Theia

Authors: Evan Compton, Electrical and Computer Engineering, Carnegie Mellon University
William Mitchell, Electrical and Computer Engineering, Carnegie Mellon University
Allison Scibisz, Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—When walkers and runners listen to music on their phone, they become distracted and disconnected from the world around them, and this lack of awareness can lead to trail congestion, unexpected collisions, and even dangerous motor vehicle accidents. Theia is a wearable system that detects an object approaching the user from behind, classifies it as either a car, a biker, or a pedestrian, and informs the user through their headphones. Three LiDAR sensors and a Raspberry Pi Cam are mounted on the user's back for object detection and recognition, with the signal and image processing occurring on a Raspberry Pi held in a waist belt. User notification occurs through the Theia iOS application, which communicates with the Raspberry Pi through a Bluetooth Low Energy (BLE) connection. The system is powered using a 4000mAh Lithium Polymer Battery, with a total system battery life of over 7 hours.**

***Index Terms*—BLE, Image Processing, k-NN LiDAR, Wearable**

I. Introduction

PEDESTRIANS in the twenty-first century are plugged in, listening to music or podcasts, hearing the news, and making phone calls whenever they walk and run. While this can promote exercise and increase productivity, it decreases the pedestrian's awareness of the environment around them, making them susceptible to accidents with cars, bikers, and other pedestrians. The goal of Theia is to help these pedestrians by informing them about potentially dangerous entities approaching them outside of their visual field. Theia is a piece of wearable technology to be used by a walker or runner to wear that communicates with their cellphone via Bluetooth, informing them through their headphones when there is something coming towards them from behind and what it is, allowing them to be safer and more aware of their surroundings without having to sacrifice the use of their headphones. The goal of Theia is to increase cognizance in modern pedestrians through a tool that can be easily integrated with their existing devices.

II. Design Requirements

Our primary goal is to create a device that accurately reports objects approaching the user from directly behind them and inform them before that object passes. In order to provide the user with this information in enough time, Theia needs to detect approaching objects that are up to 8-10 meters behind the user. In an average case, we anticipate a roughly 2 meters per second velocity differential between the user and the approaching object, so detection at 10 meters away gives Theia 5 seconds to process sensor data and inform the user before the object passes in this average case. Our device will handle objects approaching with other speed differences, this requirement was designed to best suit our average case. We also have a requirement that Theia should only report objects getting closer to the user rather than detecting everything within 8-10 meters, as the user usually will not need to be informed of objects that they recently passed or other objects that will never pass them. In addition, we want to detect any object that enters the field of view behind the user as close as 1 meter away, so if a runner turns onto the street 5 meters behind the user, the user will still be informed. 8-10 meters will simply serve as our requirement for upper limit for detection. For accuracy, we set our requirement to inform users about approaching objects at least 90% of the time. We determined that anything much less than this would not be a worthwhile tool for the user, as it would be too unreliable.

Our secondary goal is for our device to have the functionality of object recognition, being able to differentiate between pedestrians, bikers, and cars. This task is difficult to achieve with high accuracy while maintaining the low latency and low power consumption required for the system. Therefore, we only require 50% accuracy in this process. We assume that the user will turn around when an approaching object is approaching, so while we would like to have great accuracy in object recognition so that the user can best know what to expect, this is not as crucial as accurate object detection.

For informing the user, we require the user to be alerted through their headphones, with an auditory alert briefly describing the approaching object. This alert should either interrupt or decrease the volume of the background audio, returning it to its normal state after the alert has finished. This alert should be received to the phone wirelessly, allowing the user to maintain a full range of motion throughout their exercise and move their phone as they please without worrying about a wired connection to the rest of Theia. We also require complete cycle from initial object detection to alerting the user to be completed in under 5 seconds, as it is important for the user to be informed before the object passes, which takes approximately 5 seconds in the average case. The shorter the latency is, the better, as it allows the user to have more time to respond to the approaching object.

We require our device to be battery powered and not dependent on other resources, like WiFi, that would not be available to most runners or walkers outdoors. Our battery needs to last for at least one average use case, which we marked as at least 45 minutes. We also maintain a couple implicit goals regarding the comfort and usability of Theia. Theia should be lightweight and comfortable to wear, not impeding the running or walking process. The Theia app should also be user friendly, allowing those not involved with the project to use the product with ease.

III. Architecture and/or Principle of Operation

The basic architecture of our product includes multiple LiDAR sensors and a camera interfacing with a Raspberry Pi to detect approaching objects, which will then transmit information via Bluetooth to the user's cellphone, which emits an alert to the user. Specifically, we used three TFMini - Micro LiDAR sensors, which are arranged at 4.5° relative angles on the user's back to produce a wider field of view, and a Raspberry Pi Camera Module V2, all of which are mounted on a custom 3D printed board and attached to a GoPro camera harness to be worn on the user's back. These sensors are connected to a Raspberry Pi 3 Model B and all of these are powered by a 4000mAh Lithium Polymer Battery. The Raspberry Pi and the battery are kept in a running hydration belt, also worn by the user.

