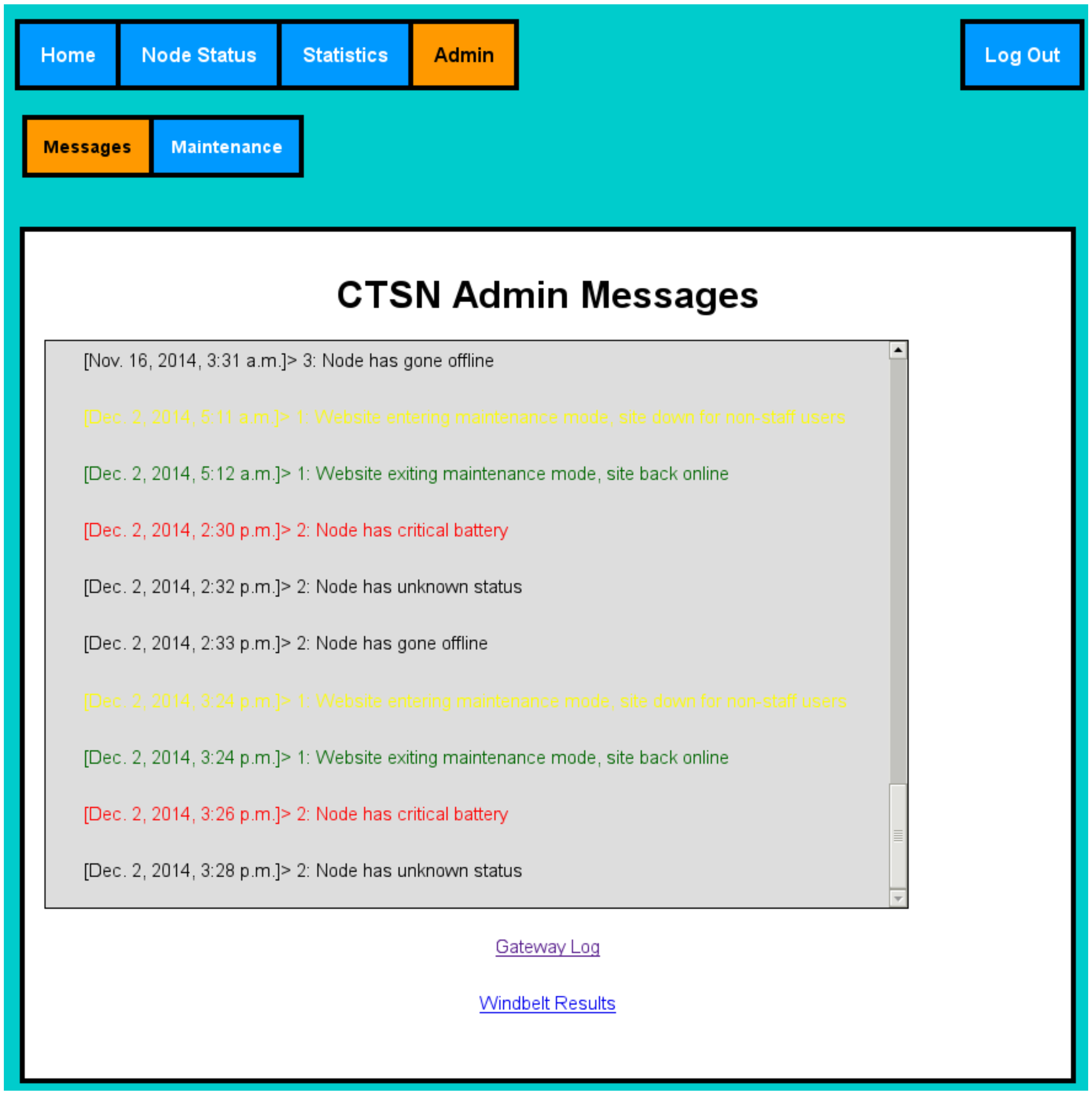
Horse 18

Results Summary

Node	Description	Unknown	Walker	Biker	Horse	Total
1	Gateway	0	1	1	0	2
2	Node 2	1	1337	1	18	1357
3	Node 3	6	7	0	0	13
Total		7	1345	2	18	1372

Figure 57 - Statistics Page

Figure 57 is the page where the user will go to get the usage data. As in the status view (Figure 56), each node is represented by a clickable marker. The user can click on any node to get specific statistics of a node, or scroll down to the table to see the total of entities viewed per node, or the total of each type of pedestrian. The Gateway and Trail node status shown on the homepage is still on this page, but cut off so the screenshot could fit.



CTSN Admin Messages

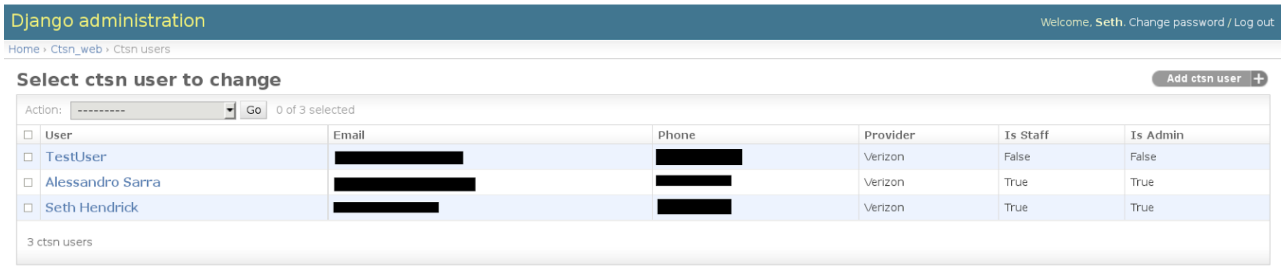
```
[Nov. 16, 2014, 3:31 a.m.]> 3: Node has gone offline
[Dec. 2, 2014, 5:11 a.m.]> 1: Website entering maintenance mode, site down for non-staff users
[Dec. 2, 2014, 5:12 a.m.]> 1: Website exiting maintenance mode, site back online
[Dec. 2, 2014, 2:30 p.m.]> 2: Node has critical battery
[Dec. 2, 2014, 2:32 p.m.]> 2: Node has unknown status
[Dec. 2, 2014, 2:33 p.m.]> 2: Node has gone offline
[Dec. 2, 2014, 3:24 p.m.]> 1: Website entering maintenance mode, site down for non-staff users
[Dec. 2, 2014, 3:24 p.m.]> 1: Website exiting maintenance mode, site back online
[Dec. 2, 2014, 3:26 p.m.]> 2: Node has critical battery
[Dec. 2, 2014, 3:28 p.m.]> 2: Node has unknown status
```

Gateway Log

Windbelt Results

Figure 58 - Admin message center

Figure 58 shows the admin message center. This is only viewable to admins. It shows all important messages that developers or maintainers need to know. Each message is color-coordinated by its severity. The Gateway and Trail node status shown on the homepage is still on this page, but cut off so the screenshot could fit.



Django administration

Welcome, **Seth**. Change password / Log out

Home › Ctsn_web › Ctsn users

Select ctsn user to change

Add ctsn user

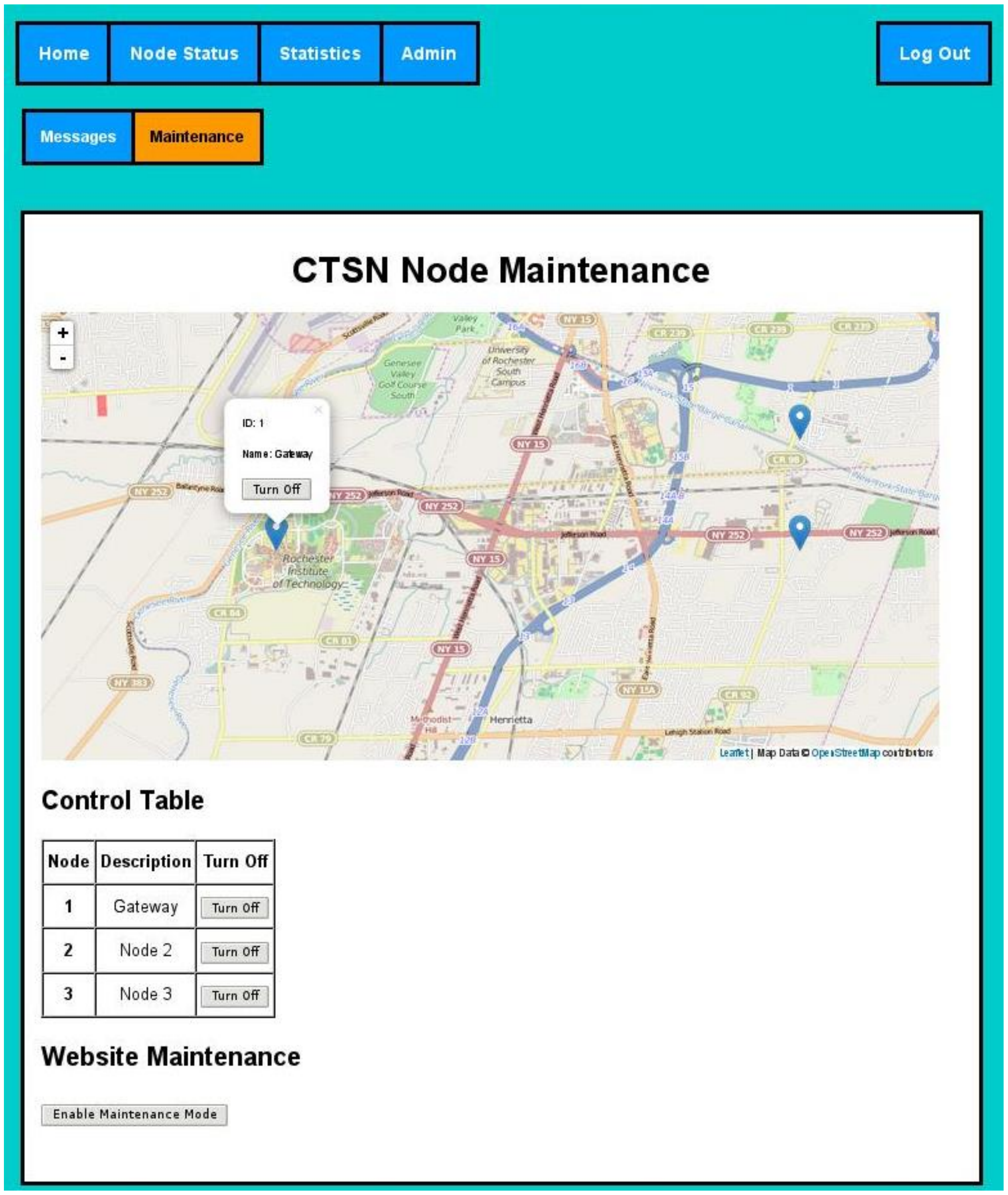
Action: --------- Go 0 of 3 selected

User	Email	Phone	Provider	Is Staff	Is Admin
TestUser	[redacted]	[redacted]	Verizon	False	False
Alessandro Sarra	[redacted]	[redacted]	Verizon	True	True
Seth Hendrick	[redacted]	[redacted]	Verizon	True	True

3 ctsn users

Figure 59 - Admin Add/Remove User View

Figure 59 shows the view associated with the Admin user account that allows an administrator to reset the password or change the user account privileges for any stored account. This is automatically provided by Django. A similar page is used to add or remove nodes from the system as well.



