Ride-AR

Ethan Wang: Author, Chad Taylor: Author, and Fayyaz Zaidi: Author

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of altering a city cyclist when there is a potential danger approaching from their rear and blind sides. This, as of, now includes both oncoming vehicles and pedestrians. We did this through the use of a Lidar in tandem with a camera implementing an object recognition algorithm. With the two together, we would continuously run scannings about the space around the biker and hope to allow ample time for the customer to react accordingly..**

***Index Terms*—**Kinematics, Lidar, Object Detection

I. Introduction

In the US in 2019, there were over 40,000 bicycle accidents with motor vehicles resulting in over 800 deaths. A large number of bicycle accidents occur because they fail to yield or see a bicyclist and end up colliding with them. While there are many safety systems bicyclists use to improve their visibility such as lights and reflectors, vehicles still cause many accidents with bicyclists.

The goal of our system is to use a LIDAR and a camera to detect cars approaching cyclists outside their field of vision to improve their situational awareness. The system will use a visual warning to alert bicyclists of a dangerous situation to give them more time to avoid an accident.

First, we will use an object detection algorithm to identify cars behind a biker in the image. Then, we will use the LIDAR readings to determine the distances to the detected cars. This is so we can selectively alert on approaching vehicles and we do not accidentally detect inanimate objects such as trees and mailboxes on the side of the road.

The most common biker safety systems are passive systems such as lights and reflectors. While these systems make bikers more visible to cars, they do not help bikers avoid oncoming cars. Our system seeks to address these issues by providing an active warning system.

There exist some commercial products similar to ours which use radar to warn bikers of oncoming vehicles. However, our use of a LIDAR and a camera will give our system a much more accurate and granular level of detection and we will be able to visually display information such as the speed and direction of cars that are approaching. Additionally, we will be able to use object detection to selectively alert on cars approaching whereas radar sensors could be activated by inanimate objects on the side of the road.

II. Use-Case Requirements

For our system to work effectively in detecting and warning bikers from cars, there are a few requirements the system must meet. Our system is mainly designed for bikers to use when biking in more residential areas or urban environments when the speed limit is 30mph or below.

A car traveling at 30mph travels 44ft per second so it is imperative that our system can detect and alert bikers quickly of oncoming vehicles.

Our system needs to be able to detect and locate cars within a 120 degree field of view behind the biker to give the biker effective situational awareness. We also want to be able to detect cars up to 60m away, which would give a cyclist over 4.5 seconds to react to a car approaching at 30mph. For this we would need a latency of <0.3s.

Additionally, we want to be able to correctly identify the distance to the car within 1m. This is so we can accurately display information about where a car is located relative to the biker. Using this distance information, we can calculate the speed of the car relative to the biker, and selectively warn the biker based on speed.

For our object detection algorithm, we want to be able to detect a car in the camera's field of view with a <10% error rate. This error rate implies the rate of false negatives which means not classifying a car/pedestrian that is present in the image. While we recognize that achieving 100% accuracy in detecting cars is not always feasible, we want our system to be as reliable as possible as it concerns human safety.

Our system needs to be portable and be able to be powered without an external power source. As a result, our power system needs to be able to power the system for at least 6 hours to last a day of biking.

Due to the physical profile of a bicycle, we want our entire system to weigh less than 3 pounds and be able to be contained in a module less than 0.5ft^3 so that it has a small footprint on the bicycle. We don't want the system to make biking awkward or uncomfortable to the user.

III. Architecture and/or Principle of Operation

At the rear of the bicycle, we will mount the Slamtech A1M8 LIDAR and the Microsoft LifeCam Studio camera. This camera differed from the original Sainsmart camera as we experienced issues with the CSI connector of the latter. We pivoted and tried a new camera with a USB connection and seemed to have better luck. The Slamtech LIDAR can detect objects up to 12m away, sweeps at 5.5hz, and has a 360 degree field of view. The Microsoft Lifecam Studio camera has a 75 degree field of view and supports 1080p @ 30fps and 720p @ 60fps video recording. The camera will send the coordinates of bounding boxes to a file that is read from a ROS node. This ROS node will be published to a ROS topic that is also interfacing with the LIDAR that collects the range information. The integration of the ROS nodes interacting with each other is also different from the original design proposal. Using the data from the object detection algorithm and the LIDAR, we developed a separate algorithm which determines the location of detected cars and displays that

information to the user. The information will be displayed on a wearable display.