When the sensors determine that an object is approaching from behind the user, the camera is triggered to take a picture. This image is then processed on the Raspberry Pi, where the approaching object will be detected and classified as either a car, a biker, or a pedestrian. Next, this information is transmitted to the user's iPhone app via Bluetooth. Finally, the user will be alerted about the object which is approaching with an audio message played over their headphones. The overall architecture has not changed since the Design Review Report. This system is visualized in Figure 1.

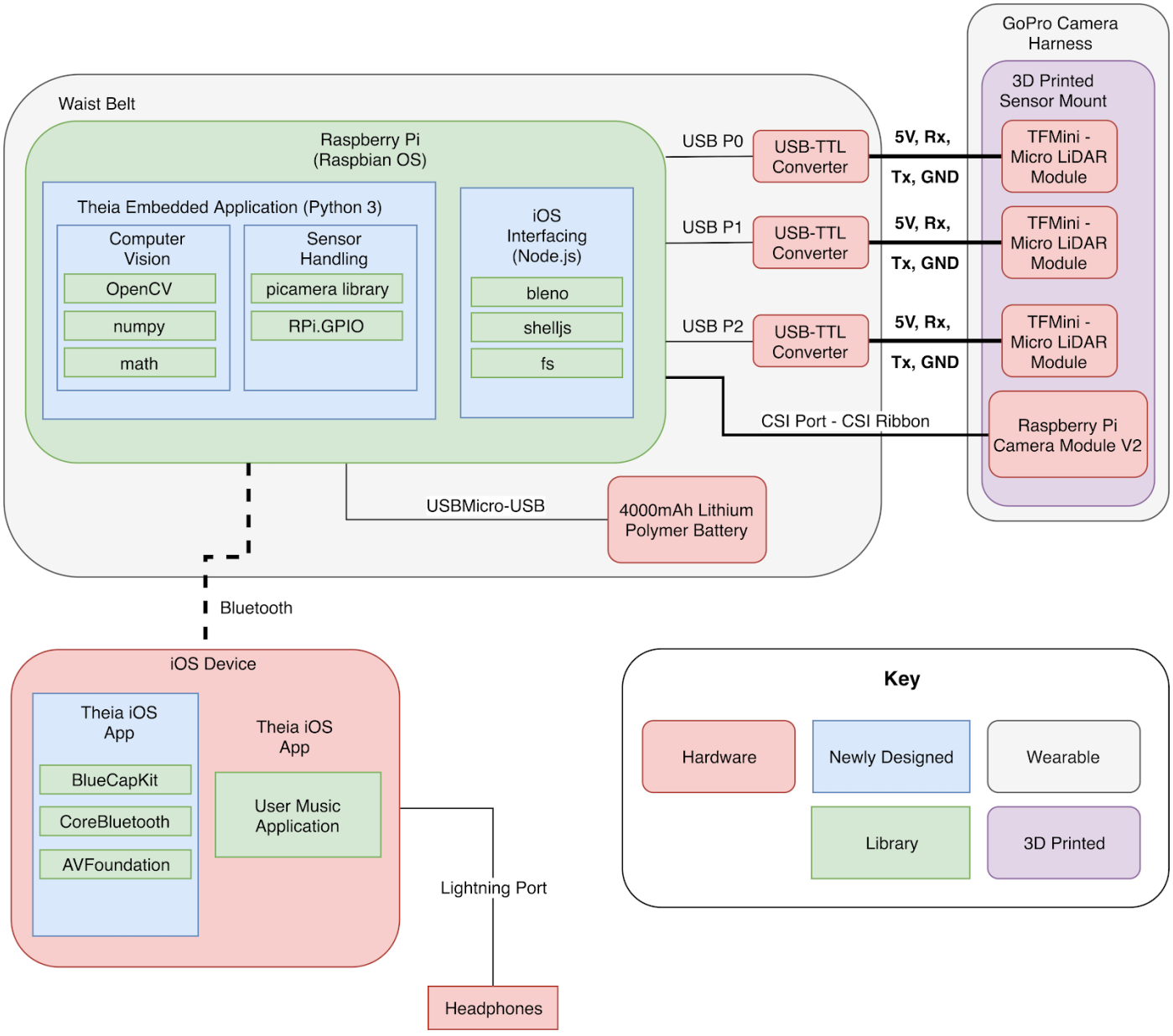

Figure 1: Complete Block Diagram

IV. Design Trade Studies

The major design decisions for our project included the sensor choices, the machine learning choices, the device choices, and communication protocol. Each decision affected how well our final device would meet our design requirements, so we took into account many possibilities in our decision making process.