CTSN Node Maintenance

Control Table

Node	Description	Turn Off
1	Gateway	Turn Off
2	Node 2	Turn Off
3	Node 3	Turn Off

Website Maintenance

Figure 60 - Admin Maintenance View

Figure 60 shows the maintenance view associated with the admin user group. A future iteration of the project will allow the admins to actually power cycle a node when the "turn off" button is pressed. Right now, nothing happens when clicked. Admins can enable or disable website maintenance mode from this page. When in maintenance mode, the website is not accessible by anyone other than administrators. The Gateway and Trail node status shown on the homepage is still on this page, but cut off so the screenshot could fit.

For each of the interactive map UIs, a library called Leaflet.JS was used to put the markers on the map. Leaflet.js allows the user to zoom in and out and move the map around. OpenStreetMap is used as the base map.

Engineering Standards

- C/C++
  - Both C and C++ are standardized by the International Standards Organization. C and C++ will be used to program the Pixy Cameras, and can also be used in the gateway node to read in data from the trail nodes, and write it to the database.
- Hypertext Transfer Protocol (HTTP)
  - When a client tries the access the website, they will be using HTTP “get” and “post” requests. Django will handle the HTTP the requests, and return a webpage to the user. HTTP is also used as a form of interprocess communication inside of the gateway node so separate processes can talk to each other.
- Portable Operating System Interface (POSIX)
  - The Raspberry Pi used for the server and gateway will be running Linux, which is a “mostly-POSIX compliant” computing platform.
- ZigBee
  - ZigBee is a communication protocol for radio modules. The Digi XBee is based off on the ZigBee specification.
- Universal Asynchronous Receiver/Transmitter (UART)
  - UART is used for transmitting serial data. UART will be used to send or receive data between the MCUs and the XBee radio modules that need to be sent out or are received from the mesh network.

There are currently no industry or engineering standards for vision systems, and algorithms can be changed as desired to increase overall system performance.

Multidisciplinary Aspects

- Electrical Engineering
    - Needed to design and build the hardware associated with the signal conditioning circuit, and to interface AC with DC.
- Computer Engineering
    - Needed to program digital components such as the image processor, image sensor, and boost converter.
- Computer Engineering Technology
    - Needed to complete PCB design and implementation.
- Software Engineering / Computer Science
    - Write software for recognizing the mode of transportation of the person who walked in front of the node.
    - Write software to save all data to a database.
    - Write software so the nodes can communicate with each other.
- Mechanical Engineering
    - Build a housing to protect the nodes from the weather.
    - Design and build the windbelt for the trickle-charger.

Background

For this project, several classes offered at RIT will be of use.

- Data Communications
    - Gives knowledge of networking, which may be important for setting up the nodes to be in a network.
- Interface and Digital Electronics
    - Gives knowledge on how to use analog filter, which is used in the windbelt.
- Computer Science 1 and 2
    - Gives knowledge on computer programming, design patterns and data structures with Python.
- Computer Science 4, Applied Programming
    - Gives knowledge on how to write code in C and C++, which is how the MCU and the gateway process is programmed.
- Web I
    - Gives knowledge on how to design a static website using HTML and CSS.
- Intro to Software Engineering
    - Gives knowledge on how to build a website using the Django web framework.
    - Gives knowledge on how to communicate with a database through a website.
- In addition, some independent research was performed on EAGLE PCB design and fabrication in order to complete the battery management module.

Outside Contributors

- Dr. Jeff Wagner
    - Professor in the RIT Department of Economics. The requirements specification were based to support Dr. Wagner’s research in the area of benefits transfer analysis.
- Stanley Chan and Jared Stroud
    - Security majors at RIT. They penetration tested the server and the website and reported any vulnerabilities they found, along with ways to fix them.
- Nick Conn
    - Assisted with soldering by providing a temperature controlled toaster oven that was modified for reflow purposes, and for assistance with Eagle.
- Jeff Lonneville
    - Allowed access to Surface Mount Technology Facilities and assistance and instruction on how to use equipment.

Constraints / Considerations

Sustainability - Since the nodes will be operating far away from any potential power source, a low current source must be provided through use of wind energy. Nodes will be precharged, and the trickle charge provided to them by the windbelts will merely be a way in which the pre-existing charge will be maintained for a longer period of time. Therefore, the nodes will all be operating within a variable time constraint, and the Li-Ion batteries will require a manual recharge when this period expires. The time constraint will fluctuate dependent on the amount of wind that is converted to energy within a given time period. During windier periods, the nodes will remain in deployment for longer, and during periods of less wind nodes will need to be manually recharged more frequently. Power management algorithms will also be employed on the device to minimize the amount of power being used by the microcontroller and associated peripherals during periods of low charge. Ideally, the windbelts combined with effective power conditioning and management should significantly lengthen the deployment period of the devices.

Ethical/Privacy - The nodes will be recording the people walking by them. This makes the privacy of the people on the trail a consideration. With this in mind, the nodes will not save any photos of the trail users unless explicit approval of the trail owners has been provided. Nodes will function solely to determine the person’s mode of transportation, and send statistical data through the gateway node to the cloud server.

Property Rights/Political - Another consideration is whether or not the network can actually be deployed on the trail. The owners of the trail will need to be notified, and proper permissions will need to be obtained to perform research within the 15 mile stretch of trail.

Ethical/Privacy - It should also be considered as to who has access to the collected data. It could be a confined group of people performing research and development for debugging and statistical use, town personnel could also have access to it, or all the information could be available to the public. Access level should also be changed throughout the course of development to deployment.

Extensibility - The modules should be able to be repurposed for a variety of analog tasks. The basic platform is being adapted to the current application of collecting trail usage data for the Lehigh Valley Trail, but a variety of analogous applications can be imagined: wildlife data collection, agricultural data collection, etc. The modules need to be developed with this kind of flexibility in mind. The same needs to be considered for the database backend; it too will need to be flexible so as to be able to adapt it to other applications.

Manufacturability - The modules are fairly complicated in both the number of systems and their integration. This was one of the considerations that caused us to choose the Pixy sensor over developing our own unit. Manufacturing should be off loaded onto specialist partners as much as possible. We think we have done that to a high efficiency with our component selection.

Reliability - This is a strong consideration as we are using the network for data collection. We need to show that the modules will reliably capture the required information and that that data will make it to the database backend. We hope to have full redundancy for the wireless network across the nodes to increase reliability.

Environmental - There is a small chance that the battery may explode, causing a fire. The enclosure for the nodes should be designed to prevent the fire from spreading by making it fire retardant.

Cost

The table below shows cost for the project. This table shows the price of 5 trail nodes (for a proof concept), each with its own power system, camera, and wind belt, and one gateway node.

Table 5 - Bill of Materials

Part	Part Name	Price	Our Price	Quantity	Total Price	Source
Li-Ion Battery	**Tenergy Li-Ion 3.7V 15600mAh**	$40.49	$40.49	5	$202.45	All-Battery
Schottky Diodes	1SS389(TL3,F,D) High Speed 0.23V VF	$0.16	$0.00	5	$0.00	CE Department
Resistors, Capacitors, Inductors	--	$12.61	$12.61	5	$63.05	Mouse and Coilcraft
Boost Converter	bq25504	$6.17	$0.00	5	$0.00	Mouser
Buck Converter EVM	tps62740 EVM	$49.00	$0.00	5	$0.00	Texas Instruments
Boost Converter EVM	tps61032 EVM	$49.00	$0.00	5	$0.00	Texas Instruments
Windbelt Frame and components	--	$20.00	$20.00	5	$100.00	Built From Wood
Magnet	--	$1.00	$1.00	5	$5.00	Home Depot
Windbelt Coil	--	$5.00	$5.00	1	$5.00	Home Depot

PCB	--	$8.00	$8.00	5	$40.00	OSH Park
Radio Module	XBP9B-DMST-002	$39.00	$39.00	6	$234.00	Mouser
Camera Board	Pixy Board	$75.00	$75.00	5	$375.00	Charmed Labs
Antenna	ANT-ELE-S01-005	$3.15	$3.15	5	$15.75	Mouser
Web server / Gateway	Raspberry Pi	$35.00	$0.00	2	$0.00	Group Members
Wired Router	-	$30.00	$0.00	1	$0.00	Group Members
Ethernet Cables	-	$6.00	$0.00	4	$0.00	Group Members
XBee Breakout	992-XBEE-USB XBee-USB	$27.92	$27.92	1	$27.92	Mouser
Infrared Sensor	ZEPIR0AxS02MODG	$10.80	$10.80	5	$54.00	DigiKey
				Total Price:	$1,122.17	

Testing

Signal Acquisition and Conditioning

Testing of the power acquisition and conditioning circuit involves five phases as shown in Table 1. Phase 1, early testing, involves varying the length of the windbelt to maximize AC current production and minimize length. Subsequent tests of the early stage involve varying fan speed and the angle at which wind is applied.