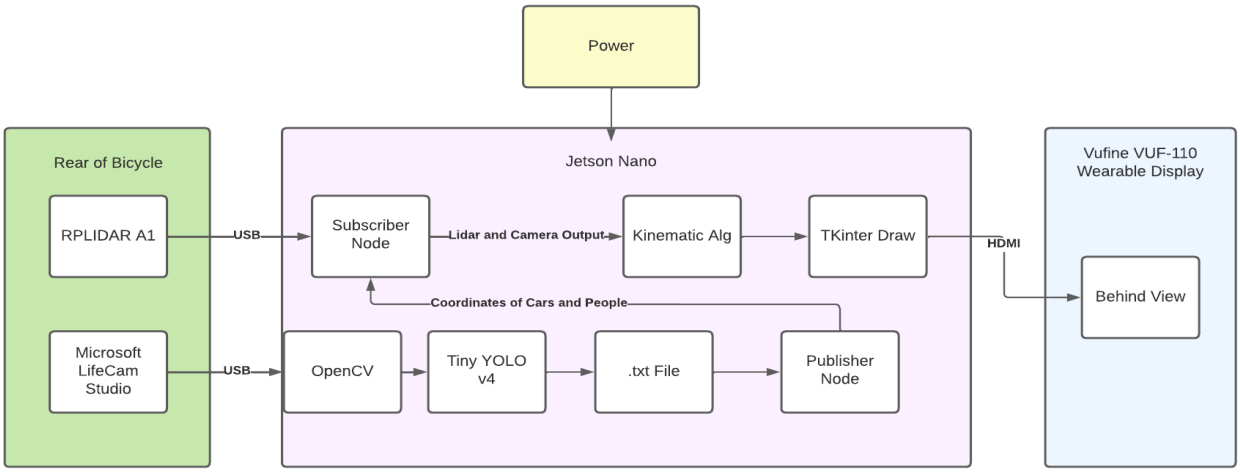


Figure (1). There are three main components in the design. The sensor module which will be mounted at the rear of the bicycle, the computing and power module which takes input from the sensor module, and the display module which the biker will wear which displays visual information regarding oncoming cars.

(Figure 2) SLAMTECH Lidar

(Figure 3) Nvidia Jetson Nano

(Figure 4) Microsoft Lifecam Camera

(Figure 5) Vufine Wearable Display

(Figure 6) Battery Pack used for system

IV. Design Requirements

Based on our use-case requirement to detect and range cars within a 120 degree field of view behind the biker, our camera needs to have at least a 120 degree field of view and our LIDAR sensor also needs to have at least a 120 degree field of view.

We want to detect cars up to 60m (200ft) away to give bikers 4.5 seconds to react to cars approaching at 30mph.

In our case, the average LIDAR with a sufficient field of view within our budget has around 12m of range, or about 40 feet. For the sake of our project we will be implementing and testing our system to detect cars using LIDAR within 12m as a proof of concept, knowing that in an actual application we can easily extend the system to detect cars at a farther distance using a stronger LIDAR system which can detect up to 60m.

While our specific LIDAR is only limited to a 12m range, we want our camera to be able to view cars clearly at 60m away to give our object detection algorithm the best chance to detect the car. The specific camera we chose is able to shoot video at 1080p which is sufficient.

In addition to our sensors having sufficient range and field of view, our sensors also need to sample at a sufficient rate. Cars can move extremely quickly relative to bikers, so it is imperative that our sensors have a fast sampling rate. Therefore, we want our camera to capture frames at 30fps. Taking frames at 30fps would allow us to see a car in the image in 0.03s. This level of granularity is overkill for our application, but most cameras by default capture in 30fps or 60fps, so this requirement is sufficient.

Our specific LIDAR is limited to 5.5hz (5.5 sweeps per second) which is sufficient for our testing purposes but in a practical application we would need a LIDAR with a faster spin rate. The LIDAR does not need to capture at 30hz, but a spin rate of closer to 10hz would allow us to detect changes in distance every 0.1s, which equates to about 4.4 ft for a car traveling 30mph and would be sufficient for our use case.

We want our warning system to be able to trigger within 0.3 seconds. This includes the object detection algorithm finding the bounding boxes of cars and people, sending to the ROS publisher node, the ROS node publishing to the topic, the LIDAR node reading from the topic and using the coordinates with the LIDAR measurements to draw rear view. The bulk of the computation time is the object detection algorithm. We are experimenting with different algorithms, but one algorithm we are looking at is the lightweight YOLO v3 algorithm. We want to optimize our system and algorithm for quick throughput and little to no latency that will allow for cyclist

V. Design Trade Studies

A. Design Specification for object detection algorithm

The YOLO object detection algorithm is orders of magnitude faster (45 frames per second) than other object detection algorithms. This is because of how the YOLO algorithm functions compared to other algorithms. The limitation of the YOLO algorithm is that it struggles with small objects within the image, for example it might have difficulties in detecting a flock of birds. This is due to the spatial constraints of the algorithm. This may have caused some issues if a car is far behind but speeding towards the cyclist, we would prefer to alert the cyclist as early as possible in order to give the cyclist enough time to avoid a potentially dangerous situation. This is why we are also planning to experiment with the SSD algorithm as it detects smaller objects with more accuracy than the YOLO algorithm.

The YOLO algorithm has multiple different versions for different usage. Looking online, people who have implemented the YOLO algorithm on the Jetson Nano seem to struggle with frame rate. For example, with YOLO v3 implemented, many users were able to get 1-2 fps. However, there is another version made specifically for this reason to run more naturally for devices such as the Jetson Nano. This version is called tiny-YOLO. Many people who implemented this version were able to achieve speeds of 10-15 fps. There is also an implementation already done for the Jetson Nano called JetsonYolo which is an implementation of Yolov5. We also implemented these versions ourselves for experimentation.