A. Sensors

We initially looked into using ultrasonic sensors for object detection, but we quickly determined that the range on the ultrasonic sensors we could afford was insufficient and their field of view was too wide, making it impossible for us to focus on objects directly behind the user without picking up a lot of noise which could trigger false positives. We decided to switch to LiDAR because of their better long distance range, as we required our device to be able to pick up objects approaching from up to 10 meters away. With a budget of $600, the best LiDAR sensor in our price range was the Garmin LiDAR-Lite v3, with great reviews and costing $130. With a 40m range, these far exceeded our distance goal, but with an 8 mRadian field of view, they could truly only detect along a very thin line, and we did not want our product to only work if the approaching object was in a super specific line behind the user. If we could have several of these sensors at slightly different angles, we could achieve the field of view we wanted, but we could not afford it. Instead, we went with the Benewake TFMini Micro LiDAR, which has a larger 2.3 degree field of view with an advertised 12 meter range costing roughly $40 each. Using three of these LiDAR sensors at slightly different angles, we were able to achieve a 11 degree field of view and detect objects 10 meters away, so this was the best choice for us. Unfortunately, we later realized that the range of these sensors diminished greatly in direct sunlight, but of the options we found, this was still the best solution in our budget.

B. Camera and Image Capture

We chose to have a camera triggered by LiDAR sensors rather than constantly streaming video, because this approach drains less power, both in terms of the camera use and the frequency of heavy computation being done. When choosing cameras, we had a Raspberry Pi Camera Module V1 available to us, which took decent quality photos, but we decided to buy the Raspberry Pi Camera Module V2 to use instead because of its improved resolution and faster shutter speed. We were happy with this decision in the end, as the images with the new camera were much higher quality, so it was worth the minor increase in cost.

C. Machine Learning

For the Image processing part of the project, we chose the approach of extracting features and using a fast machine learning algorithm like k-NN, rather than a deep learning approach, because of the speed and power benefits. Deep learning for image classification would have been a good bit slower on the Raspberry Pi than the .73 seconds it takes with k-NN, and would have been more power intensive and thus drained the battery quickly, despite being more accurate.

D. Raspberry Pi

We decided to use a Raspberry Pi for the bulk of our computation because it is small, which we require for our wearable, it has good processing ability, which was important for the functionality of our project, and it has numerous ports, which is necessary for us since we are using multiple sensors. Raspberry Pis are also widely used and fairly well documented, so it seemed like a safe choice. We initially chose to use a Raspberry Pi 3 Model B because we had one available to us, meaning that we could concentrate our budget on other items. We also liked that it had Bluetooth functionality, which was our ideal method for communicating with the user's cellphone. Unfortunately, the Raspberry Pi did not have enough UART ports for our three LiDAR sensors, so we decided to use three WITMOTION USB-UART 6-in-1 Multifunctional Serial Adapters, allowing us to receive data from the LiDAR sensors through three of the Pi's four USB ports. These adapters did have more functionality than we needed, but it was the only option that was both available reasonably soon that had overwhelmingly positive reviews about their functionality and durability.

E. Communication

For transmitting data between the iOS device and the Raspberry Pi, we decided to use Bluetooth Low Energy (BLE) communication. We decided that communication over WiFi, which is one of the most common and well documented methods for communicating between a Raspberry Pi and another device, is not ideal for us, as we want Theia to work even when the user is not in an area with WiFi. We also did not want to have a wired connection from the iPhone to the Pi, as we want the user to be able to hold their phone however they prefer, without being limited by a wire connected to the wearable. The Raspberry Pi 3 Model B comes with Bluetooth built in, so BLE seemed like the best path to go. We had initially planned to use the built in BLE services for communication, but they did not suit our purposes. Bluetooth communication proved to be more difficult than any of us anticipated, but we found that the Bleno Node.js module allowed us to create custom services on the Raspberry Pi with relative ease and this path suited our purposes. By connecting the swift application with the UUID of the BLE service we created, the iOS device was able to seamlessly connect with the Theia wearable and communication between the two could happen in under 0.1 seconds. This met our requirement to have an app that is user friendly, as the user does not have to do anything related to the Bluetooth communication other than make sure their iPhone's Bluetooth is on and that the Theia device is on and in range. It also met our latency requirement, as this form of communication is very quick.

Figure 2: (a) Design for 3D printed mounting board (b) sensor harness (c) hydration belt for battery and Raspberry Pi

V. System Description

A. Wearable Design

The wearable design is made up of two major components: the sensor harness and the load bearing belt. The sensor harness (Figure 2b) is designed to house the sensors on the users back while focusing on maintaining stability for accurate sensor readings. We purchased a CamKix Chest Mount Harness and used it backwards on the users back. We chose this style of harness for its proven stability in mounting GoPro Cameras with an active user. Since our sensors and mounting board are significantly lighter than the GoPro the harness is designed for, found similar results in stability with our Theia sensors. To attach our four sensors (described below) to the harness, we created a custom 3D printed a mounting board (Figure 2a). The mounting board is attached using GoPro's standard mounting single-pivot clips and has fixtures to bolt our four sensors. We chose to 3D print with SLA filament for this mount board because of our need for a light, sturdy material with which we can mount sensors at varying angles.

