# 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 intend to do 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.**

***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.

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 SAINSMART IMX219 camera. The Slamtech LIDAR can detect objects up to 12m away, sweeps at 5.5hz, and has a 360 degree field of view. The SAINSMART camera has a 160 degree field of view and supports 1080p @ 30fps and 720p @ 60fps video recording. The output of these two sensors will be piped to a Nvidia Jetson Nano which will perform the object and range detection of the cars. We will use ROS to interface with the LIDAR and collect the range information. Using the data from the object detection algorithm and the LIDAR, we develop 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.

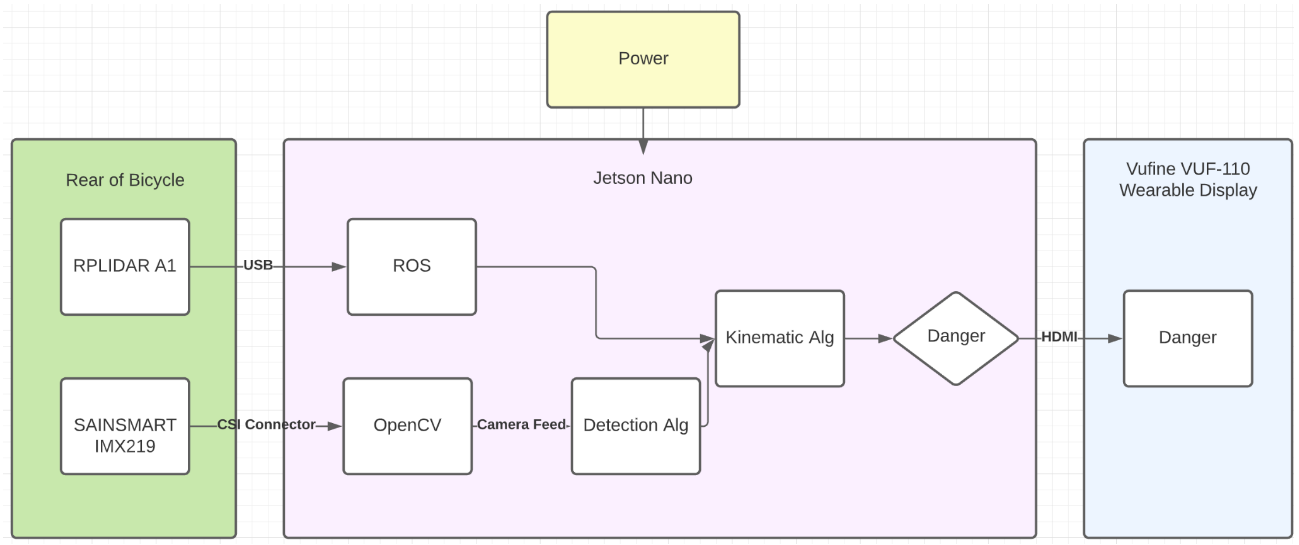


*Figure (1). There are three main components in the design. The sensor module which will be mounted at the rear of the bicycle, the computing and power module which takes input from the sensor module, and the display module which the biker will wear which displays visual information regarding oncoming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

*Figure (2) Slamtech LIDAR*

*Figure (3) Nvidia Jetson Nano*

*(Figure 4) SAINSMART Camera*

*(Figure 5) Vufine Wearable Display*

## 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.

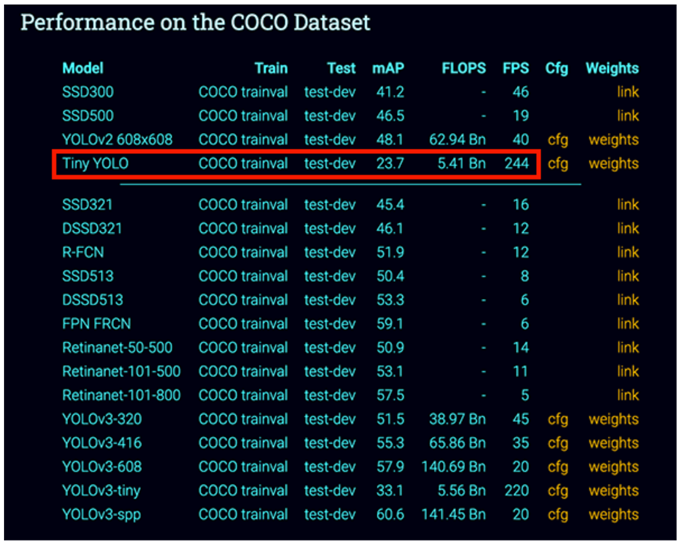
We want our warning system to be able to trigger within 0.3 seconds. The bulk of the computation time is the object detection algorithm. We are experimenting with different algorithms, but one algorithm we are looking at is the lightweight YOLO v3 algorithm. We want to optimize our system and algorithm for quick throughput and latency.

