Bikewards View

Final Report

Author: Albany Bloor, Emily Clayton, Jason Xu: Electrical & Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of enhancing bicyclist safety via the use of microwave sensors for blindspot detection. Microwave sensors are the up and coming technology for blindspot detection for cars, but we are unaware of any system available for bikes. Bikewards view will be retrofittable and also include an array of LEDs both for signaling to surrounding traffic and to display information to the user. Custom device drivers and signals processing software will be implemented to handle the information from the sensors.**

***Index Terms*—Design, Sensor, Microwave, Safety, Bike**

I. Introduction

BikewardsView is a retrofittable handle bar/tail mount system designed to provide increased safety to bicyclists everywhere. When on the road it can be difficult to split attention between the path ahead of you and the traffic behind. BikewardsView seeks to solve this problem by seamlessly delivering warning information to the user. The system employs a microwave sensor running at 24GHz on the tail light mount to provide blindspot detection capabilities in tandem with a grouping of lights to signal to surrounding traffic. The handle bar mount houses both LEDs to provide information from the blindspot sensors to the bicyclist as well as a toggle switch to enable the bicyclist to control the rear signaling LEDs. Compared to simple mirror based systems, BikewardsView allows the bicyclist to both send and receive information. This provides opportunity for both the bicyclist and surrounding traffic to course correct if necessary, lowering the chances of collision significantly.

We determined that the blindspot detector should be able to cover any area not easily viewed by the bicyclist with small head motions while keeping their shoulder stationary. By running some experiments on the members of our team, we determined that this meant we needed a field of view encompassing forty five degrees to either side from directly back from the bicyclists head. For the system to function properly to avoid collisions, both the bicyclist and other motor vehiclist must be given time to react appropriately to any information provided by Bikewards View. From our experiences in urban environments or other small roads where cars and bikes would be colocated. An average speed of a bicyclist is estimated to be about 20kph and that of a car to be about 50kph. This gives us a relative speed of 30kph. Some further testing on our team members was used to determine that the average time it takes a human to process data is around 250ms. Combining this with one full second to allow for any necessary reaction and our assumed relative speed, we determined that we needed to be able to sense objects at a minimum of 10.42 meters out. To ensure user trust in the system, we hope to maintain a no-fail policy for detection within this range while minimizing false positives to only ten percent of presented information. To ensure the system is usable we also want any rear signaling light to be visible and clear in all weather conditions at the specified range and the informational LEDs on the handlebar to be understandable in any lighting condition. Also to enhance user experience, we want to be able to provide an entire day's operation between charges, this to us, means one hour of active use to account for commuting time and eight hours on standby while the user goes about their day.

II. Design Requirements

The above user requirements further inform several requirements for chosen components. To be able to continuously provide the user new data as soon as they are able to process it, we want to keep our latency to under 250ms. The ranging and varying lighting conditions also meant we wanted a sensor that was not light sensitive. Further to ensure correct action was taken, it was determined that the sensor needed at a minimum to be able to distinguish between objects behind and to the left or right of the bicyclist, meaning either multiple sensors or one with angular data would need to be used.

Metric	Requirement	Results
Minimum Range	10.42 Meters	10.05m
Minimum FOV	90 degrees	~100 degrees
False Positive Rate	10 percent	37 percent
Latency	250ms	330ms
Accuracy	100 percent	72 percent