The second wearable is the load bearing belt (Figure 2c). We choose to use a manufactured Neoprene Running Hydration Belt because of its proven success in carrying water bottles and other heavy devices for runners. This device is designed to carry loads above 550 grams comfortably around the hips of runners. The historic success of this device in carrying larger weights comfortably for runners makes it ideal to manage holding our heavy components, which includes the Raspberry Pi, Lithium Polymer Battery, and three WitMotion USB-UART Converters. These devices weigh far less than the 550 grams of water the belt is designed to hold, allowing the belt to be worn very comfortably for walkers and runners. These devices are securely attached to the belt and wired to the chest harness with the appropriate sensor connections. We used zip ties to hold the wires in place and prevent them from blocking the sensors for our final demo. The entire final wearable system is shown in Figure 3.

B. Sensor Design

The wearable contains four sensors: three TFMini - Micro LiDAR Module and one Raspberry Pi Camera V2 Module. The optical sensor is for Image processing/object classification, while the three LiDAR sensors are used in conjunction for real time detection of approaching objects behind the user. The LiDAR sensors where set to distance mode where they are optimized for detecting objects 3 to 12 meters away and the sensors provide a distance value in cm with a strength of accuracy value every 10ms.

Our goal in approaching object detection was to quickly determine that an object was approaching the user from behind, while filtering out objects the user was passing by. To do this we developed a python script that used a non-blocking round-robin scheduling scheme to read the three LiDAR sensors continuously. Then the minimum valid value of the three at any given time step is recorded as a data point for that timestep. If this data point has below a certain (settable parameter) strength threshold, or the data point is an outlier relative to neighbor data points, it is ignored/remove from the next portion of the algorithm.

After we filter out bad data points we attempt to determine if the object that the sensor is picking up is approaching the user. To do this we sample every X distance points and

Figure 3: Final wearable prototype

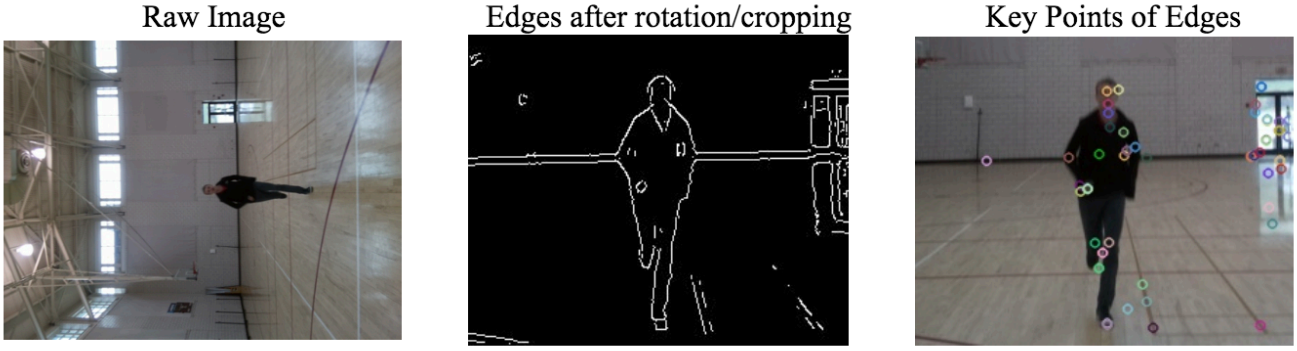

Figure 4: Example workflow of the object recognition algorithm. This image has selected sections for both the biker and the pedestrian classes. Since the distribution of key points is closer to the pedestrian training data than the biker, pedestrian is selected.

determine if there is negative velocity (relative to user and detected object). If the object were approaching for Y points of the X samples then we say we detected an approaching object. The outlier sensitivity, number of samples (X), necessary number of approaching data points (Y), and minimum required strength value are all changeable parameters used to tune our sensor algorithm to balance the latency of detection, limiting false positives, and accurately detecting an approaching object.

For example, increasing the sample size increased latency while improving elimination of false positives. Increasing the number of required approaching data points would decrease false positives but would also increase true negatives. We found these relations nonlinear and also dependent on environment of use.

C. Image Processing

For the image processing part, our goal was at least reasonable accuracy (for predominantly blurry images), with speed and power prioritized. We had three classes for our image processing: pedestrians, bikers and cars. Our approach was to use OpenCV to extract features from the image, and then use a k-Nearest Neighbors based machine learning algorithm (with online training data) to pick the most likely class. K-Nearest Neighbors was picked over other machine learning algorithms (notably a deep learning approach) because it is the best in terms of speed and power.

After receiving an image from the camera, the first step in our pipeline is cropping the image to include only the part where our LiDAR sensors would have detected an approaching object. Next, we use OpenCV to get the edges, and find the SIFT key points of the edges. The key points were used as the basis for features because they really capture the geometrical differences between the different objects we are trying to detect. For example, the key points for a person tend to be around the head, the arms, and legs, whereas the key points for a car tend to be around the headlights. Once we have the key points for the cropped area, we divide it into sections and find the most similar parts to a car, a biker, and a pedestrian. We look at 5 sections for pedestrians and bikers (the 5 columns of the image), and 3 for cars (left half, right half, and middle section), based on about how big the objects usually are when the camera takes a picture.