Performance on the COCO Dataset

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46		link
SSD500	COCO trainval	test-dev	46.5	-	19		link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244	cfg	weights
SSD321	COCO trainval	test-dev	45.4	-	16		link
DSSD321	COCO trainval	test-dev	46.1	-	12		link
R-FCN	COCO trainval	test-dev	51.9	-	12		link
SSD513	COCO trainval	test-dev	50.4	-	8		link
DSSD513	COCO trainval	test-dev	53.3	-	6		link
FPN FRCN	COCO trainval	test-dev	59.1	-	6		link
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14		link
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11		link
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5		link
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45	cfg	weights
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35	cfg	weights
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20	cfg	weights
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220	cfg	weights
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20	cfg	weights

(Figure 7) Accuracy and detection speeds of different Real-time detection algorithms

The image above contains detection and accuracy speeds of different algorithms and versions on the famous COCO dataset. As you can see Tiny-YOLO is much faster than the other larger versions but is less accurate because of this. s.

We also tried the YOLO algorithm along with the TensorRT package instead of darknet as this was recommended by many employees at Nvidia which apparently can get you even more fps on the Jetson Nano. We tested with both packages and see which would be better for our use case in terms of accuracy and speed. The main limitation for this design specification is the processing power of the Jetson Nano to compute all the detections in real-time.

After the implementation and testing of the aforementioned cases, specifically the use of the YOLO v3.v4, v5, algorithms with the Darknet or TensorRT packages, we found that TensorRT packages were not working and that v3 was not as accurate as we wanted to the system to be. Not only this, the v5 algorithm was rather large and took up too much computing time. From this, we decided to choose the YOLO v4 algorithm with the Darknet packages as it allowed for good accuracy, computing time, and compatibility.

B. Design Specification for Object Distance Mapping

Through our design process, we were trying to decide on the best hardware pieces that could map out the surroundings of the blind spot of the driver and provide accurate and precise readings.. Through this discussion, we discussed the use of Radar, Ultrasonic, and Lidar.

An ultrasonic sensor provides a cheap way of mapping out the surroundings. It does not require a contact to travel to the surface interface and back. They can be transmitted through other mediums such as air. This, though, does not outweigh the massive limitations that it presents. Ultrasonic needs constant temperature as well as lack accuracy and precision. The sound waves must be reflected in a straight line which means the reflective surface must be flat. In our case, the surface is constantly changing due to the motion of the cyclist. Because of this, there is a clear indication that ultrasonic mapping is not viable and will not allow for clear or accurate readings.

Radar was the clear competitor against Lidar. Changes in the density, acidity, or viscosity of the fluid (or air) do not affect the accuracy of the measurements, as compared to the Lidar that uses light waves as a medium and is easily affected by the medium itself (such as temperature, humidity, and moisture). The setbacks come with the accuracy and imaging. LiDAR uses light signals that work in smaller wavelengths that can allow for 3D imaging and more accurate and precise mappings, which we need to ensure we do not receive a significant amount of false positives.

As we wanted to ensure clear measurements and precision in our design, we decided to spend a bit more on our system as well as not care about different changes in the environment. This is why we chose LIDAR in our implementation.

C. Design Specification for Alerts

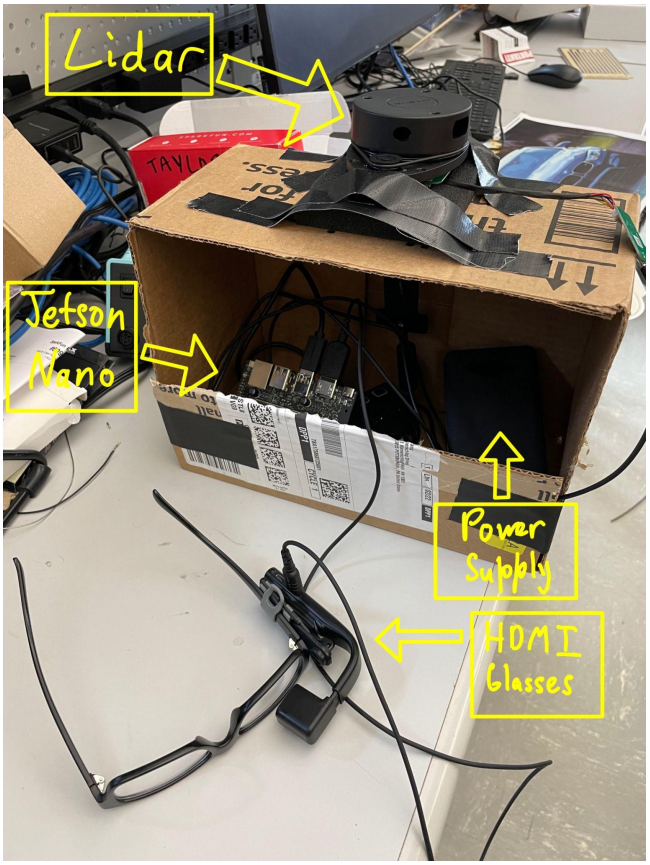
We decided to use visual alerts with HUD glasses instead of audio alerts with a speaker. This is because our use-case is focused on city cyclists. In the city, it is typically noisier than on the outskirts, hence, audio alerts can be misinterpreted for other sounds from the surrounding environment. Therefore, we decided to use visual alerts instead.