III. Architecture and/or Principle of Operation

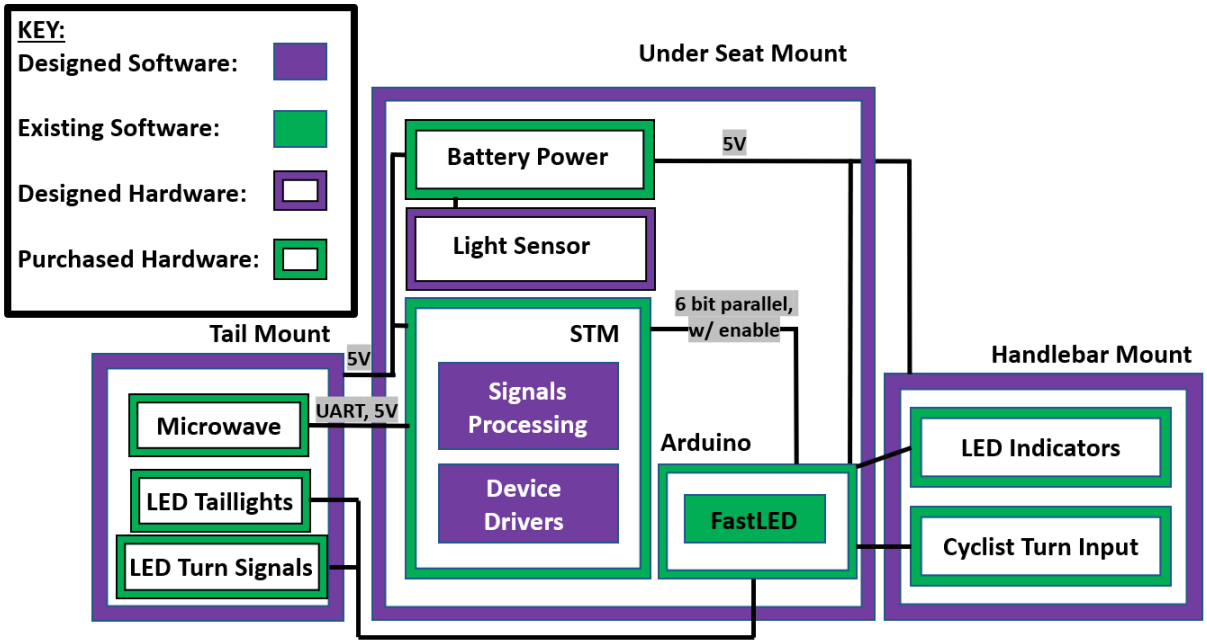


Fig. 1.
System Diagram

The main principle of our design is to feed information about objects behind a biker and outside of their comfortable field of view to them via a handlebar mounted LED array. Two microwave sensors are the source of the data and they are used to determine the general location of the object relative to the bike: back left, center, or right. Data from these sensors is processed through a STM32F401RE processor for signals processing and a decision tree then determines whether or not our Arduino will turn on the LEDs to display information to the driver. The driver will also have the ability to control their tail lights via a switch on the handle bar to indicate turns. All of the lights decrease in brightness during the night so as to not hurt the eyes of drivers or cyclists. At night, the tail light also turns on to provide extra safety for the cyclist. The physical mount is custom made, as well as the sensor processing code for the data from the sensors.

IV. Design Trade Studies

The two most major components we were initially concerned with the selection of were the sensor for blind spot detection and the processor. The sensor likely went through the most design changes of any part as is laid out below.

A. Design Specification of the Main Sensor

The selection of the sensor for blindspot detection was identified as the most critical design decision in our build. Our initial thought was to implement this system using an array of low cost ultrasonic sensors, but most models we could find had a limited maximum range of 10m which did not satisfy our calculated requirements. The MB1260 XL MaxSonar EZL0, one of the options we were looking at, was able to achieve a 10.5 m range, but at a great cost to FOV [2].

Following a suggestion from one of the team members, we then looked into the possibility of using a rotating LIDAR. This initially seemed promising. Most such sensors we found had high read rates and a seemingly large range. This was also when we introduced a requirement for minimum angular resolution for near single point devices. As the rotating LIDARs did not have a significant FOV for single measurements, they're ability to cover our FOV adequately was dependent on the number of points that would be reported for any given angular segment. This requirement was technology driven, however, and abandoned along with the rotating LIDAR. At the time of the proposal presentation, we had settled on the Slamtec A1M8, which had a 12m range [3] and had identified that maintaining this range in daylight conditions would be a major challenge. Further research, however, brought to our attention that even in perfect ambient lighting conditions, the paint color of the cars we were tracking would greatly affect our range. In fact, for black painted cars. we found that reflectivity could fall below five percent [4]. This severely limited the range of LIDAR equipment we were using. We thus, initially tried to overcome this issue by using longer range LIDARs that would still provide some range with low reflectivity, but at our predicted minimum reflectivity, no readily available sensor could meet this requirement.

Jason, thereupon, recommended we look into the usage of microwave sensors which seemed to us to be a more up and coming technology. This brought us quickly to the SEN0306, 24GHz microwave sensor from DFRobot. This sensor had a 3db beamwidth of 78 degrees horizontally and

23 degrees vertically along with a reporting rate of 10Hz and reported range of 20m [1]. Testing of these sensors (discussed below) would show the true range to only be between 10m and 11m with a reporting rate closer to 3Hz. We did, however, due to timing considerations given the point in the semester at which we had procured and tested these sensors, chose to continue to use them for blindspot detection. The beamwidth of the sensors did seem to meet specification, and so it was determined that we could use two of them to meet our 90 degree FOV requirement. Further with two sensors, we were able to implement some degree of direction finding for objects (back, backleft, backright) which for the limited amount of information we want to feed back to the biker is sufficient.