For each section of the image, we extract features from the key points, and then use those features to determine how similar the section is to the objects we are looking for. For extracting features from the key points, we decided to break each section it into a 3 column by for 4 row grid and figure out the percentage of key points in each block. We then end with a 12-element feature vector for each section. This feature extraction works reasonably well because the geometries are very different between these objects. After extracting features from an image, we go through the training data and pick the closest 3 training vectors (k=3) for each relevant class (just cars for the car sections and pedestrians and bikers for their joint sections) based on Euclidean distance. At the end we pick the overall closest section for each class by comparing distances. It's almost always accurate with finding the correct section of the image an object is in. We then compare the best sections together and pick the overall closest one as the selected class.

This works reasonably well (66.7% accuracy amongst our tests) and is not much worse with images that are a bit blurry. Overall this algorithm takes on average about .73 seconds to run on the Raspberry Pi (the bulk of which is extracting the key points). Taking the picture takes about .46 seconds, making the total time for image processing about 1.19 seconds. We believe the accuracy would have improved if we had time to gather training images from our wearable, as opposed to using online images. It is also notably better at selecting cars, as opposed to distinguishing between pedestrians and bikers, which makes sense because of the greater structural similarity between pedestrians and bikers than between either pedestrians or bikers and cars.

D. iOS Application

The user will interact with Theia though the Theia iOS application, which will be on user's cellphone that they are carrying and using to listen to audio (i.e. music) during their run or walk. The app will allow the user to connect their phone via Bluetooth to the Raspberry Pi on their belt, which is receiving information from the sensors on their vest. After connecting, the user can begin their session, during which they can listen to their own music or podcasts and will be alerted when an object is approaching from behind. In the

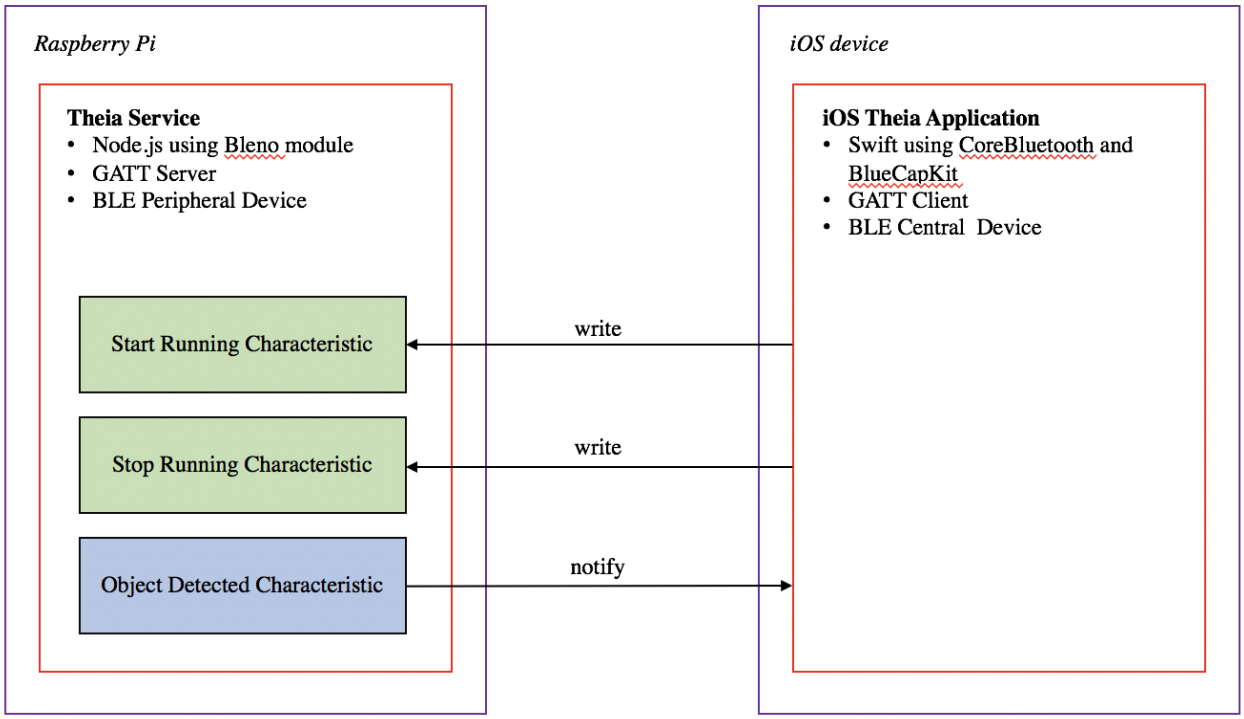

Figure 5: Bluetooth communication protocol between Raspberry Pi and iPhone

average use case, the user will be alerted through their headphones, with a vocal alert informing them that an object is approaching and indicating whether the approaching object is a car, a biker, or a pedestrian, if the information is available. This warning indicating an approaching object will also be displayed on the screen of the app. When the user can stop their session by hitting the stop button, and they will no longer receive notifications of approaching objects unless they hit start again.