## 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 cause 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 will also implement 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 6) Accuracy and detection speeds of different SSD and YOLO 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.

The SSD real-time detection algorithm is also another famous algorithm alongside YOLO. YOLO has more popularity because of its speed and accuracy, however, there are some versions of SSD that have better accuracy than most YOLO versions.

We will also be trying the SSD 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 will test with both algorithms 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.

## *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.

## *C. Design Specification for Alerts*

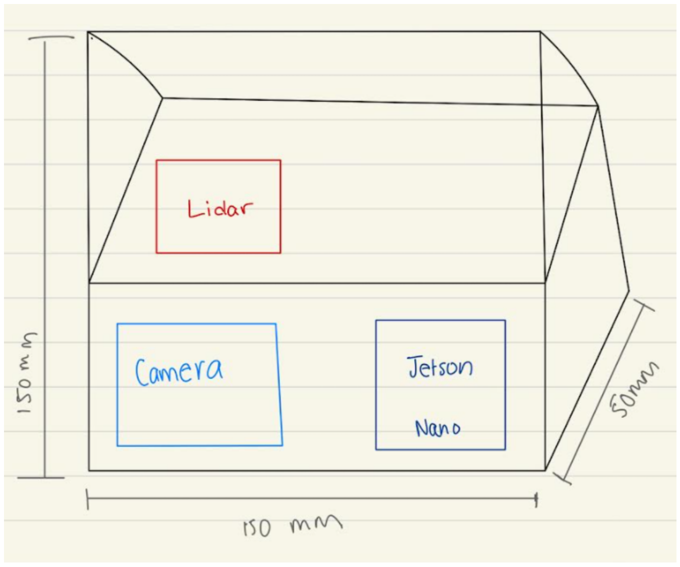
We decided to use visual alerts with HUD glasses instead of audio alerts with a speaker. This is because our use-case is focused on city cyclists. In the city, it is typically noisier than on the outskirts, hence, audio alerts can be misinterpreted for other sounds from the surrounding environment. Therefore,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

## VI. System Implementation

The Slamtech LIDAR and the SAINSMART 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 exactly being done on the camera but instead will be done on its output. The camera will be connected to the Jetson Nano via a CSI 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. This is to make processing results with the kinematic algorithm much easier.

The Jetson along with the lidar sensor and camera will be encased in a 3D printed case to stabilize the components. The case will look something like the figure below:

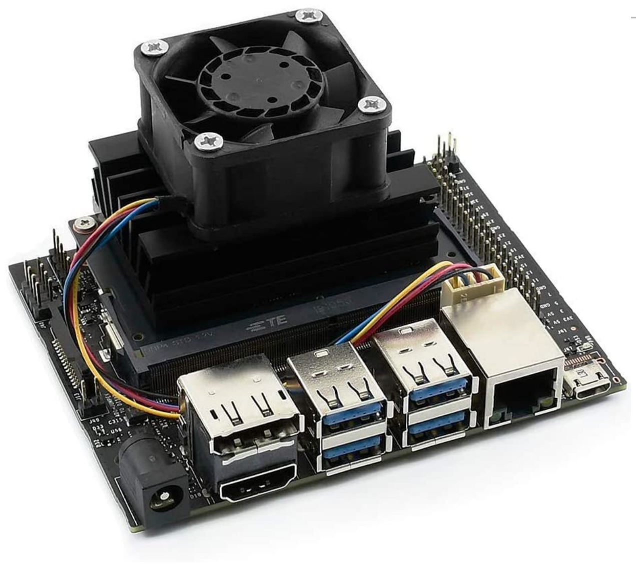


*(Figure 7) Case housing most of the components*

The lidar sensor will also be connected to the Jetson Nano instead via USB. All of the data being collected by the lidar will be processed using ROS. A ROS environment will be installed on the Jetson Nano along with the necessary drivers for the Slamtech LIDAR which are available on open source repositories online.