*(**Warning: All Lithium-Ion battery testing MUST be accomplished with the battery in plain sight prior to remotely testing. Li-Ion batteries can become highly unstable if charging conditions are less than nominal.**)*

Table 6 - Signal Acquisition Test Descriptors

Test ID	Phase -Component	Description	Pass Condition	Outcome
P.1	Early - Windbelt Component	Vary length of the belt and measure AC voltage peak-peak using an oscilloscope.	AC current production is at a maximum, size of windbelt is at a minimum.	The size of the windbelt was larger than expected, but $2V_{AC}$ was generated consistently.
P.2	Early - Windbelt Component	Vary fan speed and measure AC voltage using oscilloscope.	AC current production is maximized.	Lowering fan speed surprisingly did not have a large effect on voltage generated. Outdoor testing saw the same numbers as indoor.
P.3	Early - Windbelt Component	Vary the angle at	AC current production is maximized.	Angle has a large effect

		which the optimal wind speed is applied and measure AC voltage.		on voltage generated. When outdoors, the randomness of the angle serves to normalize output.
P.4	Intermediate I - AC/DC Rectification Component	Connect optimal design from early testing phase to half-wave rectifier circuit. *(Requires completion of windbelt.)*	Smoothing levels are adequate, and levels are held at 80% or higher of 3.3V with minimum fall time.	Although 3.3 V was not achieved, 1.2 V_{DC} proved to be adequate voltage for the boost converter to operate.
P.5	Intermediate I - AC/DC Rectification Component	Connect optimal design from early testing phase to full-wave rectifier circuit. *(Requires completion of windbelt.)*	Smoothing levels are adequate, and levels are held at 85% or higher of 3.3V with minimum fall time.	A full-wave rectifier was decided against, due to a larger voltage drop and higher cost and complexity.
P.6	Intermediate II - Primary Boost Converter Compone nt	Connect output of rectifier to boost converter at VIN_DC port (pin 2). Measure output of boost controller at VSTOR (pin 15) using an	Measure voltage levels at output over elongated periods of time. Levels should not exceed those allowable by the battery (4.2V). Observe and track the DC output signal as it fluctuates over time to determine whether MPPT is necessary. Verify that the boost controller remains on	The boost converter was not tested for a configuration where MPPT was not enabled. This was considered a waste of time, since MPPT is a

		oscilloscope (*Requires completion of custom PCB power management module.*) When running without MPPT, reference Figure 4 in the bq25504 technical document for the proper port configuration.	during harvesting periods.	very powerful feature of the device. With MPPT enabled, the boost converter provided levels that were safe for the battery, never exceeding 4.2 V or dropping below 3.1 V.
P.7	Intermediate II - Lithium Ion Battery Component	Manually charge the Li-Ion battery at the specified voltage for a nominal charge period. Measure the output voltage with a multimeter. *(May be done in an automated fashion as well.)*	Measure voltage levels of the battery periodically until it is determined that the battery has fully discharged. Record the time it took for the battery to discharge.	The battery was attached to the boost converter and allowed to discharge to the under-voltage level. At this point, the boost converter automatically cut power, and the battery was disconnected.
P.8	Intermediate II - Voltage Regulation Component	Connect fully charged Li-Ion battery to voltage regulator on custom PCB.	Measure voltage levels at both ports to ensure that 3.3 V is being provided to the XBee and 3.7 V is being provided to the second	The battery reads 4.2 V when fully charged, and about 3.5 V when

		Test DC output with multimeter at both outputs. *(Should be completed after battery testing)*	boost converter (5 V out).	discharged to the point that the boost converter shuts down. During operating periods, the buck converter reads 3.3 V and the secondary boost reads 5 V.
P.9	Intermediate II - Secondary Boost Converter Component	Connect 3.7 V pin to secondary boost converter and measure voltage at the output using a multimeter. Do not configure MPPT yet. *(Not sure which boost converter will be used yet)*	Measure voltage levels at the output of the boost converter. Output should be 5V to adequately supply the Cortex processors. Observe the output over time to determine whether MPPT is required at the secondary boost converter level.	MPPT is required to operate the circuit more efficiently. 5 V was supplied to the camera module, regardless of input voltage from the battery management solution.
P.10	Advanced I - Primary Boost Converter Component	Connect the battery to the boost converter as shown in Figure 4 of the bq25504 technical document. Measure duration and level of	Battery duration, charge level and output should be within 20% of the measured discharge period of the battery when measured separately. Compare discharge rate with those of the battery without the sustainable energy source.	It was determined that running the boost converter without MPPT was inefficient and impractical for use with a battery

		charge without MPPT enabled across the terminals of the battery and at pin 15. *(Should be completed after battery testing. Do not leave the battery unattended!)*		management application. This test was not performed.
P.11	Advanced I - Primary Boost Converter Component	Configure the boost converter as shown in Figure 2 of the bq25504 technical document. Measure duration and level of charge with MPPT enabled for a solar energy application with the Li-Ion battery attached. *(Should be completed after battery testing. Do not leave the battery unattended!)*	Battery duration, charge level and output should be within 20% as above for elongated periods. Compare discharge rates with those of the battery without the sustainable energy source, and with the source but without MPPT.	This test was completed for a windbelt application. The intermediate step was not necessary. Discharge rates were determined acceptable for the application. Adding the sustainable source elongated the battery life by about 2 hours over a 24 hour period.
P.12	Advanced I - Secondary	Configure the boost	Measure voltage at the output over elongated	The signal was

	Boost Converter Component	converter for use with MPPT and measure the output of the converter using a multimeter. *(Should be completed after non-MPPT testing if required)*	periods to ensure that the signal provided is 5 V, and that adding MPPT further stabilizes the signal for use with the MCU.	measured at 5 V regardless of the input voltage to the boost converter when MPPT is enabled.
P.13	Advanced II - Secondary Boost Converter Component	Configure the boost converter for use with MPPT and connect the M0 and M4 processors. Measure voltage at the output using a multimeter. *(Should be completed after MPPT testing without load.)*	Measure voltage at the output over elongated periods to ensure that the signal provided is 5 V.	Although voltage level drops at the output of the boost converter when the camera module is connected by about 0.35 V, the secondary boost converter still reads 5 V at the output.
P.14	Advanced II - Integration with Focus on Signal Acquisition	Deploy a single node in a controlled environment and monitor power conditions closely. *(Should be completed*	Measure voltage levels, battery duration and charge throughout the elongated period, and confirm that CV algorithms, networking, and all power dependent subsystems are fully-functional.	This was not completed since CV was not functional on the embedded camera. Bursts of 1000 packets were sent to test the radio

		after all other component testing.)		functionality, and the ability of the buck converter to provide current to the radio. This was successful during a 4 hour testing period.
P.15	Advanced II - Integration with Focus on Signal Acquisition	Deploy a single node in an outdoor environment. Allow node to operate without intervention and monitor conditions remotely. *(Should be completed after controlled integration testing.)*	Measure voltage levels, battery duration and charge throughout the elongated period, and confirm that CV algorithms, networking, and all power dependent subsystems are fully-functional.	With both the camera module and radio connected to the power module, the node stayed powered on for approximately 24 hours. CV was not functioning at the time, but TX tests were run.
P.16	Advanced III - Acceptance with Focus on Power Management Requirements	Deploy multiple (3-4) nodes in an outdoor environment. Allow node to operate without intervention and monitor conditions remotely. *(Should be completed*	Measure voltage levels, battery duration and charge throughout the elongated period, and confirm that CV algorithms, networking, and all power dependent subsystems are fully-functional.	This was not completed, as CV was not fully functional on the cameras that the power module was characterized for.

		after controlled and single-node integration testing.)		

Intermediate testing focused on continuing testing with the optimized design chosen from phase 1. The rectifier circuit was connected to the windbelt, and smoothing and fall time were optimized. For the second intermediate phase, the focus was on providing safe levels at the output of the boost converter with the MCU connected to a non-battery power source.

Advanced testing allowed for connection of the battery and MCU to the boost converter and deployment in an outdoor environment. Battery voltage levels, duration, and charge levels will be measured for as long as the battery maintains its charge. Once components are proven fully functional, the signal acquisition system will be tested as a whole in a controlled testing environment. Acceptance testing was performed in the field, with less distance between nodes for easy maintenance and monitoring prior to actual deployment on the Lehigh Valley Trail. Tests were run to ensure that all power-dependent requirements were satisfied after integration level testing was completed.

Computer Vision Testing

The CV was not tested extensively before this iteration ended. The Pi Cam was able to detect pedestrians in a lab setting, but that's it. For the next iteration, the tests described below will be performed.

To test the software algorithm that determines what walks by a node, the nodes can be set up in a controlled environment, such as inside of a lab. A person can then walk by with or without a bike, and the node should be able to identify if the person is walking or riding a bike. The subject should walk past the node at various speeds, and the node should still be able to make a capture. The node should be able to pass captured information to the rest of the nodes, and to the gateway so it can be recorded by the database.

Various lighting scenarios should be tested as well. In a lab environment, the lighting should be adjusted so it gets dimmer until the node is unable to make accurate readings. This will be the minimum light needed for operation, and the node might not require frames to be captured at times like this in the field. During the lab test, a light should be positioned at various angles, such as behind the node shining on the target, behind the target shining on the node, overhead, and to the left and right of the node. Regardless of where the light is positioned, the node should still operate accurately.

When the node is built, and the software written, a stress test that can be performed is to deploy the nodes on the RIT quarter mile, which gets a lot of foot traffic. This test will also show just how accurate and fast a node is. If the node just cannot keep up with the traffic, some redesigning might need to occur.