This Theia application is written in Swift. While the Raspberry Pi 3 does come equipped with on-board Bluetooth, the available built in Bluetooth Low Energy (BLE) services and characteristics proved insufficient for transmitting data from the iPhone to the Raspberry Pi, as none of them had a write property. So, instead of using these built in BLE services as we had planned to in the design review report, we decided to create a Node.js program to run on the Raspberry Pi using the Bleno module, which allowed us to create a custom BLE service. The Node.js program makes and advertises our service, which has writable characteristics for start and stop and a notifying characteristic for reporting detected objects. In this system, the Raspberry Pi serves as the BLE peripheral and the iOS device will serve as the BLE central. We used the CoreBluetooth and BlueCapKit swift libraries to connect and communicate with the Raspberry Pi via Bluetooth. The iOS app writes to the start characteristic to indicate when the sensors should start running, writes the stop characteristic to indicate when the sensors should stop running, and gets notified by the object detected characteristic. The Node.js program will write a notification to the object detected characteristic of 0, 1, or 2 based on whether the object recognition process identifies a pedestrian, a biker, or a car, respectively. We used AVAudioPlayer from the AVFoundation framework to play the appropriate sound file for each of these situations to inform the user of the approaching object. We also used the duckOthers category option to turn down the volume in the background music while the notification is playing and restore the volume to full volume when it is finished. We also print the detected object on the app so that the user can visually see the last detected object on the screen if they miss the auditory notification. We chose to lower the volume of the background audio during the notification process rather than pause it because this creates a more seamless user experience. This is also the approach that Google Maps takes when providing driving instructions when background music is present, so we trusted their success and followed their lead. The entire process from the Node.js program learning of a newly detected object to the start of the audio clip informing the user takes less than 0.1 seconds. This communication protocol is illustrated in Figure 5.

VI. Validation and Results

We completed a significant amount of testing to determine the accuracy of our system in detecting and recognizing a variety of objects approaching a user from behind. All of our tests occurred outdoors at Carnegie Mellon University on the Engineering Quad, the Gesling Stadium track and football field, and the Donner Ditch parking lot in cloudy or partially cloudy conditions during daytime. For consistency, Evan

Compton was the test user for all tests.

For our first test, we tested just the functionality of the LiDAR sensor system in accurately detecting and reporting the existence approaching objects. We ran 90 tests, 10 in each of 9 different categories illustrated in Table 1. These tests took place largely in the absence of other obstacles, there was simply the user either standing still, walking, or running with their back to the given approaching object. This test was not designed to test for false positives, but a false positive during the running of a test was recorded as a failure, as it was not an accurate reading. Overall, we had 92% accuracy in correctly determining if an object was approaching in these trials.

TABLE I. ACCURACY IN OBJECT DETECTION

Approaching Object	**Theia User Status**		
	User Standing	***User Walking***	***User Running***
Pedestrian	100%	100%	80%
Bike	100%	80%	70%
Car	100%	100%	100%

For our second test, we were still testing solely the accuracy of the object detection process, not the object recognition process, but this test was designed to test the occurrence of false positives. We had the user wear the Theia wearable and pass other people who were not moving (while the user was walking or running), walking (while the user was running), and running (while the user was running faster or in the opposite direction). A successful trial in these test cases was one where nothing was detected. We ran 10 trials for each situation. These results are shown in Table 2. We had an overall success rate of 86%.

TABLE II. ACCURACY IN OBJECT DETECTION (TESTING FALSE POSITIVES)

Theia User Status		
User Standing	***User Walking***	***User Running***
100%	90%	70%

For our final test, we focused on object recognition. We ran 21 trials for this, with 7 of each type of object approaching, in various locations around campus. For each object type, we ran 2 tests with the user standing still, 2 with the user walking, and 3 with the user running. These results are shown in Table 3. We had an overall success rate of 67% in these trials.

TABLE III. ACCURACY IN OBJECT RECOGNITION

Approaching Object	**Theia User Status**		
	User Standing	***User Walking***	***User Running***
Pedestrian	50%	50%	33%
Bike	100%	50%	67%
Car	100%	100%	67%

We also tested battery life during all of these tests, continuously running the Theia program on the Raspberry Pi using the 4000mAh Lithium Polymer Battery. On average, our battery lasted 430 minutes (7 hours and 10 minutes), well exceeding our goal of a battery life of one average run or walk length of approximately 45 minutes.

VII. PROJECT MANAGEMENT

A. Schedule

Our schedule has changed some as we have progressed since the design review report. Getting the LiDAR sensors to work dependably and adjusting various parameters ended up taking longer than anticipated, while creating the algorithm for detecting approaching objects to trigger the camera at the appropriate time took less time. Establishing Bluetooth communication between the Raspberry Pi and the iOS application proved to be more complicated than anticipated, so that work extended beyond when we scheduled for it to be completed. Integration was also more challenging and took longer than planned. Fortunately, we had built enough slack time into our schedule that these minor setbacks did not prevent us from having a functional demo on time. Our most recently updated schedule is seen on page 9 in Figure 6.