B. Processor Selection

There are fortunately many microprocessors that can meet our requirements, with a vast variety of memory bank sizes and I/O port selection. We decided to pick the STM32F413 model since it has ample processing power and enough I/O for both the sensors and the LEDs, and a relatively low power draw that will not affect our power system significantly. Another major factor in our selection is that this model is the microcontroller of choice for 18-349 Embedded Systems, which two members of the team are taking or have taken. This makes it easy to work with for us.

We chose to get this microcontroller on a development board, the Nucleo F401RE, to save the time of PCB development for the microcontroller, and also to get a tried and tested component to reduce our risk factor in development.

C. LED, Battery, and Other Component Selection

We looked into multiple methods of displaying information to the cyclist. We chose light indication over audio considering that the audio would need to be repeated at a high volume to ensure that the cyclist would hear and understand the data. We also determined that we could convey more information over a shorter period of time using lights of different colors in different locations. Thus, we chose to use LEDs.

At first, we thought that having single point LEDs would be sufficient in displaying information to the cyclist. However, when thinking about practical use at night and during the day, LED strips would be less likely to blind the cyclist while being equally as noticable considering that they can output over a larger area. We decided to go with addressable RGB lighting over single color because this will allow us more flexibility in outputting to the cyclist.

We originally planned to control the LEDs from the STM32 processor directly, however we decided that a much simpler solution would be to program an Arduino to listen to processor commands and communicate with LEDs using their Arduino drivers. For this configuration, the LEDs are attached to the Arduino through their serial ports, and a 6 pin GPIO bus is used by the processor to send codes indicating LED modes to the Arduino.

For the power system, we calculated that the system would consume at most 4A with all lights on and the system running. Additionally, the Arduino and STM32 are going to be run off of a 5V input. The microwave sensor requires a 4-8V input, which the STM32 will supply through its 5V output. From our design requirements, we require that the system run for 1 hour actively. Thus, we needed at least a 20Wh battery. We decided to go with two of the Blomiky NiMH batteries. They are rechargeable and output at most 2.2A, meeting our power requirements. The batteries have most capacity that we will actually require, considering that it is unlikely that all of the lights of the system will be fully on for an hour, but this will allow a more reasonable use time for the user, as well as being convenient for testing.

V. System Description

A. Hardware Systems and Control of LEDs

An Arduino serves as the central controller for all LEDs. On this device, we define LED modes and the STM32 sends an interrupt to the Arduino, the 6 pin GPIO bus originating from the processor determines which mode has been requested. We then use the library provided to us by the LED vendors to set all lights accordingly (FastLED).

The current LEDs we plan on using for the handlebar display are the BTF-LIGHTING WS2812B RGB 5050SMD. This is an individually addressable light strip that will allow us to segment the display into different warning systems and also use one of the end LEDs to indicate system status (on or off) to the rider [5]. A tripole switch will be used to allow the rider to indicate and a toggle switch will allow them to turn the system off when not in use and to remove the battery for charging. The battery is a Blomiky 5V, 2200mAh which should last comfortably for an entire day of use if the system is switched off when not in actively being used [6].

B. Signals Processing

Our signals processing code is hosted on the STM32F4 in C language to be compatible with Jason's driver designs. The signals processing code encompasses both taking raw sensor data and boiling it down to a more understandable format, and also encompases the decision matrix for what to show the biker. The processing code first determines object distances and speed for each sensor, combines these two values into a "danger" value presented in hertz which may be read as one over the estimated time until impact with the bike if the same relative speed was kept from the current distance. These danger values are then reviewed to determine if any are sufficiently similar to correspond to an object that should be shown in the center portion of the display.