Table 7 - Computer Vision Test Descriptors

Test ID	**Phase**	**Description**	**Pass Condition**
V.1	Early - Image Sensor Component	Verify that the image sensor is correctly capturing images by connecting the	Images in front of the lens are rendered correctly in the Pixy Cam GUI window. It can be

		Pixy Cam via USB, loading the GUI and observing.	assumed that the M0 is functioning correctly.
V.2	Early - Image Sensor Component	Verify that the image sensor is correctly identifying colors by running the included color identification algorithms with the board connected via USB	The chosen color is amplified in processed images, as shown in the Pixy Cam GUI window. It can be assumed that the M0 is functioning correctly.
V.3	Intermediate - Infrared Component	Connect the board via USB and connect an Infrared sensor to the camera via GPIO or UART. Dim the room of all light sources. Verify that the sensor data are correctly displayed in the Pixy Cam GUI window by walking in front of the camera and observing..	The image should be correctly represented with more orange to red colors showing higher temperatures. The background should contain blue and purple colors for lower temperatures.
V.4	Intermediate - CV Algorithm Component	Compile and load the CV software onto the Pixy Cam. Walk by the Pixy Cam to verify whether the algorithm is functioning properly. *(To be tested with and without Infrared sensor connected.)*	The image should be identified, and the correct CV size capture, speed and direction determinations are made.
V.5	Intermediate - CV Algorithm Component	Vary speeds at which the subject walks by the Pixy Cam to verify that the algorithm still functions properly. *(To be tested with and without Infrared sensor connected.)*	The image should be identified and the correct CV size capture, speed and direction determinations are made.
V.6	Advanced - CV Algorithm Component	Walk by the Pixy Cam with a bike or other large object to verify that the algorithm still functions properly. *(To be tested with and without Infrared sensor connected.)*	The CV algorithm is able to correctly discern between a walking and biking commuter based on speed of the individual, and the correct output displays in the GUI.

V.7	Advanced - CV Algorithm and Infrared Component Integration	Compile and load the CV software onto the Pixy Cam. Dim the lights in the room. Walk by the Pixy Cam to verify whether the CV algorithm is functioning properly. *(To be tested after CV and IR component tests pass.)*	The algorithm should be able to use the Infrared data with the lights dimmed to accurately detect size, direction and speed of an individual.
V.8	Advanced - Computer Vision and Networking System Level Integration	Compile and load the CV software onto the Pixy Cam. Connect the cam via USB. Connect the PixyCam to the XBee as well via GPIO or UART header and provide power to it via pre-charged Li-Ion battery connected to custom voltage regulation unit. Dim the lights in the room. Walk by the Pixy Cam to verify whether the CV algorithm is functioning properly. Turn on the lights and perform the same tests. *(To be tested after CV, IR, voltage regulation and GPIO component tests pass. The website GUI should also be operational, but data can alternatively be viewed through local GUI.)*	The algorithm should be able to use the Infrared data with the lights dimmed to accurately detect size, direction and speed of an individual. The algorithm should also work with lights on. These data should be broadcasted to a central node, server, and pushed to a website to view for confirmation.
V.9	Advanced - Computer Vision and Power Management System Level Integration	Compile and load the CV software onto the Pixy Cam. Connect the PixyCam to the secondary boost converter. *(Warning: Do not perform this step until component level testing of the secondary boost converter is complete.)* Connect the PixyCam to the XBee as well via GPIO or UART header and provide power to it via pre-charged	The algorithm should be able to use the Infrared data with the lights dimmed to accurately detect size, direction and speed of an individual. The algorithm should also work with lights on. This data should be broadcasted to a central node, server, and pushed to a website to view for confirmation.

		Li-Ion battery connected to custom voltage regulation unit. Leave out everything prior to the battery in the Signal Conditioning circuit. Walk by the Pixy Cam to verify whether the CV algorithm is functioning properly. Turn on the lights and perform the same tests.	
V.10	Advanced - Computer Vision Acceptance Testing	Compile and load the CV software onto the Pixy Cam. Connect the PixyCam to the secondary boost converter. *(Warning: Do not perform this step until component level testing of the secondary boost converter is complete.)* Connect the PixyCam to the XBee as well via GPIO or UART header and provide power to it via pre-charged Li-Ion battery connected to custom voltage regulation unit and add signal acquisition and conditioning circuit including primary boost converter, rectifier, and windbelt. Walk by the Pixy Cam to verify whether the CV algorithm is functioning properly. Turn on the lights and perform the same tests. Vary size of CV subjects. *(Perform this test after System level integration is complete for each system involved.)*	The algorithm should be able to use the Infrared data with the lights dimmed to accurately detect size, direction and speed of an individual. The algorithm should also work with lights on. This data should be broadcasted to a central node, server, and pushed to a website to view for confirmation. Extended testing should be performed at this level once all system level and integration is complete. The node should be able to process CV algorithms for an extended period with sustainable energy available.

Enclosure testing

The node will be deployed outside in Rochester's infamously bad weather. Therefore, testing to ensure the node remains intact in extreme weather needs to occur once the weather-proof chamber is completed. Since the enclosure was only recently completed, these tests were not performed in this iteration.

Table 8 - Enclosure Test Descriptors

Test #	**Test Name**	**Description**	**Reason**	**Pass Condition**
E.1	Temperature Test	The nodes should be placed in a cold environment to simulate Rochester's winter, and a hot environment to simulate Rochester's summer.	Rochester has a variety of weather conditions that can range to several degrees below zero, and up to almost 100 degrees. The nodes will be outside during these times, and should be able to survive in them.	If all the node's hardware still works after being in the extreme environments, the test passes.
E.2	Rain Test	Water can be sprayed on the enclosure to simulate rain, and a powerful fan can be used to simulate consistent wind. For these tests, paper towels or some kind of other material that reacts to water should be placed inside the enclosure instead of the expensive hardware. This way, if water does manage to get in, no hardware is damaged.	Rochester can have days of rain in a row. The housing needs to be able to keep the water out at all costs, otherwise the hardware might get fried.	If no water gets into the enclosure, the test passes.

E.3	Drop Test	There should be a drop test of 5-6 feet to ensure that the trail node's enclosure is sturdy enough to protect the hardware inside. This can be done by placing a raw egg, or something else cheap and easily breakable, in the enclosure instead of the expensive hardware.	It is possible that while deploying the nodes, they could be dropped. The enclosure should protect the hardware from drops.	If the egg cracks when being dropped, then the enclosure is not sturdy enough, but if the egg is intact, the the test passes.
E.4	Fire test	Pack flammable materials inside of the enclosure and set fire to it (with a fire extinguisher nearby).	There is a chance that the battery can catch fire. To prevent a forest fire, the enclosure should be able to contain any fire that may occur from the inside.	If the fire stays contained within the enclosure, the test passes.

Network Architecture Testing

It is essential that all the nodes are able to communicate with each other over a long distance. To ensure that all the nodes will be communicating with each other through the XBees, the following tests can be done:

1) The first test that should be performed is a small scale test to ensure that the XBees are able to be configured through UART and send messages via the DigiMesh protocol. This test should be performed once the XBees are acquired.
   a) Acquire 4 XBees.
   b) Attach the 4 XBees to either Raspberry Pis or the Pixy Cam through UART. A mixture of Pis and Pixy Cams is preferred, as the Pixy Cam will be the MCU for the trail nodes, and the Pi will eventually act as the gateway node.
   c) Ensure the XBees can be configured through UART. If they are unable to be configured, then the XBee Development kit might need to be purchased in order to program the XBees.
   d) Ensure the XBees can send and receive messages from each other. If they are able to, then the XBee Development Kit is not needed.
   e) For this test, the XBees can be in close proximity with each other. Test 3 increases the distance.

Test 1 passed. After the XBees were configured, and the packet forming and parsing software was written, the XBees did exactly what was advertised. The programming was easy thanks to Digi's XCTU software. The XBees were able to send and receive messages to each other, as well as broadcast a message to all XBees.

2) The second test that should be performed is a test to see how far away the XBees are able to communicate with the selected antenna. This should be done once the previous test passes.
   a) Create 2 nodes by connecting an XBee to either a Pi or a Pixy Cam through UART.
   b) Make one node stationary, and take the other node and go as far away from the stationary node as possible before communication cuts out.

c) The biggest distance between two nodes on the trail is about 1.75 miles, so the XBees should be able to communicate with each other from at least 2 miles apart.
d) If this test fails, then a more powerful antenna needs to be purchased.
e) Preferably, the test should be performed on the trail itself, as that is where the nodes will eventually be placed.

Test 2 failed. It was done by connecting a stationary XBee to a power supply on campus, and another was connected via laptop and was walked around campus. Digi's XCTU software was running on it, and an XBee was connected via USB. The XCTU software could connect to the stationary XBee if it was in range. The maximum range on campus was only a couple hundred yards. The good news is this matches the XBee's documentation for urban or indoor environments. So in theory, when brought into an open environment such as the trail, the range should increase dramatically. If not, more powerful antennas are needed.

3) The final test is a small scale trail test. This is where a subset of the trail has nodes deployed to it. If this test passes, then it is probably safe to purchase more nodes and cover the entire trail. This test can be performed when the last test passes.
   a) Attach 4 XBees to either Raspberry-Pis or Pixy Cams.
   b) Deploy 4 nodes at positions 17, 2, 3, and 4 (referring to Figure 2). These four positions are the most spread apart spots on the trail.
   c) If all the nodes can communicate with each other, then the test passes, and it is probably safe to scale up to cover the whole trail.
   d) If the test fails, then some of the nodes might need a more powerful antenna.

Since test 2 failed, this one was never performed.

Server Security Testing

Once the Raspberry-Pi server and all security implementation is in place, an attempt was made to take control of the server from an outside source using DDOS, SQL injection and brute forcing.