VI. System Implementation

One aspect of our design that has changed since the design report is that we had to use a different camera due to issues integrating the SAINSMART camera. Instead, we used a microsoft webcam which had 720p resolution and a 75 degree field of view, instead of the +120 degree field of view we were trying to achieve.

The Slamtech LIDAR and the Microsoft camera coupled together serve as the peripheral of our project for detecting cars and pedestrians and also the distance from them. The object detection isn't being done on the camera but instead will be done on its output. The camera and LIDAR sensor will be connected to the Jetson Nano via a USB connection. With this we will be able to receive input from the camera and begin processing.

As for the structure of how the camera and lidar sensor will be paired, the lidar sensor will be placed as close to the camera as possible right above it so as to line up the 0 degree (center) of the lidar with the camera so they are both facing the exact opposite of the front of the bicycle. The Jetson along with the lidar sensor and camera was encased in a cardboard box. The design can be seen below:



(Figure 8) Rear View of the Ride-ar System

(Figure 9) Front View of Ride-ar System

Initially we had planned to 3d print the casing, however this proved to be difficult because of the planned dimensions. However, for the sake of proof of concept we improvised a housing module out of a cardboard box that would be similar to the actual housing.

The different components of our system will communicate using ROS.

A. Object Detection Algorithm

For processing the images received from the camera, the input will be processed with OpenCV where we can apply our real-time detection algorithm. After testing different object detection algorithms such as YOLOv4 and YOLOv3, we ended up choosing the tiny YOLOv3 algorithm as it gave us the highest fps given the limited hardware of the Nvidia Jetson.

Both these algorithms are also able to detect multiple objects, however, they can be configured when running these software so as to only detect specific objects. To satisfy our use-case requirements, these real-time detection algorithms will only be focused on detecting cars and pedestrians.

We modified the code inside the YOLOv4 code to output the bounding box coordinates of detected cars and pedestrians to a file. This information will be used by the Kinematic Algorithm to determine how far away the detected objects are. A ROS publisher node periodically reads from the file and publishes the bounding box coordinates on the "car_pos" topic. Then the Kinematic Algorithm subscribes to this topic to receive the bounding box coordinates.

B. Kinematic Algorithm

The LIDAR itself is already integrated with ROS. The LIDAR publishes its range data to the "scan" topic which the Kinematic Algorithm subscribes to to receive the data. A time synchronizer is used which synchronizes the messages containing the range data from the LIDAR and the bounding box coordinates from YOLO.

The raw LIDAR data includes:

- minimum_angle (starting angle)
- angle_increment (difference in theta between measurements)
- a list of ranges

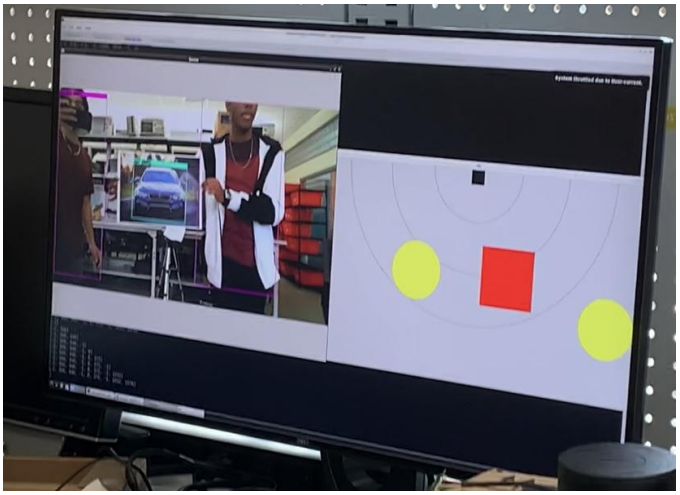
To determine the range to an object at a specific angle theta to the LIDAR, index into the range list by dividing theta by the angle_increment and rounding to the nearest integer.

Since we know the camera has a horizontal field of view of 75 degrees, we can linearly map each pixel along the x axis in the image to a specific angle. For example, a pixel that is 100 pixels away from the center of an image if the image size is 1000 is 100/1000 * 75 = 7.5 degrees from the center.

We use the coordinates of the bounding box of a detected object to calculate the angle to the left side of the object and the right side of the object. Then, using the angle measurements, we check the slice of the range list in the LIDAR between the left and right angles of the object. If any

of those ranges are below a certain detection threshold, we display them on the GUI.

The horizontal pixel coordinates of the bounding boxes determine the x coordinates of where the warning indicator is displayed, and the y coordinates are determined by the range to the object by the LIDAR. The yellow circles represent people, and the red squares represent cars detected.



(Figure 11): GUI displaying the cars and people detected in frame

The GUI is then output via HDMI from the Nvidia Jetson to the HUD glasses.

VII. Test, Verification and Validation

A. Results for YOLO Algorithm Error Rate

For the accuracy of the predictions, we desired a <10% error rate with which the error rate implies the rate at which cars or persons are misclassified as something else. For our tests, this was very simple to do with people as we set up the camera in the lab and walked back and forth through the camera's view and printed the prediction in the terminal as well as the accuracy of that prediction. We would also try facing different angles such as, our back to the camera, sideways and in front. We did 20 trials of this test and the algorithm did very well with predicting people. It was able to classify people correctly every time with an average accuracy of 98%. This would mean a 0% error rate which would pass our use-case requirement of <10% error rate.