We should then be able to receive distances from the lidar in an array format along with the corresponding angles from a defined center which would be 0 degrees.

A cooling fan may also be bought for the Jetson in order to cool it down when running the object detection algorithm on it. This is because the Jetson Nano could potentially overheat and shut down due to the processing power needed for the real-time detection algorithm. There is a 5V supply cooling fan specifically designed for the Jetson Nano: B01 Version PWM Speed Adjustment Strong Cooling Air Fan 40mm×40mm×20mm. Below is an image of how the cooling fan would fit on the Jetson Nano:



*(Figure 8): B01 Version Cooling Air Fan for the Jetson Nano*

Finally the output of the Jetson which would be the HDMI in our case will be fed directly to the HUD glasses which takes in HDMI input. The display on the glasses will then show whatever is sent to it from the Jetson.

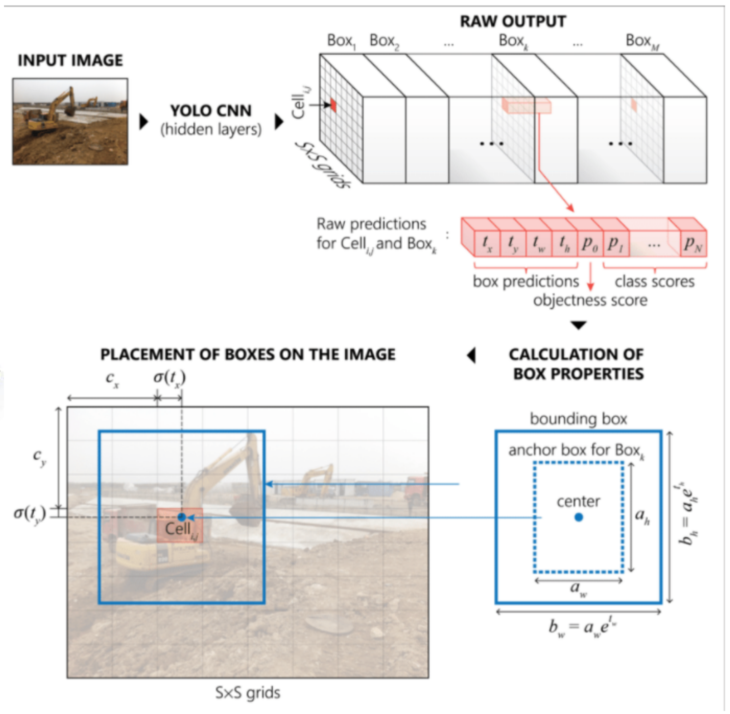
For powering this system, we intend to power the system through use of an outlet for the mvp. However, if time provides we will integrate our system with a battery pack that can support 5V and 2A to power the Jetson Nano which will be powering the rest of the system. The HUD glasses hold charge on its own and will not need any power from the Jetson.

*Refer to Figure 10 for all ports and connectors mentioned.*

## A. *Real-time 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. In our case, this would be either with the YOLO or the SSD detection algorithm. For the YOLO algorithm we will be experimenting with the darknet implementation which is an open source implementation of different versions of the algorithm such as Yolov3 and Yolov4. There is also a Jetson Nano specific version of the Yolo implementation called JetsonYolo which uses the Yolov5 version and can achieve 12 frames per second. We will also be using the TensorRT package developed by Nvidia to optimize TorchScript code. This package is an implementation of the SSD300 model which is based on the SSD: Single Shot Multibox Detector.

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.



*(Figure 9): Image showing brief description of the YOLO algorithm.*

We can see in Figure 9 above how the algorithm uses anchor boxes to detect objects which will be important for our kinematic algorithm.