Table 9 - Server Security Test Descriptors

Test #	**Test Name**	**Description**	**Reason**	**Pass Condition**	**Outcome**
S.1	Ping test	The router should not be pingable from an outside source.	When many people try to ping a server, it could create a DDOS attack. Therefore , pinging should be removed to prevent this from happening.	Trying to ping the router from outside the network should not work.	Since the server was located on RIT's campus, this test passes automatically since someone cannot ping rit.edu.
S.2	Disable Root Login	While SSHing into the server, users should not be able to login as root	Root is all knowing in Linux. If someone were to somehow gain access to root, they could wreak havoc on the server.	Trying to login as root results in an access denied.	Pass. Trying to login as root fails.
S.3	SSH non-standard port test	While trying to SSH into the server using the default SSH port, the result should be not being able to connect.	By moving the SSH port to a non-standard port, it makes it difficult for some hackers to find the SSH port, and try brute forcing.	Trying to login to port 22 via SSH will fail. Trying to login to the non-standard port will succeed.	Pass. Trying port 22 fails.

S.4	Disabled Password Test	Trying to login with SSH will have an access denied without an SSH key. SSHing will not be allowed without a password.	SSHing with a user name and a password is not safe for a server. A hacker can brute force their way in.	SSHing into a server without a key will result in an access denied. SSHing will not ask for a password.	Pass. A key is required to login to the server.
S.5	White hat hacker test	Borrow one of the many security majors the group knows, give them the IP address, and have them attempt to take the server down.	Having a friendly hacker attempt to break into the server will emulate a more sinister hacker trying to get in. If any information is compromised by the friendly hacker, they will not steal it.	The hackers should not be able to take the server down.	Pass. Two security majors tried to hack the server. They even complimented on how secure it was.

Website Testing

The website is the gateway to the data. However, it can also be a gateway for a hacker wreak havoc. These tests will confirm that anyone with dark intentions will not be able to compromise the server from the website.

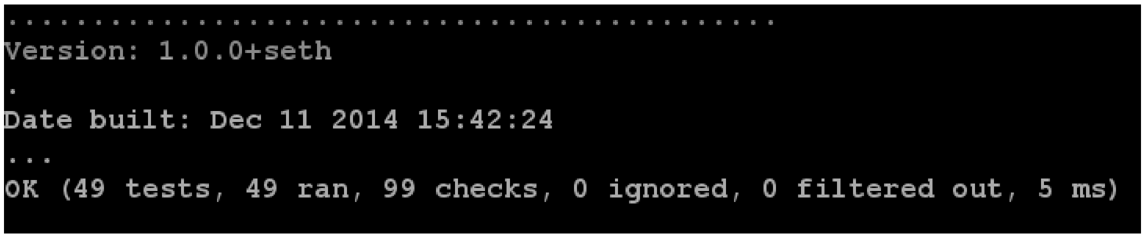
Table 10 - Website Test Descriptors

Test #	Test Name	Description	Reason	Pass Condition	Outcome
W.1	Sanitation Test	Data from the website is sent to the server via http get and post requests. Some commands sent that could break the server include "; && rm -rf /" and "drop table *;". These commands should break the server.	If the data are not "sanitized" correctly, it is possible for a user to drop a table from the SQL database, or execute shell commands.	Posting and getting from the server using common commands that break a server should not break the server.	Pass. Security Majors tried this and Django prevents it from happening.
W.2	Password protection test	Connect Wireshark, and see if user names / passwords are able to be seen leaving the client and going to the server.	If the website is meant to have a limited set of users, then a user name and a password are needed. The passwords need to be protected.	The password leaving the client should be hashed and salted so a hacker cannot steal them.	Failed originally. We were not using https, which means the passwords were being sent over in plain text. Since then, https has been implemented.
W.3	User login test	Ensure valid users can log in and invalid users cannot log in	The data need to be available to a limited set of users. Therefore, a user name and	Valid users can log in, invalid users are rejected.	Pass. Need a valid username to login

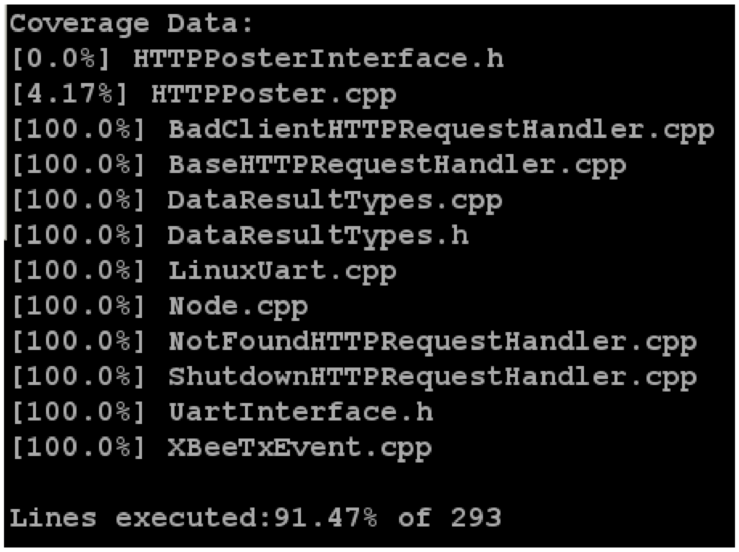
			a password is needed to gain access to the website.		
W.4	White-hat hacker test	Borrow one of the many security majors the group knows, tell them the website, and see if they can break in.	Having a friendly hacker attempt to break into the website will emulate a more sinister hacker trying to get in. If any information is compromised by the friendly hacker, they will not steal it.	If the friendly-hacker cannot break in, the test passes.	The outcome of this was the security majors provided us with a document with suggestions on how to make the site more secure. Their suggestions were followed.

Gateway Software Testing

The software that ran on the gateway node was extensively unit tested with the help of the C++ unit testing library CppUTest. There were two sections of code that were extensively unit tested, and that is the "common" software that could run on any software in the CTSN (such as the gateway, or on the CV server), and the gateway software that only ran on the gateway node. For the common software, 49 unit tests were written with 91.47 % test coverage. This is shown in Figure 61**.**



```
.........................................
Version: 1.0.0+seth
.
Date built: Dec 11 2014 15:42:24
...
OK (49 tests, 49 ran, 99 checks, 0 ignored, 0 filtered out, 5 ms)
```



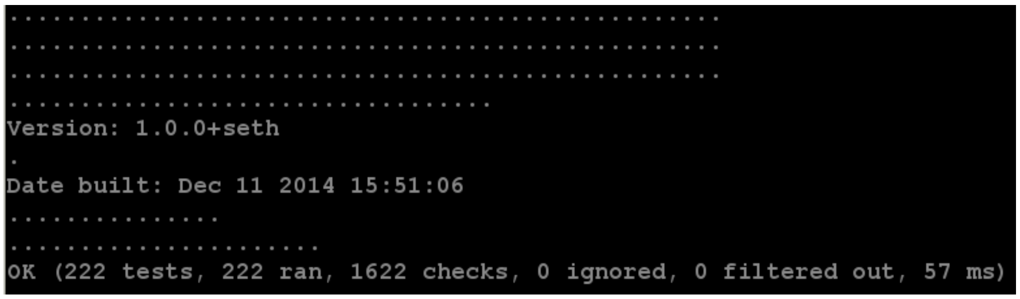
```
Coverage Data:
[0.0%] HTTPPosterInterface.h
[4.17%] HTTPPoster.cpp
[100.0%] BadClientHTTPRequestHandler.cpp
[100.0%] BaseHTTPRequestHandler.cpp
[100.0%] DataResultTypes.cpp
[100.0%] DataResultTypes.h
[100.0%] LinuxUart.cpp
[100.0%] Node.cpp
[100.0%] NotFoundHTTPRequestHandler.cpp
[100.0%] ShutdownHTTPRequestHandler.cpp
[100.0%] UartInterface.h
[100.0%] XBeeTxEvent.cpp

Lines executed:91.47% of 293
```

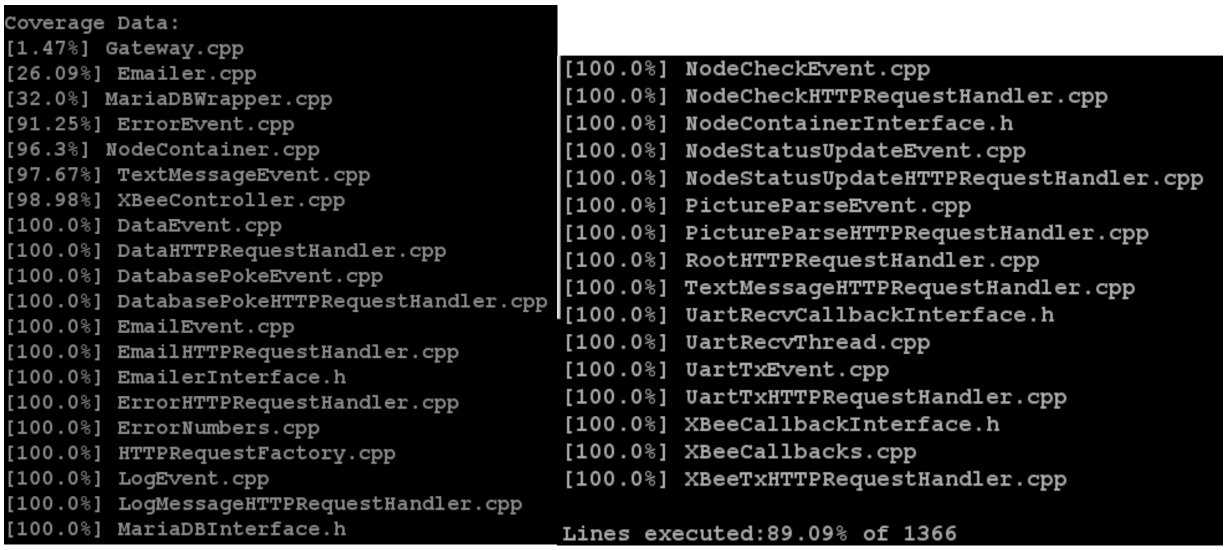
Figure 61- Unit Test Results for CTSN common

The top screenshot shows that all tests pass for CTSN common. The bottom picture shows the test coverage for each file in the CTSN common project, and how much of each file is unit tested.