For cars, we initially downloaded a Youtube video of a highway traffic camera and ran the YOLO algorithm on this. Cars that were closer to the camera had a higher accuracy than cars that were farther away, however, even far away the algorithm was still able to accurately predict that the object was a car. Testing with a live feed proved to be more difficult as our system was initially bound to the lab. For these tests, we printed out large pictures of cars at different angles and held them in front of the camera. We would move around and bring the car closer to and further from the camera. Just as before, as we brought the picture further the accuracy would decrease, but it would still accurately predict the car as an object. Out of 20 trials, the car was able to be predicted correctly with an average accuracy of 80%. This would also mean a 0% error rate which would pass our use-case requirement of <10% error rate.

Also this error rate could be seen as after the different subsystems were fully integrated, and the lidar data was also used, the correct drawings would be made each for a person and a car.

B. Results for Latency Use-case Requirement

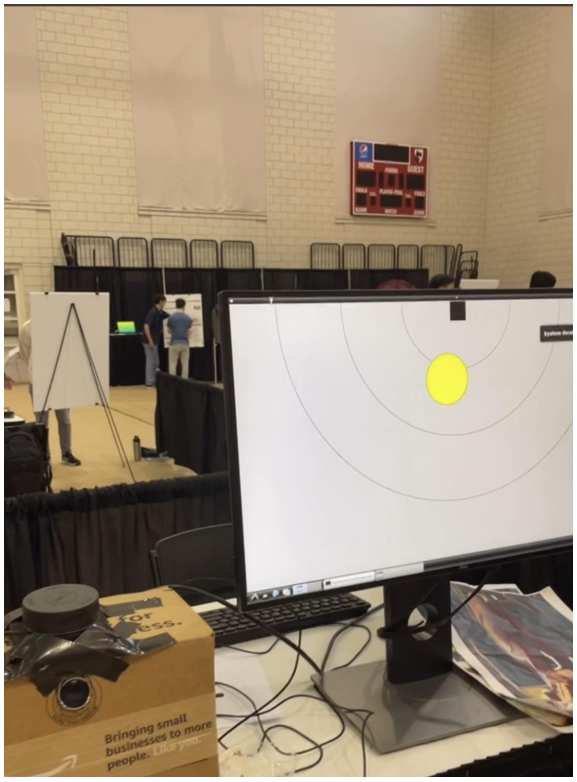
We tested the latency of the entire system by first calculating the latency of the YOLO subsystem. This was done by using the value of the average FPS of the system whilst predictions were being made on the camera feed. The time taken for the predictions to be made on each frame was calculated by doing 1/(Average FPS).

After this we found the latency of the GUI update by using python timers within the script. We were able to find an average of 20 times it takes to update the GUI being displayed on the HUD glasses. This average was used as the latency of the GUI update. For the lidar, there was negligible latency so this was not considered in the calculations.

The total latency was calculated by adding up the latency found above for each subsystem. The total latency of the system was found to be 0.43 seconds. The majority of the latency was from the YOLO algorithm itself as it was found to be 0.4s. This exceeded our desired latency which was 0.3s, however, because we were using a Jetson Nano, which has much less processing power, this result was anticipated. For example, if a Jetson Xavier was used, the YOLO algorithm could have been run much faster to achieve much less latency. Also because the latency was so high, we could see issues while we were testing on cars driving by which will be discussed next.

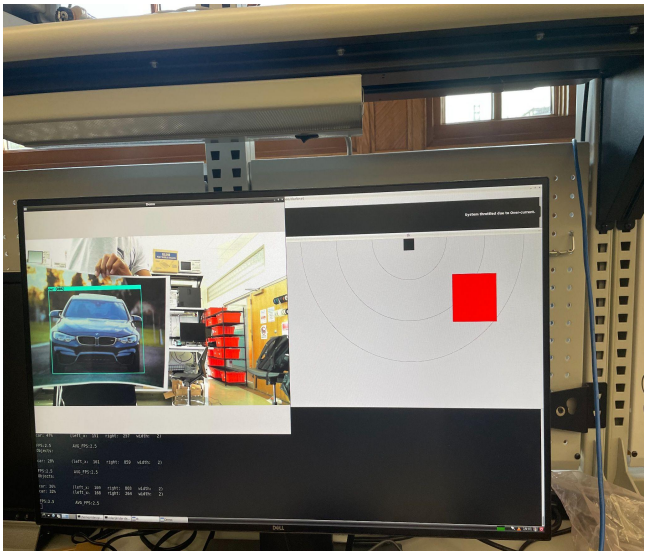
C. Results for Alert Design Specification

For the visual alerts, we tested this by having people walk across the camera, closer to and further from the camera in the lab. This worked really well as you could see the yellow circles being drawn corresponding to the number of people in range of the lidart and being predicted by the YOLO algorithm. The only issue with this is sometimes, the YOLO algorithm would draw 2 bounding boxes around a single person, thinking the person is more than one, and because of this on the GUI would draw two yellow circles almost overlapping each other. Below is an example of people being drawn on the GUI.