## B. *Kinematic Algorithm*

For our kinetic algorithm, we will be using both the output from the object detection algorithm and processed data from the ROS environment. The speed of objects will be calculated from the lidar sensor using the frequency of the laser and the distance from the object. The frequency will give us the distance from the object every second. Once we achieve this we can calculate the speed based on a simple velocity equation:

$$V = \Delta x / \Delta t \quad (1)$$

Where V is the velocity, Δx is the change in displacement and Δt is the change in time which would be 1 second since we have the frequency.

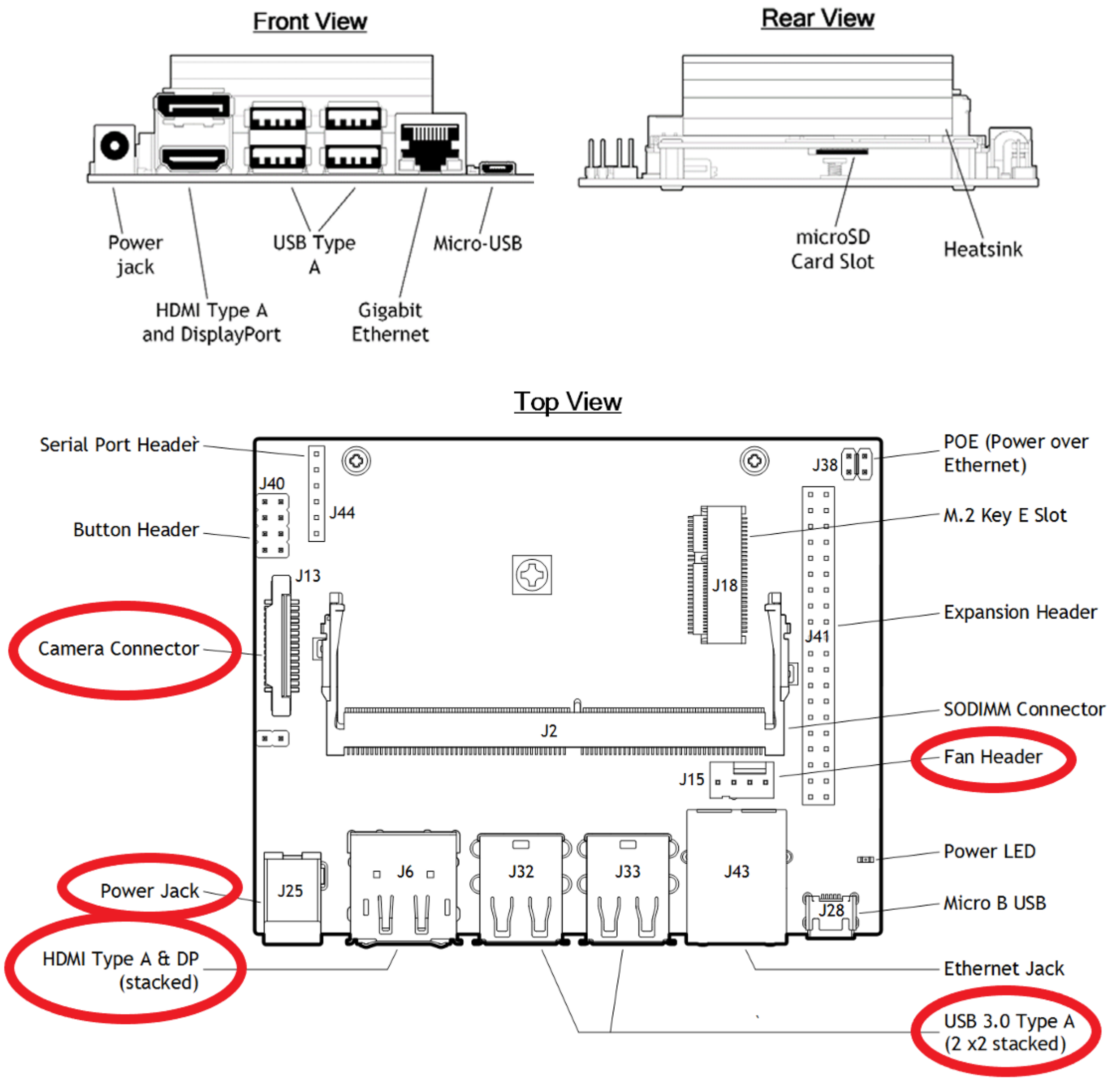
We would also have the angle from which the origin where this object was detected. This angle would be used to see if a car or pedestrian was detected in the vicinity of the image. Because the camera and lidar would be aligned, the center of the image would represent 0 degrees (origin). From here we will use tests initially to fine tune what angles align with what areas in the image.