The 126 point, linearly mapped distance array from the

sensor is first fed into a function that determines the five closest target distances based on spikes in this array. It was noted in reviewing early testing data that all spikes we saw in the array data from the sensor were single index spikes, rather than the bulbous, plateauing spikes we expected from the manufacturer's documentation. The distance calculating code for our project is largely based off of the sample code for multi-target detection provided by the manufacturer in their documentation [1]. According to the manufacturer, the magnitude of these array spikes could range from 1 to 44 (unitless) [1], which did match our observations. We further observed that closer objects generally generated larger magnitude spikes, and that on occasion further objects caused short spikes early in the data while demonstrating a larger spike at the correctly mapped index. To stop this from creating false positives we ignore any spikes in the input array data that have an index value less than 45 (corresponding to about 4m) and a spike magnitude of less than 5. The distance determination code fills an array of length five with integer values of the indexes where spikes were seen. If less than five objects were detected, later array values will be negative one.

The code then passes this new distance array, along with the previous distance array, into a function to determine danger values, still at the single sensor level. The code first looks at the first distance value of the new distances array and compares it with the first value of the old array to determine if the values show distances so far apart that it would be unlikely for a vehicle to have moved through the linearly mapped distance in between samples. This being the case, the first value of the new array is assumed to be an object the bicyclist is passing, and the second is paired with the first value of the old array for determining speeds. Similarly the last two values are looked at to determine if an object would have fallen off the back of the old array, thus not being seen in the new array. The distance values for the old and new array thus being matched, the old array index value is subtracted from the new array value to determine at most five object velocities paired with the five distance values from the new distances array.

These velocity/distance pairs are then examined and danger values returned in the following manner. If the new distance value corresponds to a sensor data array index of less than 50 (~4.5m), a danger value of 40 (comfortably into the red zone for the LEDs is assigned), otherwise should either of the values be negative, a danger value of -1 (LEDs off) is assigned. Finally for dually positive velocity/distance pairs the danger value is determined to be the velocity times three (to account for the sampling rate) over the distance.

These danger values are returned in an array of length five from each sensor. These arrays are sorted to ensure the highest danger values are earliest in the arrays. The two arrays from each sensor are compared and the earliest sufficiently similar pair of danger values from the two arrays is taken to be the danger value for the center segment of the LEDs. If no sufficiently similar pair can be found the danger value for the center segment is set to -1. The danger values for the left and right segments of the LED display strip are the highest values in the array that were not sorted to the center sections. A danger value greater than 2 corresponds to red on the LED display, and a danger value greater than -0.25 corresponds to yellow on the LED display. Any value lower than this will turn off the LEDs. The chosen LED colors are the final returned product of the code. They are returned via a six bit value, the upper two bits corresponding to the left segment, the middle two bits to the center segment, and the lower two bits to the right segment of the LEDs. To turn the LEDs off these bits are set to 00, to turn the LEDs to yellow they are set to 01, and to turn the LEDs to red they are set to 11.

The signals processing code was custom made for our project and only relies on the following basic C language libraries. <stdlib.h> was used for basic functions such as absolute value of integers. <stdio.h> was used primarily for print statements during debugging. <math.h> was used to find the absolute value of floating point numbers. The three libraries are all included in the C standard library.

C. *Embedded Software Architecture*

The embedded software on the STM32 microprocessor ties together the various subsystems in our design by reading inputs from the pair of microwave sensors, hosting the signal processing algorithms, and outputting commands to the LED driver. It is also able to show the latency within the signal processing loop and the subroutines in it through debugging tools.

The software consists of a single loop that starts upon available sensor data on UART. It then updates the danger heuristic according to the new and previous data stored on the processor, and sets a new LED code reflecting the update. The software then generates a GPIO interrupt, and then updates every GPIO port with the corresponding bit value.

Debugging output is done by coalescing raw sensor data, updated LED code and sensor identifiers into a buffer, which is sent through the USB debugger. This data is then de-compressed and displayed by a Python program using PySerial to read USB input.

VI. Test and Validation

We have determined that in addition to several quantitative tests to assure that everything is up to spec, some human testing will be required to ensure that the interface works well and is understandable in all feasible scenarios.

A. *Sensor Testing & Analysis*

Using the STM32 or Arduino, we can read the serial output of the microwave sensors. We used this to store and analyze the exact data we were getting back from the sensors. The sensors, along with headers and footers, return a two byte distance value in centimeters along with an array

of length 126 that is linearly mapped to distance that displays multiple detected objects as spikes in the array values [1]. We first performed testing on one sensor at a time, getting roughly the same results from each. We started by moving backwards from a vehicle to see when the spike in array value fell off the end of the array. In doing so we noted the average range of the sensors to be about 10m directly back from the center. It was also noted in performing these simple initial tests that an accurate two byte distance value was returned for another meter after information disappeared from the linearly mapped array. It was also important to note that with our look angle and height off of the ground, we were not getting any spikes caused by the road surface.