B. Team Member Responsibilities

We divided the work such that each of the three of us will be primarily responsible for one area. Will was in charge of the sensors and the interface between the sensors and the Raspberry Pi. Evan was in charge of the object detection and recognition algorithms on the Raspberry Pi, using images from the Pi Cam. Alli was in charge of building the iOS application, interfacing with the Raspberry Pi via Bluetooth, and transmitting alerts to the user. While each person was primarily responsible for their designated section, we found that working together was necessary for many aspects of this project. For example, although Will was technically the one on sensors, we all worked together in testing the sensors and making decisions about which sensors to use as we moved forward, as these tasks go poorly when working alone. Similarly, when facing issues with Bluetooth communication, Alli sought input from Evan to solve some issues. Likewise, collaboration was needed when making significant design decisions to ensure that our different components would integrate properly. While most of the coding was completed independently, integration and testing was a group effort.

C. Budget

We have been able to stay safely under our $600 budget cap. Our complete bill of materials is on page 10 in Figure 7. The only change to our budget since the design review report was the purchase of an additional LiDAR sensor, as one of our LiDAR sensors stopped working and we had to purchase a new spare.

D. Risk Management

A semester long project that no one has ever done before, especially using tools we were not familiar with, inherently contains a high level of risk. To alleviate some of the risk due to uncertainty, we initially had several backup plans. This proved useful, as we initially planned to use ultrasonic sensors for detecting approaching objects before we realized that their range was insufficient, their field of view was too wide, and

they would interfere with one another if we used multiple sensors together. When it became clear that ultrasonic sensors were not the correct solution area, we had already done preliminary research on LiDAR sensors, so we simply jumped onto this idea and quickly determined that this was a better answer to our design problems. To mitigate other risks associated with unfamiliarity, we allowed significant time for research before we started implementation, we asked questions to others when we were working with new tools, and we worked together rather than individually on particularly difficult unfamiliar tasks.

There is also significant risk associated with purchasing parts that we haven't used before. To help mitigate this, we researched all of the items pretty extensively before we purchased them, reading reviews and spec sheets and ensuring that they were from reputable sources. In addition, we purchased spares when we were ordering parts, as we knew that parts break in the process of building and testing a prototype and we wanted a part failure to throw off our schedule. This added expensive paid off, as we did have sensors stop working, and having extras allowed us to continue working without waiting for replacements to come in.

VIII. Summary

Overall, we succeeded in creating the basic prototype that we had planned at the beginning of the semester. The full communication pipeline was successfully implemented and we hit our targets for accuracy and latency. Due to constraints in time, we were unable to reach some of our stretch goals, including being able to recognize approaching trains and implementing haptic feedback. If we had another week or so, this is what we would have implemented. Additionally, while we were able to meet our requirements for object detection distance in certain environments (like indoors or with sufficient cloud coverage), with the inexpensive LiDAR sensors we used, we found that the range became quite limited in direct sunlight. So, even though our latency was far shorter than our goal, our system was still not performing ideally in direct sunlight conditions when we could only detect approaching objects up to 4-5 meters away. We still believe LiDAR sensors are a strong candidate for this problem area, but the cheap class of LiDARs we utilized are not. If we had additional budget to better sensors, or if perhaps we had concentrated even more of our given budget on better LiDAR sensors, we could have achieved a longer range so that even in direct sunlight, we would be able to achieve our distance requirement.

Through our work with the LiDAR sensors, as well as various ultrasonic sensors that we were investigating early in the semester, we learned that inexpensive sensors are inexpensive for a reason. Often times, the advertised range of a sensor is only achievable under extremely ideal conditions, which likely will not be the average use case, so we learned not to necessarily trust for the sensors to work as well as they're advertised to. As we were warned by the TAs and professors, we learned that sometimes sensors break or just spontaneously stop working, so ordering backups is crucial. One of our LiDAR sensors stopped working in the middle of the semester, but we had ordered a spare so we were able to continue working without delay. We also learned that integration can be painful and time consuming, so we were grateful that we had planned sufficient time for that process. Finally, we learned that in a process this long, it is hard to know exactly where things might go wrong, so we were glad to have left slack time so that certain items were not completed on time, it did not throw off our ability to complete the project.