Now if a car was detected in the area, we would now consider the speed of the car in order to determine the danger of the situation. The danger of the situation will be determined as seen below:

*Table I: Distances for different situation*

| Distance | Image Sent |
|---|---|
| 20 mph - 30 mph | Warning Image |
| > 30 mph | Flashing danger Image |

The warning image will involve a classic yellow warning symbol while the flashing danger image will involve a classic red danger symbol.



*(Figure 10) Layout of the Jetson Nano and all ports/connectors we plan to use*

## VII. Test, Verification and Validation

### A. *Tests for Use-Case Car/Pedestrian Recognition*

To test car recognition, we will place a camera in front of a vehicle at different orientations and distances. We will measure both the percentage of accurate car detection as well as the time needed to recognize the vehicle. We will also test in different lighting and weather conditions to see how the camera will react. This exact procedure will also be repeated for people as well for pedestrian recognition.

### B. *Tests for Use-Case Distance Detection*

We will move a car near the LIDAR at different points around the LIDAR sensor and at different distances. We need to be able to detect the presence of a car or pedestrian and also measure the distance to the object within a 1 meter margin of error.

### C. *Tests for Use-Case Danger Detection*

To test if the system can correctly detect a car approaching at a dangerous speed, we will either use preset speed values in the algorithm for certain approaching cars, or use a car to drive slowly in a controlled environment around the sensor to see if the system can detect it. Most likely, for faster speeds, to achieve danger signals we will use preset speeds within the code itself for safety reasons.

### D. *Tests for Use-Case Latency Testing*

We also want to be able to detect cars with very low latency. To test this, we will move a car into the field of view of the system and use timers within the code to measure how fast the algorithm is able to detect a car.

### E. *Tests for Use-Case Object Detection*

Finally, for object detection, we will be recording the accuracy and detection speed of predictions from both detection algorithms in similar scenarios and then comparing both. This will help us to decide which detection algorithm we will use.

## 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.

As of right now, we are towards the end of phase 1 and moving on to phase 2.

### B. *Team Member Responsibilities*

We intend 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 will be 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.
2) Ethan will be 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.
3) Chad will be 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 would also create an algorithm that uses kinematics, lidar output, and object detection to determine surrounding and potential danger and provide a sensible output to be interpreted and sent to the user.

### C. *Bill of Materials and Budget*

Please see Figure 12 for a list of equipment we used and their respective costs.

### D. *Risk Mitigation Plans*

- One risk is that because the Jetson Nano may overheat and shut down because of the processing power required for the real-time detection algorithm. The Jetson Nano will also be responsible for processing the lidar sensor data as well which will need even more processing power. For this reason, we plan to purchase a cooling fan designed for the Jetson Nano as the built-in heatsink on the board may not be enough to keep the system cool.
- Another risk we have is not developing an accurate system that takes into account all the factors that

surround a biker when they are driving. In other words, we are limited to the testing sites inside Hamershlag. We cannot fully test how our system would work traveling at 20+ mph as we are hardcoding our speeds.

## IX. 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.

## X. 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. Challenges in the future include testing limitations considering we cannot be in a city environment and ensuring our design works with the budgeted hardware we have as it is a bit extensive

## Glossary of Acronyms

| | |
|---|---|
| HUD | Heads Up Display |
| ROS | Robot Operating System |
| SSD | Single Shot Multibox Detector |
| 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