(Figure 12) GUI with Prediction of a Person

For cars, we were able to get an external power supply that could supply enough current to the system. With this, we took the system outside and tested it on cars in the parking lot. We tested it on cars driving by and parked cars. It was able to accurately draw the red boxes we intended for cars, however, because of the high latency, we would see problems where for cars driving by quickly, it would go by really quickly on the GUI and not show the box moving smoothly but instead flash in and out at different spots. This was anticipated because of the high latency discussed in the previous sub-section. Also, the lidar we were using for this system was a lidar intended for indoor use only, so we believe while using outside, the sunlight was interfering with the lidar detection. Below is an example of a car being drawn on the GUI.



(Figure 13) GUI with Prediction of a Car

D. Results for Power Use-case Requirement

With the battery used for our system which was able to provide 5 V, 3 A is sufficient to keep the Jetson Nano powered and running functionally. We tested this by letting the power supply power the Jetson with both the YOLO algorithm and Lidar script running from 100% to 1% and timed this. With this power supply fully charged beforehand, it was able to keep the entire system running for ~3 hours. This fails our use-case requirement of 6 hours, however, there are definitely batteries that could power the system for this long and even longer. The power supply purchased for this project just about fit the budget and hence why we chose it.

E. Results for Weight and Size Use-case Requirement

The entire system all together was weighed to be 2.89 pounds with a balance from a physics lab which is <3 pounds and passes this use-case requirement. The dimensions of the box were measured with a ruler to be 0.5ft x 1ft x 0.7ft. The volume of the system would thus be calculated to be 0.35 ft^3. This passes our use-case requirement of being <0.5ft^3 so as to make a small footprint on the bicycle.

VIII. Project Management

A. Schedule

This is a basic overview of our schedule. Please see the Gannt chart below for a fully laid plan of our timeline. Our project has several different phases that are dependent on one another, the first being the initial/setup phase. Here, we are working towards setting up the system and ensuring that all parts are ordered and can function with one another. Phase two involves implementing the core part of our design. We intend here to implement the Kinematics algorithm, connect the Lidar and camera to the Jetson, and develop the circuit that

encompasses everything including the glasses. By the end of phase two, we should have a basic MVP that meets the basic framework of our guidelines. The last phase, phase 3, would be our testing phase. Here we would iron any wrinkles without design and ensure that all use requirements are met. We intend this phase to be the most extensive as the majority of our system can only be refined through a significant amount of practice data.

B. Team Member Responsibilities

We intended to have the work and responsibilities of our team members evenly split with one another. They are divided and assigned mainly through the ECE area as well as playing to each of our strengths and experiences. Each member is responsible for two main tasks, roughly one for the first half and second half of the project.

1) Fayyaz was responsible for integrating the camera and Lidar together with jetson such that they can speak to each other and other Jetson oriented tasks as well as designing and implementing the circuit to integrate glasses display when the algorithm determines danger. He also designed the ROS node and system such that the camera and LIDAR data were able to retrieved in a central script that effectively drew our the rear view
2) Ethan was responsible for designing an effective module to cage the Jetson, Lidar, and Camera and attach it to the bike. He will also be responsible for configuring lidar sensors to ensure that we read precise measurements. From this as well with the data from the camera, he built the algorithm that drew the respective figures on the canvas depending on their bounding box coordinates and LIDAR measurements.
3) Chad was responsible for installing and implementing a computer vision algorithm to detect cars. This entails detailed research on the benefits and trade backs of different algorithms and testing several to determine the best response and feedback. He also worked with Fayyaz to design the ROS node and system such that the camera and LIDAR data were able to retrieved in a central script that effectively drew our the rear view

C. Bill of Materials and Budget

Please see Figure 14 for a list of equipment we used and their respective costs. We ended up using all the components that we ordered.

What differed from the original Bill of Materials, as indicative from our final implementation, is instead of using the Sainsmart camera, we opted to use a Microsoft camera provided for us by Professor Savvides. Not only this, but we ordered the heads up display glasses. This was mentioned in our initial design report but was not bought until later.

D. Risk Management

For our risk management, whilst testing the system, we noticed that the lidar processing did not add much to the systems performance and that it was mainly the YOLO algorithm that would cause the entire system to heat up. Instead of getting a cooling fan, we were able to overclock the Jetson Nano by maxing the processing power that Cuda uses. With this, we were able to assign more processing power for the YOLO algorithm and the heat sync alone was enough to dissipate the heat produced and keep the overall system relatively cool.

IX. Ethical Issues

With any ethical issues, there is some at play in our project as we are dealing with the idea of safety and having users placing trust in our system to keep them safe and out of danger. The most possible edge cases for our product and what could cause any ethical issues is seen when there is a car or person at a certain angle or speed that might not register on the system. The cyclists using our system would be the ones affected adversely by these edge cases. If the cyclist decided to move in the way of the unseen obstruction, there could be a highly dangerous accident that would be our fault. To mitigate the risks of this obviously improving our system such that we can catch these edge cases to ensure there are no cases where the cyclist is not aware of what is behind them. The system should also be ideally reviewed by professionals to ensure that our timing is sufficient with standard cycling practices.