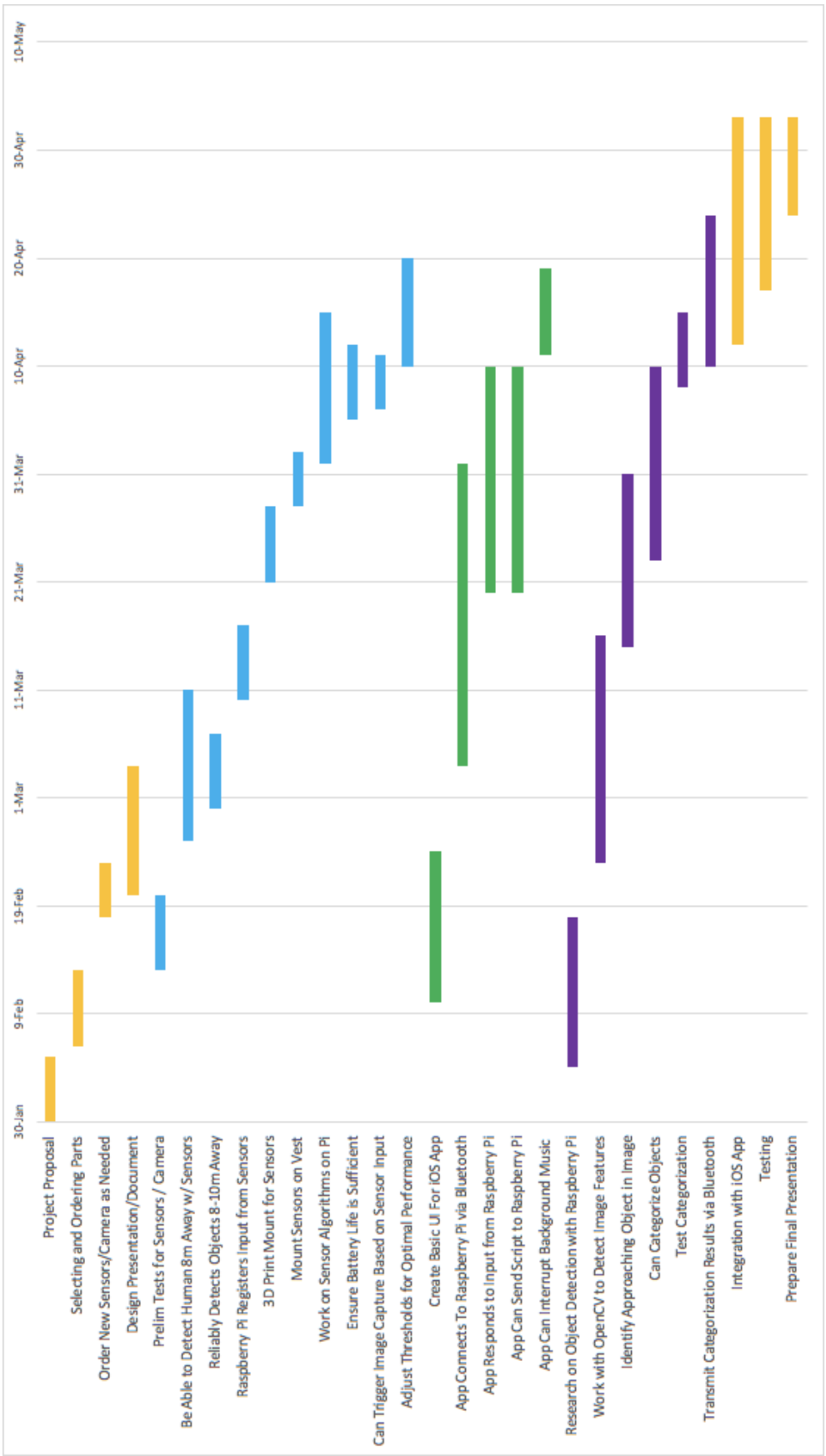

Figure 6: Final Schedule

Bill of Materials:

Budget Components

Part Type/Name	Quantity	Unit Price	Total Price	Source
URM37 Ultrasonic Sensor V5.0	1	$29.90	$29.90	Dfrobot
32Gb Micro SD Card	1	$12.03	$12.03	Amazon
TFMini - Micro LiDAR Module	5	$43.32	$216.60	Amazon
CamKix Chest Mount Harness	1	$14.99	$14.99	Amazon
4000mAh Lithium Polymer Battery	1	$25.95	$25.95	Amazon
Raspberry Pi Camera Module V2	1	$27.43	$27.43	Amazon
Neoprene Running Hydration Belt	1	$23.95	$23.95	Amazon
WitMotion USB-UART Converter (USB to TTL)	4	$18.59	$74.36	Amazon

Total Budget: $600; Total Spending: $425.21; Surplus: $174.79

Non-Budget Components

Part Type/Name	Quanity	Source
Rasbery Pi 3 Model B	1	Previous project surplus
Single Strand Wires	<15m	Provided by Lab
Wire Crimps	<30	Provided by Lab
USB to MicroUSB cable	1	Previous project surplus
3D Printer Filament (PLA)	<1 Spool	Previous project surplus
IPhone 7	1	Personal device
Lightning Headphones	1	Personal device

Tools

Tools Name	Source
Makerspace 3D printer	Makerspace
XCode Platform (Devoloper Account)	Previous Project Dev Acct
Python 3	Python.org
Autodesk Inventor Pro	CMU Cluster

Figure 7: Final Bill of Materials