We then moved on to record the array spike location and distance values for the sensors at 1m intervals from 1m to 9m. The results for which can be seen in figure 2.

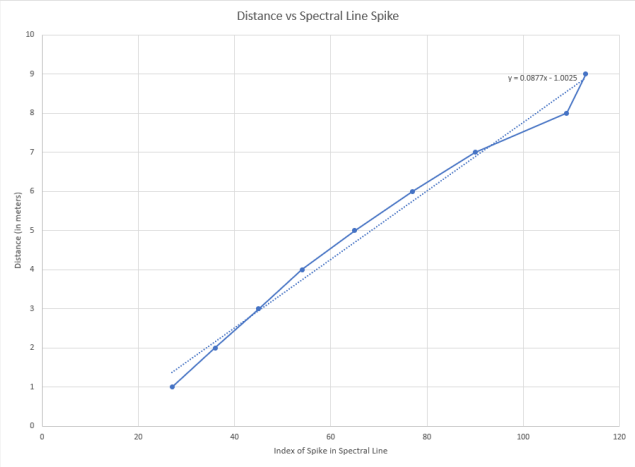


Figure 2: Graph of target distance versus returned spectral line spike and fitted line

The data we got from this experiment was close to being linear as expected, and we fitted a linear approximation to this data to have a better understanding of the sensor's range. Using this fitted line, at the maximum array index of 125 we get an approximated range of 9.96m. It was also interesting to note that the y-intercept of the fitted line was non-zero. We attribute this y-intercept of -1 in part to sensor error and in part to us measuring distance from the license plate, the furthest forward point on the vehicle. Our testing setup for single sensor testing may be seen in the below image 1.

Image 1: Single sensor testing

A similar setup was used for initial dual sensor testing and for system testing. Any testing of the blindspot system was done in a controlled environment in a turnaround in a park. Initial dual sensor testing was performed to ensure that no issues arose having both sensors running simultaneously and to generate some data to use when refining the code prior to integration.

B. System Accuracy and Range Testing

Both sensors were also tested simultaneously during full system testing. First the sensors were approached from a distance until a signal was shown on the handlebar display from the center and from thirty degrees to either side of the centerline of the mount. It was found that at thirty degrees an object was first reported between 9.3 and 9.6 meters. At the center, however, we did not see the object reported on the handle bar display until we were only 6.5 meters out. We believe this may be due to the lobes of the sensors' field of view being centered six degrees to either side of the center creating a small pinchoff, though the drop-off was larger than expected and we would have expected a larger drop-off to the sides than at the center. We also tested the field of view of the system by walking in arc two meters in radius away from a point between the two sensors and noted that objects were first reported on the handlebar about 50 degrees to either side of the centerline, exceeding our requirements.

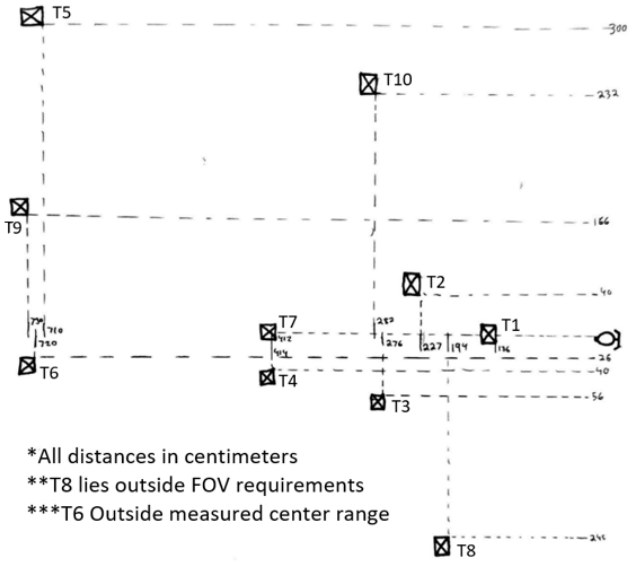


Figure 3: Accuracy testing target locations. Distances given in centimeters

Target	Correct Display	Accuracy	False Positives
T1	Red Center	100%	10%
T2	Red Center or Left	100%	20%
T3	Red Center or Right	100%	0%
T4	Red/Yellow Center	80%	70%
T5	Yellow Left	70%	30%
T6	Yellow Center	0%	60%
T7	Red/Yellow Center	50%	100%
T8	Red/Yellow Right	100%	0%
T9	Yellow Center or Left	30%	40%
T10	Red/Yellow Left	90%	40%
Average		72%	37%