The other potential ethical issues arise with the use of our camera. As we are using a camera to recognize persons and cars, both of which are kept private by a number of the population, there are issues where our system could be maliciously attacked. As our Nano does have internet capabilities and if someone decides to connect it to a network, it could be tapped in to and an attacker can see the camera feed. This could expose the privacy of the user or those surrounding the user and where they are going and even where they might live. To mitigate this, we should remove all network capabilities within our system such that this could not happen. We could also transition to a preinstalled software system that does not require a Nano such that there is no network presence at all.

Lastly, in relation to the first issue, there has been clear research done in the field of recognition theory about the individuals with darker complexion and how they are not as easily identified when trained on certain datasets. We should look into the datasets and conduct our own research to ensure that this is not an issue and there is not a problem where individuals like this are not being detected within our system.

X. Related Work

Regarding other related works, there are quite a number of projects that are similar to ours. The major differentiation between our project and the others is our incorporation of both Lidar and Object detection.

The majority of works only use Lidar systems. An example of this is a work done by University of Michigan-Dearborn professor Fred Feng, who developed an application for lidar that uses a lidar to develop a 360 degree view of a biker's surroundings. A retail product that is somewhat related to ours

is the Garmin Varia RTL515. It is a stationary one dimensional radar that alerts cyclists up to objects up to 140 m away. The benefit of both these products versus our own is through their accuracy of readings and depth of what they could accomplish considering their lack of other systems on board. What they are missing is precision as they are the only system on board and could potentially allow for false positives given no other methods to check the details of their surroundings.

There are also other systems that try to lessen accidents for bikers through other methods. NavTech has developed a birds eye detection that maps out roads in real time using satellites. Through this, they can indicate to bikers if there is a potential danger coming towards them. Obviously, there is a clear differential in resources available and we cannot achieve the same kind of efficiency as this.

XI. Summary

In summary, our project hopes to ensure a safer biking experience for city cyclists. This is through a system that uses object detection and Lidar implementation to develop a rear view picture for the biker and dictate whether or not a danger is approaching. Through this, we can reduce accidents and maybe even save lives. Overall, we achieved what we wanted to get done with the available resources and use-case requirements we defined initially. We are limited to the hardware that was available to us as If we had more time, we would want to refine the algorithm such that there would be a bit more precision on the shape of the object on the camera view. If one would want to continue use in the future, with more money and resources, much less latency could be achieved with a better processor and a further range with a better lidar. You could also achieve a better accuracy by using the non-tiny version of the Yolo detection algorithm which has much higher accuracy but less speed.

Throughout this process, there were a number of key lessons learned. The first being that integration takes time. It might be easy to believe that once you have all the subcomponents working that bringing them together might be relatively easier but this is not the case. As you are probably working on something this unique for the first time, there are always going to be some challenges regarding the interaction between all of your parts. Putting time in your schedule to learn about how your submodules work together would be incredibly beneficial.

Another important lesson learned is to account for bugs and errors. As projects can be ambitious and parts could not work as intended, making sure to account for these allows for ease of mind and not frantically trying to change your system and causing more issues. In tandem with this, learning how to pivot when something is not working or you are stuck is incredibly helpful. Without us changing to the new camera, we are certain that we would not have found the bug in time for the demo.

Keeping these lessons in mind, you can ensure that your team is not frantically scrambling to get your project working and you have clear foresight on what you are able to do and accomplish

Glossary of Acronyms

HUD - Heads Up Display
GUI - Graphical User Interface
ROS - Robot Operating System
YOLO - You Only Look Once (Algorithm)

References

[1] there, S. C. H. (2021, March 8). *Yolo v3 - install and run Yolo on Nvidia Jetson Nano (with GPU)*. Pysource. Retrieved March 4, 2022, from https://pysource.com/2019/08/29/yolo-v3-install-and-run-yolo-on-nvidia-jetson-nano-with-gpu/

[2] Nishad, G. (2021, February 26). *You only look once(yolo): Implementing yolo in less than 30 lines of Python Code*. Medium. Retrieved March 4, 2022, from https://medium.com/analytics-vidhya/you-only-look-once-yolo-implementing-yolo-in-less-than-30-lines-of-python-code-97fb9835bfd2