For the gateway software, 222 unit tests were written with 89.09% test coverage. This is shown in figure 62.



```
..............................................
..............................................
..............................................
...............................
Version: 1.0.0+seth
.
Date built: Dec 11 2014 15:51:06
..............
.....................
OK (222 tests, 222 ran, 1622 checks, 0 ignored, 0 filtered out, 57 ms)
```



```
Coverage Data:
[1.47%] Gateway.cpp
[26.09%] Emailer.cpp
[32.0%] MariaDBWrapper.cpp
[91.25%] ErrorEvent.cpp
[96.3%] NodeContainer.cpp
[97.67%] TextMessageEvent.cpp
[98.98%] XBeeController.cpp
[100.0%] DataEvent.cpp
[100.0%] DataHTTPRequestHandler.cpp
[100.0%] DatabasePokeEvent.cpp
[100.0%] DatabasePokeHTTPRequestHandler.cpp
[100.0%] EmailEvent.cpp
[100.0%] EmailHTTPRequestHandler.cpp
[100.0%] EmailerInterface.h
[100.0%] ErrorHTTPRequestHandler.cpp
[100.0%] ErrorNumbers.cpp
[100.0%] HTTPRequestFactory.cpp
[100.0%] LogEvent.cpp
[100.0%] LogMessageHTTPRequestHandler.cpp
[100.0%] MariaDBInterface.h
```

```
[100.0%] NodeCheckEvent.cpp
[100.0%] NodeCheckHTTPRequestHandler.cpp
[100.0%] NodeContainerInterface.h
[100.0%] NodeStatusUpdateEvent.cpp
[100.0%] NodeStatusUpdateHTTPRequestHandler.cpp
[100.0%] PictureParseEvent.cpp
[100.0%] PictureParseHTTPRequestHandler.cpp
[100.0%] RootHTTPRequestHandler.cpp
[100.0%] TextMessageHTTPRequestHandler.cpp
[100.0%] UartRecvCallbackInterface.h
[100.0%] UartRecvThread.cpp
[100.0%] UartTxEvent.cpp
[100.0%] UartTxHTTPRequestHandler.cpp
[100.0%] XBeeCallbackInterface.h
[100.0%] XBeeCallbacks.cpp
[100.0%] XBeeTxHTTPRequestHandler.cpp

Lines executed:89.09% of 1366
```

Figure 62- Unit Test Results for Gateway Software

The top screenshot shows that all tests pass for the gateway software. The bottom pictures show the test coverage for each file in the gateway software project, and how much of each file is unit tested.

In total, 271 unit tests were written for the gateway and CTSN common software with a very high amount of test coverage.

Risks

Theft and Weather-proofing

One risk is the nodes might get damaged or stolen. The nodes will need to be weather-proofed to prevent damage from the weather, and ruggedized in case they are dropped. To minimize the risk of the nodes being stolen, they might need to be hidden, such as in a fake rock or camouflaged. Another option would be to place nodes at a vantage point by raising them up with a pole or extending device. This would not only deter theft, but also would allow for the field of view necessary for the CV algorithms that will be used.

Use of Low-Power Modes

There is a risk of the nodes using so much power that they cannot be deployed for very long. Some things that can be done to extend the power of the node could be to put them to sleep at night, and to ensure there is not a lot of CPU processing when the trail is empty by forcing a low power state during these periods. Although these techniques, along with the trickle charger attached to the nodes, can extend the battery's charge, it will probably not be enough to sustain the nodes indefinitely. Therefore, there should be some kind of notification sent to the frontend when a node gets close to or runs out of power, so that the batteries can be replaced.

Battery Damage and Fire

Regarding the battery, there is risk of fire if it is not properly charged with a regulated current at a specified rate, as with all Lithium based batteries. The boost converter should mitigate these risks, but the notification system should once again be used to alert the user and the system when the battery is in a bad state. This would occur after the boost converter starts exhibiting signals at the output that are in excess of what is allowed by the battery, and should result in shutdown of the device, or the signal acquisition portion of the device. Overcharge and undercharge are both a concern with this type of battery as well, and similar notifications should be sent when battery life is at 95% and less than 5% to alert the user that the node's battery is in a critical state. DC signal harvesting should not be active in either case, which can be programmed through the boost converter as well.

CV Procedural Difficulties

The most difficult part of the project is probably going to be identifying the specimens that walk or ride within the line of sight of the node. The node's software needs to determine if the person is walking, riding a bike, or riding a horse, but other factors might come into play, such as lighting, small animals, how close or how far the person is away from the node, how many people are in the shot, and other unknowns. It is going to take some trial and error to get the algorithm right. Use of the Pixy cam will mitigate these risks somewhat, due to its ability to utilize robust versions of modern-day CV algorithms, and due to all of the support and example material that is available through online forums.

High-Risk Investigations

Several components were identified as high-risk, and a detailed investigation has been conducted for each of them, including the choice of sustainable energy used for each node, the MCU, and the sensor node network architecture.

Windbelt

The windbelt can be considered a high-risk component simply due to its nature as a newly innovated sustainable energy solution. Because of this, there are very few example cases on which to conduct research. Furthermore, the results of those cases are vastly diverse due to custom construction of the windbelt and application in widely different settings. Because of this, original research will be needed in identifying an optimal variant of the design and applying that design in the most effective manner.

Other risk considerations are whether or not the windbelt can provide a large enough signal to the amplification stage of the circuit, and whether it can do so consistently. To mitigate this risk, an optimized version of the windbelt will be used, where dimensions, belt length, magnet and coil sizes and distances have been carefully considered and tested to maximize AC current production and minimize size. AC rectification will function not only to convert AC to DC, but will also provide limited reverse-current protection.

The majority of risk mitigation will be handled by the boost converter, which will allow for the use of MPPT algorithms. MPPT will track current production over time, calculate an average, and then adjust internal potentiometer values to zero in on that average during over-efficient and under-efficient periods. MPPT will greatly reduce the risk of battery damage, plating, and fire for this reason. Extensive testing will be conducted in a controlled environment prior to exposing the signal harvesting system to the outdoors. It is possible that no wind will be available in the area. During extended periods of little to no harvesting, the admin must once again be made aware through use of a notification system that no charge is being generated.

Sensor-Node Architecture

The risks associated with the sensor nodes and network communications mainly relate to whether or not range of the nodes will be sufficient. The XBee documentation claims that the communication range of the XBees is 4 miles with a 2.1dB antenna. However, documentation regarding range is typically inaccurate. If the range of the XBees is not long enough, adding a more powerful antenna to the XBee radio modules should suffice.

Schedule

Anything marked “R2” did not make it for this first iteration of the project. It will be done in the second iteration.

Table 5 - Milestone Chart

Milestone	**Team Member in Charge**	**Modified Completion Date**	**Original Completion Date**	**Comments**
1. Contact Monroe County Discuss deployment options for sensor nodes.	Jared	R2	10/27/2014	Did not get to the deployment stage.
2. Networking Architecture Configuration and Testing		R2	6/15/2014	
2.1 Configure XBees for DigiMesh and have them communicating in close proximity	Seth	10/16/2014	6/1/2014	COMPLETED
2.2 Range Test	Seth, Alex	R2	6/9/2014	Failed. We did not get the required range from the XBee. That being said, we tested it on RIT’s campus, which is considered an urban environment. If we went to a less urban environment, it may have worked better.
2.3 Small-scale trail deployment	Seth, Alex	R2	6/15/2014	Never done since 2.2 Failed.
3. Windbelt power module design		10/2/2014	6/18/2014	COMPLETE

3.1 Breadboard prototyping	Alex	11/21/2014	6/1/2014	COMPLETE Breadboard prototyping is complete. The secondary boost converter and buck converter work well together. Power has been supplied to both an XBee radio and PixyCam through the battery management system.
3.2 PCB design	Alex	11/26/2014	6/10/2014	COMPLETE
3.3 Ship design for stamping	Alex	11/5/2014	6/18/2014	COMPLETE First iteration of design was ordered. Parts for the board are known, and will be ordered within the weekend. Future iterations of design will merge the EVM functionality to the custom design.
3.4 Spice Transient Analysis	Alex	10/20/2014	9/22/2014	COMPLETE Transient analysis is complete for both the buck and secondary boost converter. Levels are attainable for both the 6-10V unregulated (Pixy) and 3-3.3V regulated (XBee) ranges per transient simulations produced by TI's WEBDESIGN application.
4. Windbelt power module construction and testing		11/26/2014	6/30/2014	COMPLETE
4.1 Solder on components	Alex	11/26/2014	6/29/2014	COMPLETE
4.2 Continuity tests	Alex	11/26/2014	6/30/2014	COMPLETE
5. Server/Gateway setup	Seth	10/10/2014	7/1/2014	COMPLETE The server is a Raspberry Pi located at ctsn.student.rit.edu.