Table 1: Accuracy and false positive measurements for ten sample locations

We then moved on to test accuracy and false positive rates. In order to test accuracy ten pseudo-random target locations were selected and are mapped out in figure 3. At each of these target locations, ten snapshots of the display were recorded at times selected by a team member who could not see the display to minimize the bias of our measurements. Accuracy and false positive rates are recorded in table 1. The display was determined to be accurate for a particular sample if it displayed the correct display value as described in table 1 when sampled. The lowest accuracy values were seen around the center of the field of view at distances greater than 7m out. Any objects with an adjacent offset of less than 2.8 meters from the sensor mount displayed 100% accuracy, and a roll-off occurred as the targets moved further away from the sensor. The sensor was marked as displaying a false positive any time that a segment of the display that should not have been lit was on or when red was displayed for a target distance that should have been yellow. T5 for example only ever displayed information on the left segment of LEDs, however, three of the ten samples for this target showed red displays rather than yellow and were thus marked as inaccurate false positives. Averaging the accuracy of the display across these ten targets we found the system to have an accuracy of 72% and a false positive rate of 37%. Similar errors to those that occurred at target T5 accounted for a fair portion of the discrepancy from our requirements. The inclusion of T6 in our data as it fell within required range, but not measured range, also brought down our accuracy.Initially we had plans to perform real world testing of the system, but these were abandoned out of safety concerns.

C. Hardware & LED Testing

Safety is the primary purpose of the system, and the outputs need to be quickly and easily understood, so we decided to use red and yellow as our distance indicators. After getting feedback from people within the class and 8 people outside the class, as well as our own experiences riding the bike, we decided that these colors were suitable.

For the power system, we monitored the battery output every half hour to ensure that they maintained their voltage. After about 1.5 hours, the batteries began to degrade to the point where their voltage could no longer power the Arduino and the LEDs would not respond due to the lack of data input. Also, after measuring and averaging the current draw from the batteries over that time, the average current draw was around 4A, supporting that our system can run for about 1.5 hours while on.

After using the LEDs while testing our system, we discovered that they were extremely weak at the solder points, to the point that they would separate from the circuitry spontaneously within 3-5 days of soldering them. In order to combat this, we used hot glue to add some strain relief at the weak points. The clear hot glue strain relief allowed the lights to shine through, and the LEDs have not broken for 2 weeks as of now.

D. Signals Code Testing

Prior to integration, the signal processing code was tested both for correctness and timing. The code was tested by manually replacing the input sensor arrays to the code with data from initial sensor testing. In terms of timing, including print statements for debugging, the signals code ran in under a millisecond and was deemed to have negligible effect on the timing of the system as a whole. The code returned expected values for each dataset fed into it. It should be noted that the testing data was from a relatively clean data set and some spikes were added in later to testing to simulate objects falling off the back or entering from the front. Some refinements were performed in the short time between integration and our final demonstration to account for longer time between sensor data sets and to implement changes in system approach as discussed in the

architecture portion of this report.

C. LED Control Testing

Before integrating the system together, we tested the LED outputs by inputting static numbers, mimicking what we would receive from the STM32, and checking the output on the LEDs against the expected output. The outputs were corrected, but because after integrating the system, we ran into timing problems with the interrupts from the STM32. In order to update the handlebar LEDs as quickly as possible, we handled these signals first, with the turn signals and the brightness inputs from the light sensor second. The light sensor was also tuned by recording the input on the Arduino during dusk when cars would turn on their lights and setting this value to the threshold for turning on the taillight and decreasing the brightness of the handlebars.

D. Software Testing

Testing of the software's correctness is done by running the sensor tests in section A with the microwave sensors connected to the STM32. This enabled us to quickly find bugs in the code to solve. Latency testing was done using global variables embedded in the code to store elapsed time between each loop iteration and between every major subroutine, such as polling a sensor or running the signal processing code.