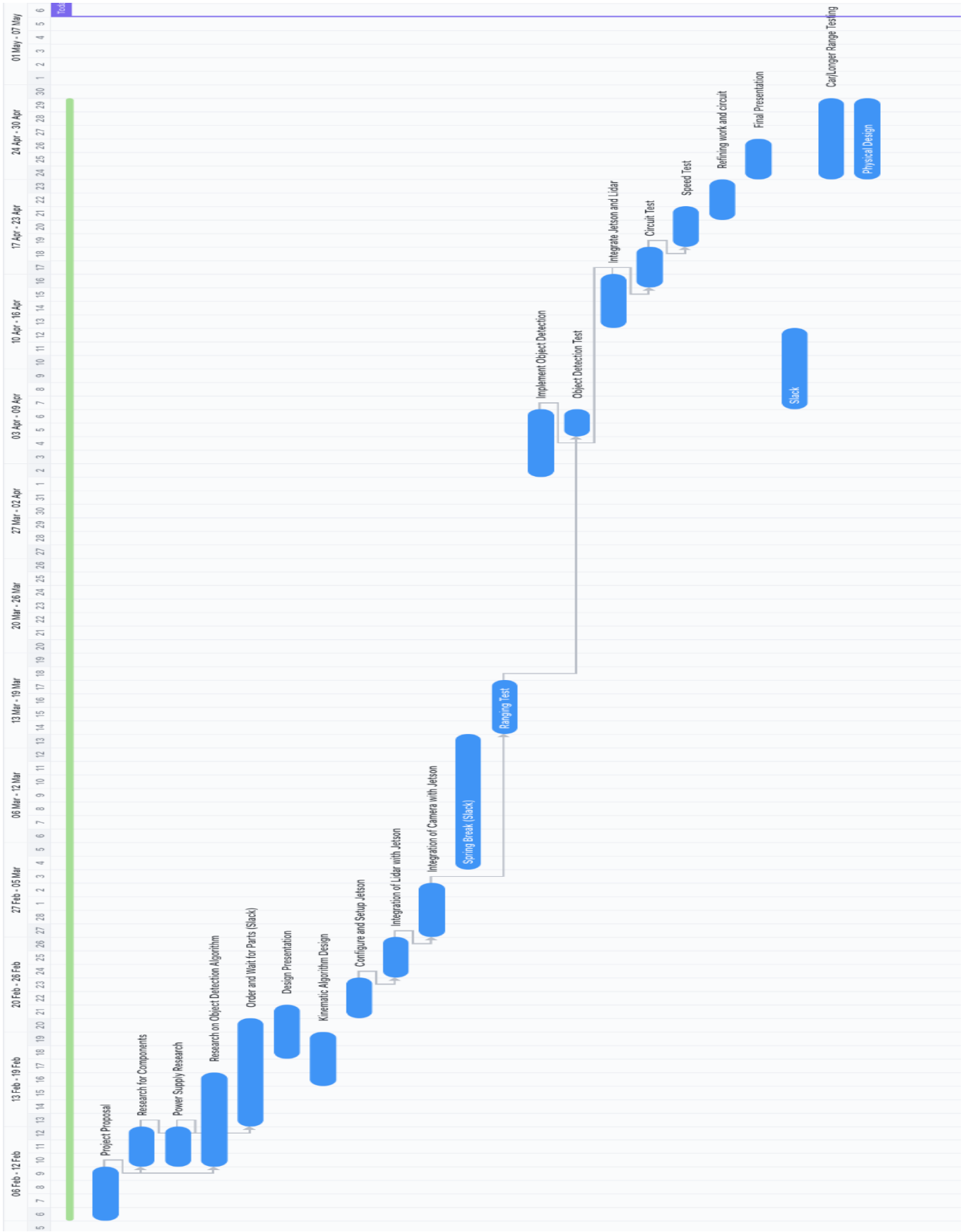
[3] Gandhi, R. (2018, July 9). *R-CNN, fast R-CNN, Faster R-CNN, YOLO - object detection algorithms*. Medium. Retrieved March 4, 2022, from https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e#:~:text=The%20bounding%20boxes%20having%20the,than%20other%20object%20detection%20algorithms.

[4] Lidar vs radar: Detection, tracking, and imaging. wevolver.com. (n.d.). Retrieved March 4, 2022, from https://www.wevolver.com/article/lidar-vs-radar-detection-tracking-and-imaging

[5] Person. (2019, March 18). *Power Supply Considerations for Jetson Nano Developer Kit*. NVIDIA Developer Forums. Retrieved March 4, 2022, from https://forums.developer.nvidia.com/t/power-supply-considerations-for-jetson-nano-developer-kit/71637

[6] Difference between Yolo and SSD. GeeksforGeeks. (2021, July 18). Retrieved March 4, 2022, from https://www.geeksforgeeks.org/difference-between-yolo-and-ssd/

[7] Getting started with Jetson Nano Developer Kit. NVIDIA Developer. (2022, January 29). Retrieved March 4, 2022, from https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit



(Figure 14) Schedule and Tasks

Jetson Nano 2GB	1	Nvidia	CMU ECE Depa	Brains of the System acting as the CPU	$0
Slamtech A1M8 LIDAR	1	Slamtec	Amazon	Used to scan surroundings and return direction and distances of surrouding objects	$99.99
Sainsmart IMX 219 (Not used)	1	Microsoft	CMU ECE Depa	Camera used for Object Detection	$21.99
Microsoft Lifecarm Studio	1	Microsoft	CMU ECE Depa	Camera used for Object Detection	$0.00
5V Power Adapter	1	SoulBay	CMU ECE Depa	Used to power Jetson for testing locally	$0
Battery Pack 5V 26800mAh	1	Ayeway	Amazon	Used to power Jetson for actual system	39.99
Keyboard	1	Logitech	CMU ECE Depa	Used for Jetson Peripherals	$0
Mouse	1	Logitech	CMU ECE Depa	Used for Jetson Peripherals	$0
Vufine VUF-110 Wearable Display	1	Vufine	Amazon	Used to act as heads up display for the cyclist to see the drawn canvas	$199.29
				Total	$361

(Figure 15) Bill of Materials