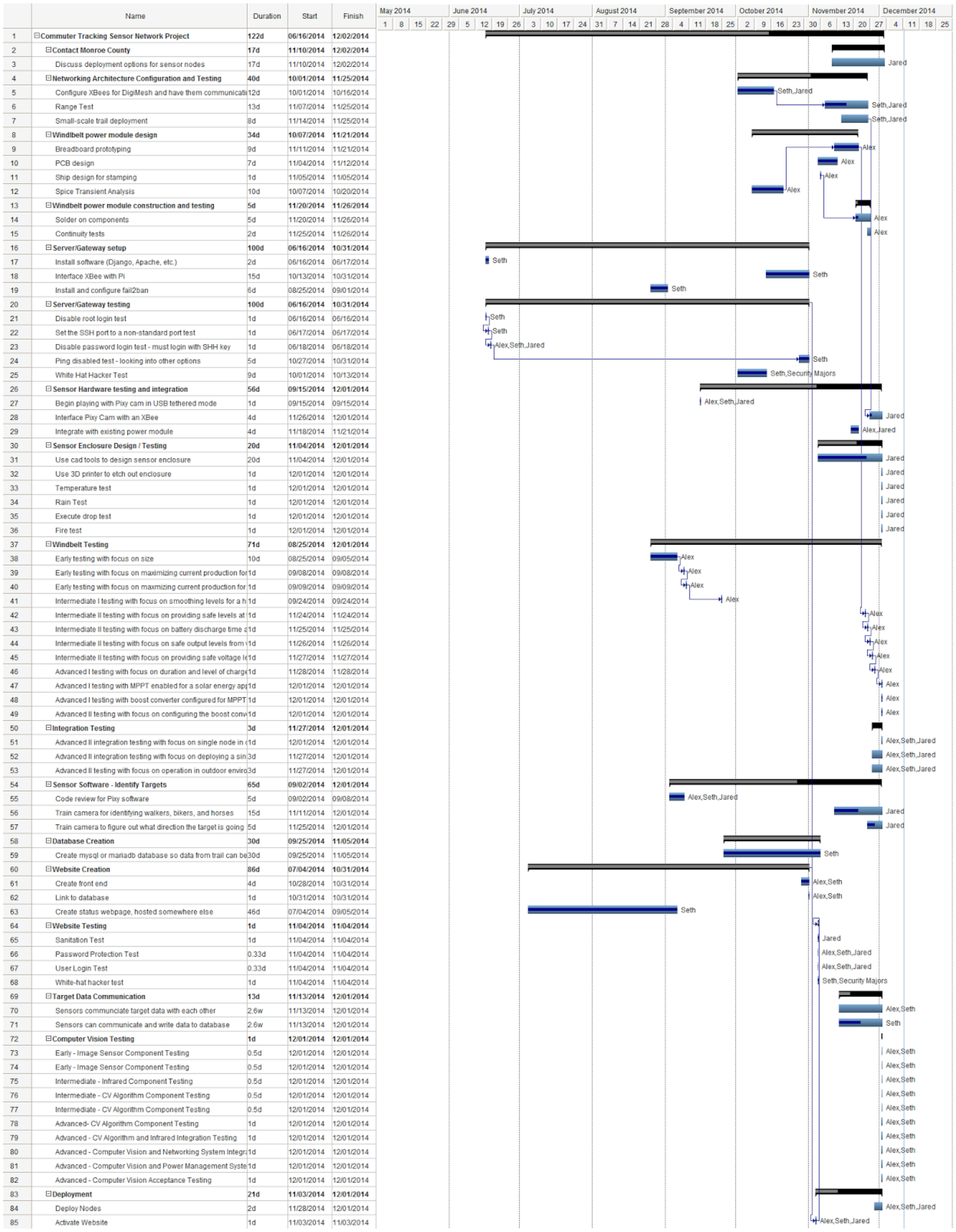
5.1 Install software (Django, Apache, etc.)	Seth	6/21/2014	6/17/2014	COMPLETE Apache, Django, MariaDB are installed and ready to go.
5.2 Interface XBee with Pi	Seth	10/31/2014	7/1/2014	COMPLETE Are able to Tx and Rx with the XBees between two Pis.
5.3 Install and configure fail2ban	Seth	9/1/2014	6/21/2014	COMPLETE
6. Server/Gateway testing		10/12/2014	7/1/2014	COMPLETE
6.1 Disable root login test	Seth	6/21/2014	6/16/2014	COMPLETE Done automatically when Raspbian was updated
6.2 Set the SSH port to a non-standard port test	Seth	6/21/2014	6/17/2014	COMPLETE SSH Port is set to 1315, not the default port of 22
6.3 Disable password login test - must log in with SSH key	Alex, Jared, Seth	9/5/2014	6/21/2014	COMPLETE SSH Keys are required to login to the server via SSH
6.4 White Hat Hacker Test	Seth	10/12/2014	6/21/2014	COMPLETE They could not access the server via SSH, get a root shell, or access the database directly. Jared (security major) will be providing a formal report of the pen test results. We will be able to fortify security based on the results.
6.5 Ping disabled test	Seth, Security Majors	R2	6/21/2014	DEFERRED While the server is on the RIT campus, this is completed since outsiders cannot ping rit.edu. If the server moves off campus for whatever reason, this will need to be revisited
7. Sensor hardware		R2	7/11/2014	

testing and integration				
7.1 Begin playing with Pixy Cam in USB tethered mode	Jared, Alex, Seth	7/11/2014	5/1/2014	COMPLETE We've all experimented and interfaced with the PixyCam now, and familiarized ourselves with its basic operation.
7.2 Interface Pixy Cam with an XBee	Jared	R2	6/22/2014	The PixyCam was more difficult than originally thought. This did not make it.
7.3 Integrate with existing power module	Jared, Alex	11/21/2014	7/11/2014	COMPLETE
8. Sensor Enclosure Design / Testing		11/14/2014	8/7/2014	
8.1 Use CAD tools to design sensor enclosure	Jared	12/1/2014	7/1/2014	COMPLETE
8.2 Use 3D printer to print the enclosures	Jared	12/1/2014	7/15/2014	COMPLETE
8.3 Test (See Gantt Chart)	Jared	R2	8/7/2014	
9. Windbelt Testing	Alex	11/21/2014	5/27/2014	COMPLETE Testing has been completed with an 800mV AC signal while driving both the PixyCam and the XBee through the battery management system. The battery holds charge for extended periods of time.
10. Sensor Software - Identify targets		10/24/2014	9/1/2014	
10.1 Code Review for Pixy Software	Alex, Seth, Jared	9/8/2014	9/8/2014	COMPLETE Code review was completed. Information was documented regarding each file's contents.

10.2 Compile GCC version of Pixy software and note differences			9/8/2014	No longer a requirement. Keil will work just fine.
10.3 Train camera for identifying walkers, bikers, and horses	Jared	12/1/2014	8/1/2014	For R1, the Camera was not trained to identify entities. Instead, the picture was sent across the network to the big computer, which handled the CV. For R1, only pedestrians were trained.
10.4 Train camera to figure out what direction the target is going	Jared	R2	9/1/2014	
11. Database Creation		9/23/2014	9/14/2014	COMPLETE
11.1 Create MySQL or MariaDB database so data from trail can be saved to it	Seth	11/5/2014	9/5/2014	COMPLETE The database is running on the gateway node. It is MariaDB.
12. Website Creation		9/26/2014	9/28/2014	COMPLETE Final website is located at https://ctsn.student.rit.edu
12.1 Create status webpage, hosted somewhere else	Seth	9/5/2014	9/5/2014	COMPLETE Status webpage that pings the gateway is functional. Its currently hosted on one of Seth's Pis off campus, located at http://people.rit.edu/~srh7240/ctsn_status .
12.2 Create web front end	Seth	10/31/2014	9/14/2014	COMPLETE
12.3 Link website to database	Seth	11/5/2014	9/21/2014	COMPLETE Website displays the results from the database.

13. Website Testing (See Gantt Chart)	Team	10/12/2014	10/4/2014	COMPLETE Jared (security major) provided a formal report of the pen test results. We were able to Fortify security based on the results.
14. Target Data Communication		11/21/2014	10/5/2014	
14.1 Sensors communicate target data with each other	Seth	R2	10/4/2014	Did not make it. The fact that we are unable to set the XBees address makes this task difficult to dynamically add or remove nodes to the trail. Although it can be done given more time, right now it is not feasible. It is a "nice to have" feature, and is not a show-stopper.
14.2 Sensors can communicate and write target data to database	Jared, Seth	R2	10/5/2014	Gateway side is done. Only need to do the PixyCam side.
15. Computer Vision Testing (See Gantt Chart)	Jared, Seth	R2	10/28/2014	Dependent on 10.3 and 10.4
16. Deployment		R2	11/9/2014	
16.1 Deploy nodes on trail	Team	R2	11/5/2014	Did not make it.
16.2 Activate website	Team	9/27/2014	11/9/2014	COMPLETE Website is located at https://ctsn.student.rit.edu (login required)
17.Integration Testing		11/13/2014		
17.1 Advanced II integration testing with focus on single	Team	R2		The XBee and the Pixy Cam were connected to the power module. The power module was able to power both at the same

node in controlled environment				time successfully. However, the Pixy Cam was not able to run the CV algorithms, so the test should be redone when the CV algorithm is working on it so it is a proper integration test.
17.2 Advanced II integration testing with focus on single node in an outdoor environment	Team	R2		Didn't make it this far.
17.3 Advanced II testing with focus on operation in outdoor environment for multiple nodes	Team	R2		Didn't make it this far.