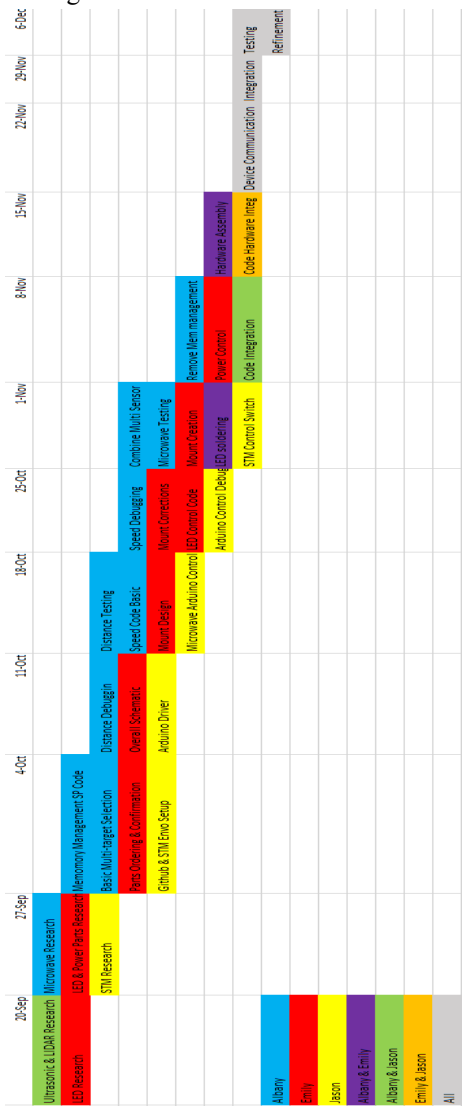
Through latency testing, we realized that the latency of our code without the communication portion was less than 1ms, which means that communication latency contributes to almost all of the delay within a loop. This enabled us to simplify the code by running just a single loop, instead of our proposed RTOS and interrupt based code.

We also found that the sensor reading takes up to 200ms to complete, with both sensors taking up to 300ms. This is unfortunately a problem that makes the true latency of the code marginally slower than the 250ms we wanted, and not within our capabilities to fix. To mitigate this, we removed some granularity of the closeness indicator LEDs, so that the user's info can still be up to date.

VII. Project Management

A. Schedule

Our schedule saw several setbacks throughout the semester that were mainly compensated for by built in slack time. Extended time spent on sensor selection meant that portions of the project directly interfacing with the sensors and component testing couldn't be performed until the early November timeframe. Since sensor testing was performed so late, we lacked the practical ability to look into other options after discovering the issues with our components. This delay further pushed the start of our system integration back a week. Though integration of the signals processing code onto the STM was fairly easy given constant communication about the desired interface, other portions of the integration, particularly the communication between the Arduino and STM required significant reworks. All-in-all, a fully functional version of the product was not ready until the Friday before the final demonstration, leaving little time for refinement.



B. Team Member Responsibilities

Albany Bloor is responsible for the implementation and design of the signals processing code. She was also in charge of sensor testing analysis and accuracy and range analysis for the system as a whole. She further performed some component soldering.

Jason Xu is responsible for setting up the software architecture, enabling interfacing with sensors and LEDs, and adapting the signals processing code for use on the embedded processor. He is also responsible for providing Arduino sketches and Python programs that are used by the team for testing, and managing the version control and GitHub repository.

Emily Clayton is in charge of all the selection and wiring

of all hardware components expecting the sensor as well as the physical assembly. She is in charge of power and resource management and will also be the principal team member for testing of the sensor and other hardware components prior to the integration of custom code. She is in charge of driving the LEDs from the Arduino.

C. *Budget*

Please see the spreadsheet on page eight. All parts ordered to date amount to about $440.

D. *Risk Management*

During development, we had some risks that we ultimately managed to resolve. The most major problem we faced concerned the sensors, which (as we mentioned in the previous sections) did not meet the advertised specs on either range or latency. Thankfully, the true range and latency were workable for us, considering that our processing speed was much faster than expected and the true range of the sensor was within our minimum desired range.

Another issue we faced was that serial communication between STM32 and Arduino did not work, despite us trying multiple methods of serial communication, including I2C and SPI, and using different implementations, including using the given Hardware Abstraction Library, and setting the necessary registers and interrupt handlers normally. Since this step is crucial, we decided to design a GPIO-based communication method with one wire for each bit, with 6 wires in total for the 6-bit LED code, and one wire to trigger an Arduino interrupt. This method, although not as clean wiring-wise, proved incredibly reliable and simple.