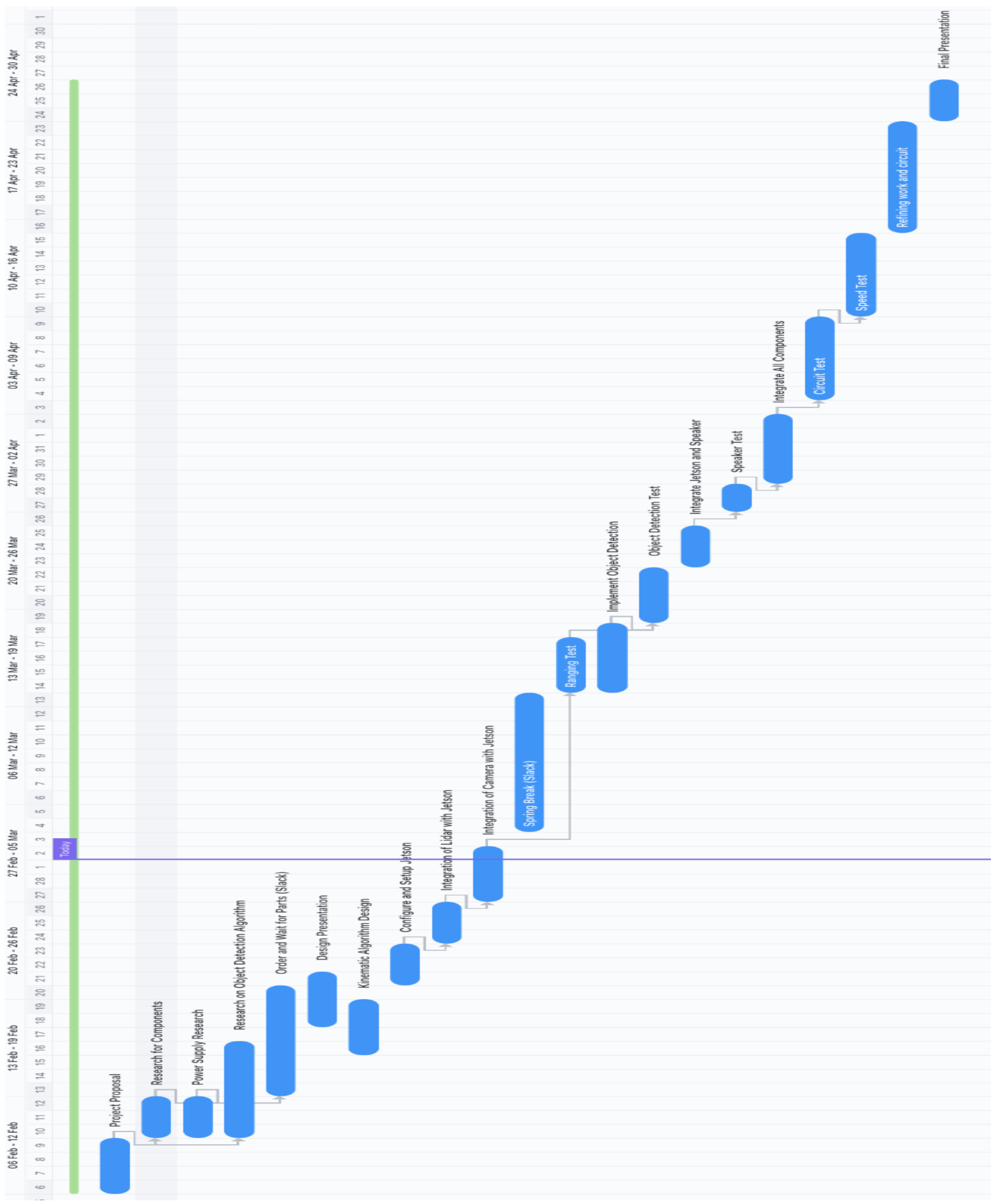
[3] Gandhi, R. (2018, July 9). *R-CNN, fast R-CNN, Faster R-CNN, YOLO - object detection algorithms*. Medium. Retrieved March 4, 2022, from https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e#:~:text=The%20bounding%20boxes%20having%20the,than%20other%20object%20detection%20algorithms.

[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 11) Schedule and Tasks*

| Materials | Quantity | Manufactorer | Source | Description | Cost |
|---|---|---|---|---|---|
| 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 IMX219 | 1 | SainSmart | Amazon | Camera used for Object Detection | $21.99 |
| 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 |
|  |  |  |  | Total | $162 |

*(Figure 12) Bill of Materials*