	Name	Duration	Start	Finish
1	**⊟Commuter Tracking Sensor Network Project**	**122d**	**06/16/2014**	**12/02/2014**
2	**⊟Contact Monroe County**	**17d**	**11/10/2014**	**12/02/2014**
3	Discuss deployment options for sensor nodes	17d	11/10/2014	12/02/2014
4	**⊟Networking Architecture Configuration and Testing**	**40d**	**10/01/2014**	**11/25/2014**
5	Configure XBees for DigiMesh and have them communicati	12d	10/01/2014	10/16/2014
6	Range Test	13d	11/07/2014	11/25/2014
7	Small-scale trail deployment	8d	11/14/2014	11/25/2014
8	**⊟Windlbelt power module design**	**34d**	**10/07/2014**	**11/21/2014**
9	Breadboard prototyping	9d	11/11/2014	11/21/2014
10	PCB design	7d	11/04/2014	11/12/2014
11	Ship design for stamping	1d	11/05/2014	11/05/2014
12	Spice Transient Analysis	10d	10/07/2014	10/20/2014
13	**⊟Windbelt power module construction and testing**	**5d**	**11/20/2014**	**11/26/2014**
14	Solder on components	5d	11/20/2014	11/26/2014
15	Continuity tests	2d	11/25/2014	11/26/2014
16	**⊟Server/Gateway setup**	**100d**	**06/16/2014**	**10/31/2014**
17	Install software (Django, Apache, etc.)	2d	06/16/2014	06/17/2014
18	Interface XBee with Pi	15d	10/13/2014	10/31/2014
19	Install and configure fail2ban	6d	08/25/2014	09/01/2014
20	**⊟Server/Gateway testing**	**100d**	**06/16/2014**	**10/31/2014**
21	Disable root login test	1d	06/16/2014	06/16/2014
22	Set the SSH port to a non-standard port test	1d	06/17/2014	06/17/2014
23	Disable password login test - must login with SHH key	1d	06/18/2014	06/18/2014
24	Ping disabled test - looking into other options	5d	10/27/2014	10/31/2014
25	White Hat Hacker Test	9d	10/01/2014	10/13/2014
26	**⊟Sensor Hardware testing and integration**	**56d**	**09/15/2014**	**12/01/2014**
27	Begin playing with Pixy cam in USB tethered mode	1d	09/15/2014	09/15/2014
28	Interface Pixy Cam with an XBee	4d	11/26/2014	12/01/2014
29	Integrate with existing power module	4d	11/18/2014	11/21/2014
30	**⊟Sensor Enclosure Design / Testing**	**20d**	**11/04/2014**	**12/01/2014**
31	Use cad tools to design sensor enclosure	20d	11/04/2014	12/01/2014
32	Use 3D printer to etch out enclosure	1d	12/01/2014	12/01/2014
33	Temperature test	1d	12/01/2014	12/01/2014
34	Rain Test	1d	12/01/2014	12/01/2014
35	Execute drop test	1d	12/01/2014	12/01/2014
36	Fire test	1d	12/01/2014	12/01/2014
37	**⊟Windbelt Testing**	**71d**	**08/25/2014**	**12/01/2014**
38	Early testing with focus on size	10d	08/25/2014	09/05/2014
39	Early testing with focus on maximizing current production for	1d	09/08/2014	09/08/2014
40	Early testing with focus on maxmizing current production for	1d	09/09/2014	09/09/2014
41	Intermediate I testing with focus on smoothing levels for a h	1d	09/24/2014	09/24/2014
42	Intermediate II testing with focus on providing safe levels at	1d	11/24/2014	11/24/2014
43	Intermediate II testing with focus on battery discharge time a	1d	11/25/2014	11/25/2014
44	Intermediate II testing with focus on safe output levels from	1d	11/26/2014	11/26/2014
45	Intermediate II testing with focus on providing safe voltage le	1d	11/27/2014	11/27/2014
46	Advanced I testing with focus on duration and level of charge	1d	11/28/2014	11/28/2014
47	Advanced I testing with MPPT enabled for a solar energy ap	1d	12/01/2014	12/01/2014
48	Advanced I testing with boost converter configured for MPPT	1d	12/01/2014	12/01/2014
49	Advanced II testing with focus on configuring the boost conv	1d	12/01/2014	12/01/2014
50	**⊟Integration Testing**	**3d**	**11/27/2014**	**12/01/2014**
51	Advanced II integration testing with focus on single node in	1d	12/01/2014	12/01/2014
52	Advanced II integration testing with focus on deploying a sin	3d	11/27/2014	12/01/2014
53	Advanced II testing with focus on operation in outdoor enviro	3d	11/27/2014	12/01/2014
54	**⊟Sensor Software - Identify Targets**	**65d**	**09/02/2014**	**12/01/2014**
55	Code review for Pixy software	5d	09/02/2014	09/08/2014
56	Train camera for identifying walkers, bikers, and horses	15d	11/11/2014	12/01/2014
57	Train camera to figure out what direction the target is going	5d	11/25/2014	12/01/2014
58	**⊟Database Creation**	**30d**	**09/25/2014**	**11/05/2014**
59	Create mysql or mariadb database so data from trail can be	30d	09/25/2014	11/05/2014
60	**⊟Website Creation**	**86d**	**07/04/2014**	**10/31/2014**
61	Create front end	4d	10/28/2014	10/31/2014
62	Link to database	1d	10/31/2014	10/31/2014
63	Create status webpage, hosted somewhere else	46d	07/04/2014	09/05/2014
64	**⊟Website Testing**	**1d**	**11/04/2014**	**11/04/2014**
65	Sanitation Test	1d	11/04/2014	11/04/2014
66	Password Protection Test	0.33d	11/04/2014	11/04/2014
67	User Login Test	0.33d	11/04/2014	11/04/2014
68	White-hat hacker test	1d	11/04/2014	11/04/2014
69	**⊟Target Data Communication**	**13d**	**11/13/2014**	**12/01/2014**
70	Sensors communciate target data with each other	2.6w	11/13/2014	12/01/2014
71	Sensors can communicate and write data to database	2.6w	11/13/2014	12/01/2014
72	**⊟Computer Vision Testing**	**1d**	**12/01/2014**	**12/01/2014**
73	Early - Image Sensor Component Testing	0.5d	12/01/2014	12/01/2014
74	Early - Image Sensor Component Testing	0.5d	12/01/2014	12/01/2014
75	Intermediate - Infrared Component Testing	0.5d	12/01/2014	12/01/2014
76	Intermediate - CV Algorithm Component Testing	0.5d	12/01/2014	12/01/2014
77	Intermediate - CV Algorithm Component Testing	0.5d	12/01/2014	12/01/2014
78	Advanced- CV Algorithm Component Testing	1d	12/01/2014	12/01/2014
79	Advanced - CV Algorithm and Infrared Integration Testing	1d	12/01/2014	12/01/2014
80	Advanced - Computer Vision and Networking System Integra	1d	12/01/2014	12/01/2014
81	Advanced - Computer Vision and Power Management Syste	1d	12/01/2014	12/01/2014
82	Advanced - Computer Vision Acceptance Testing	1d	12/01/2014	12/01/2014
83	**⊟Deployment**	**21d**	**11/03/2014**	**12/01/2014**
84	Deploy Nodes	2d	11/28/2014	12/01/2014
85	Activate Website	1d	11/03/2014	11/03/2014

Figure 63 - Gantt Chart

Perspective

Power Module

Although the learning curve for prototyping and creating a custom PCB design was large, it was extremely satisfying once the design was complete, and testing showed that it worked. Learning EAGLE is a valuable experience that will come to use in many future situations. It was also interesting to see how SMT components are soldered to a board, and the various reflow techniques that are used in industry. There is a great amount of manual skill involved in placing solder paste on a board, laying down the components, and melting and cooling the solder without blowing chips off of the board. Thankfully, resources were available on campus to advise on some of the subtleties of the trade.

Future iterations of the PCB will include an XBee mount that will allow for interfacing 20 pins with the board. The mount will be routed to an 8-pin interface that will be used to attach to a Pixy camera module. Another 8-pin connector will be used to attach the IR module, and a wake on detect pin will be connected to the power pin for both the Pixy camera and the XBee. This will allow for the devices to be in sleep mode during low traffic periods. When a commuter passes by, the infrared module will set the wake pin, turning on the peripherals needed to perform the tracking algorithms. The PCB will also have two-pin connectors to attach the windbelt to the rectifier, the boost converter to the battery, and the Pixy camera to the secondary boost converter. A BNC connector will allow for an antenna to be attached to the XBee, and be positioned outside of the node enclosure.

Networking

Overall, the networking went fairly smoothly. Almost everything went as expected when the project was originally designed. The trickiest part was getting the XBees working correctly. The XBees require any messages sent over using them to be in a certain format. The general format is a start character, the length, the address to send to, a few options, the payload and a checksum. The XBee has a few characters that need to be escaped, as they mean something special to the XBee firmware. One of the XBee's mac addresses had one of these characters in it, so we had to use the XBees with escape characters enabled. The escape characters proved to be a tad annoying since the character that needed to be escaped need a '}' character before it, and it needed to be xored with 0x20. The escape characters ('}') also did not get added to the length of the packet, nor were added to the checksum. In addition, escaped characters that were xored with 0x20 needed to be added to the checksum before being xored. Although this was annoying, and took longer than originally thought, we did get it working.

One thing that did not make it into this iteration of the project was a common operating picture across the trail nodes. This task would have been difficult to do as every trail node would need to know the address and location of every other node on the trail, and have some way to get updated information about this just in case a node went down, or was taken off of the trail. Although this could be done given a lot more time, it would not be an easy task.

Website

The website did not quite have as many features as was originally planned for. For example, we wanted the data to be sorted by date, by time, by node, along with various other filtering schemes. There is also no way for a user to reset his or her password themselves. Also, the website looks only okay on a mobile device. Once the basic functionality of the website was in place, the network architecture needed to be worked on more. It might have helped to have had a fourth person who only worked on the website and database. That would have freed Seth up to make the network architecture better.

Image Processing

The biggest hurdle that we encountered was working with the PixyCam system. We planned on a major rewrite of the firmware of a fairly complex system. As the PixyCam is an open source project there is little support and virtually no documentation. The community remains small at this stage and as such there isn't the level of information or documentation that one typically finds with a system of this depth and size. Because of this lack of formal support and documentation it was very difficult to accomplish our goals. Simple questions that would typically be addressed in code documentation could only be answered by experimentation. These conditions drastically slowed development. We ultimately had success implementing the system utilizing the PiCam because of the extensive community, example code, and documentation that went along with that system.

Knowing what we know now other hardware would have been selected for the image acquisition system. Something from a professional vendor instead of a small open source project. At least 50% of the value of the hardware you are purchasing is in the documentation and support that typically comes along with that purchase. These characteristics of component selection will be a greater consideration in the future as a result of this experience.

The Future

We've entered the project into the Texas Instruments 2015 Design Competition. As such we will be continuing development and refinement of the system. With the additional time we should be able to fully integrate the PixyCam into the system as originally envisioned.

Prior to this we would like to test the time of operation with our current configuration. A model A Raspberry Pi only has a power consumption of approximately 115mA and approximately 200mA added with the PiCam. This is about 50% higher that the PixyCam specs but could still provide a long enough of operation time to gather the needed data. A redesign of the enclosure would be needed but that is a relatively simple proposition.

Additional memory needs to be added to the PixyCam system to make it usable for the CV applications that we desire to use it for. There are SPI memory modules that can easily be added to the system to interface with the PixyCam. Having this additional memory to flash programs to and to perform the CV computation in will be very valuable to not only accomplishing what we want but in diversifying the system to perform a variety of tasks. Additional memory would allow us to offer other services on the nodes themselves besides the feature detection.