VIII. Ethical Issues

In order to avoid major ethical issues it is important for us to not overstate the accuracy of our system. Creating a false reliance on an imperfect system could result in catastrophic consequences down the road. Though our system does provide an extra layer of security for any biker, even its most basic functions must be supplemented with best practices by the biker. Though we have an indication on the handlebar as to whether or not the blindspot detection portion of our system is on and working, no such failsafe exists for our taillight or indicators. Furthermore, the ability to signal to traffic with these indicators does not excuse a biker from ensuring they have adequate space and time to perform a turn and remain visible to the surrounding traffic. Similarly, though the taillight helps keep a biker visible at night and in poor conditions, it would present a problem should the user come to rely solely on it to the abandonment of reflectors and bright clothing.

Our greatest ethical concern, of course, comes not with an overreliance on these subsystems, but with an overreliance on our blindspot detection system. Though the system has great enough accuracy to bolster the confidence of the biker and give a slight increase in security, it is by no means infallible. The biker must continue to remain vigilant and aware at all times. Should the biker become reliant on our system, and should the system fail to warn them of an impending collision, it could result in injury and even possible loss of life.

IX. Related Work

18-349 Semester Project - RTOS with scheduler running priority inheritance protocol, supporting up to 16 threads and 32 mutexes, and serial communication through UART and I2C

X. Summary

Using 2 microwave sensors, Bikewards View notifies the cyclist of objects approaching behind them. It indicates the general direction and distance of objects on handlebar mounted LEDs, decreasing the amount that they have to look behind themselves. The cyclist can utilize turn signals to decrease the amount that they have to take their hands off of the handlebars. The system operates off of 4 rechargeable batteries.

This system isn't meant to entirely replace looking behind oneself when biking, especially considering we only achieved 72% accuracy. Rather, the system is meant to make the cyclist feel more aware of what is around them, even when they aren't looking behind them. While testing and using Bikewards View, we found that it was a good supplement to have while riding.

Glossary of Acronyms.

LED - Light Emitting Diode
I/O - Input and Output
RTOS - Real-time Operating System
I2C - Inter-Integrated Circuit
(serial communication protocol)
UART - Universal Asynchronous Receiver-Transmitter
(serial communication protocol)
LIDAR - Light Detection and Ranging
GPIO - General Purpose Input/Output

References

[1] DFROBOT.,https://wiki.dfrobot.com/24GHz_Microwave_Radar_Sensor_SKU:%20SEN0306#target_0
[2] MaxBotix, https://www.maxbotix.com/ultrasonic_sensors/mb1260.html
[3] Slamtec, https://www.slamtec.com/en/Lidar/A1
[4] Ford Motor Company, https://detroitcc.org/wp-content/uploads/2018/07/IR-Reflectivity-of-Paint-Autonomy-and-CO2-Seubert.pdf
[5] Amazon.com, https://www.amazon.com/BTF-LIGHTING-Flexible-Individually-Addressable-Non-waterproof/dp/B01CDTE6Y6?th=1
[6] Amazon.com, https://www.amazon.com/Blomiky-Battery-Projects-Equipments-Portable/dp/B08J4H39JV

BUDGET:

	Sensing	Manufacturer	Model Number	Price	Number	Total
2	24GHz Microwave sensor	DFRobot	SEN0306	79.9	2	159.8
3						
4	**Processing**					
5	Eval board (with STM32 F401RE Processor)	STMicroelectronics	NUCLEO-F401RE	27.99	1	27.99
6	Arduino Uno (LED driver)	Arduino	A000066	0	1	0
7						
8	**Power**					
9	5V 2.2Ah Batteries (Pack of 2)	Blomiky	B07WQH5VNS	24.99	2	49.98
12						
13	**UI**					
14	Addressable RGB LEDs	BTF-LIGHTING Technology	WS2811SM60LB65	22.88	1	22.88
15	ON-OFF-ON Turn Switch	TWTADE	KCD4-223-JT	11.99	1	11.99
16	ON/OFF Power Switch	Nilight	90012E	8.9	1	8.9
17						
18	CAD Mount			43.5	1	43.5
19	CAD Mount Draft			43.5	1	43.5
20	Solder Boards			0	1	0
21						
22	**System**					433.51