Cooperative vs Non-Cooperative Autonomous Driving

Authors: Tito Anammah, Serris Lew, Kylee Santos
Electrical and Computer Engineering, Carnegie Mellon University

Likely to be the future framework of transportation, autonomous driving has been the forefront of much research in the artificial intelligence field. We look to explore potential improvements to the current system, by experimenting the effect of vehicle-to-vehicle communication on the road. Given the scope of our project, we simulate the demonstrations with 6 inexpensive scaled-down model cars on a figure-8 track. After determining the pose and position of each vehicle through video processing, a centralized server uses this information to compute decisions for each vehicle and send these commands to them individually. We measure and compare the throughput between the non-cooperative and cooperative approaches in order to demonstrate clear improvements to the system.

Index Terms — *Cooperative mode*: cars communicating with one another to make a collective decision to optimize the overall system goals. *Non-cooperative mode*: cars making individual decisions based off of immediate surroundings to optimize its individual goals. *Throughput*: Number of vehicles that pass through central time given a specific amount of time.

1 INTRODUCTION

Autonomous driving will likely revolutionize the transportation industry in the next couple of years. Even more pioneering is the current research on cooperative autonomous driving. We aim to explore the benefits of vehicle-to-vehicle (V2V) communication and the advantages of cooperative decision making between autonomous vehicles. Having multiple vehicles cooperate could not only improve safety but also decrease traffic congestion.

There has been a lot of research on autonomous vehicles that make decisions solely based on their immediate sensor input. Simulating V2V communication implicitly extends a vehicle's sensory range and allows it to make more informed decisions based on its environment. It provides additional information such as another vehicle's intended path that cannot be fielded through sensor data. For demo purposes, we will have a figure-8 track setup with three cars on each loop. The center lane will be shared by cars from both loops, and our goal for this project is to achieve at least a 30% increase in throughput compared to a non-cooperative scenario.

2 DESIGN REQUIREMENTS

Our track size was dependent on the size and the speed of each vehicle. We used these metrics in order to determine a minimum following distance between vehicles so that they would still be able to safely come to a stop. We defined the total following distance, s_0, as consisting of 3 subdistances: latency distance, stopping distance, and a buffer distance. **Latency distance** is the distance the vehicle travels during execution of the server's computation, including object detection, localization, and path planning. **Stopping distance** is the distance the vehicle travels between receiving a stop command to actually coming to a complete stop. **Buffer distance** is the distance between the stopped vehicle and the obstacle. A breakdown of these distances is shown below on Figure 1.

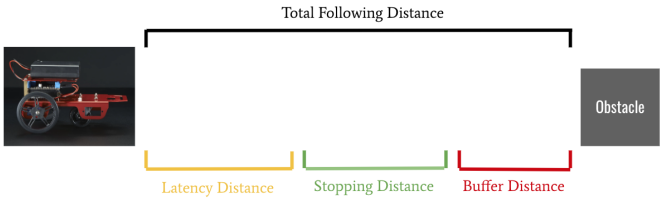

Figure 1: Breakdown of total following distance

In order to compute the minimum necessary circumference of each circle of the track, we used the following equation,

$$c \geq (n+1) * (s_0 + l), \tag{1}$$

where l is the length of each vehicle and n is the number of vehicles on each circle of the track. We acknowledged that another vehicle from the second circle might occupy the shared lane of the figure-8 track. Therefore, each circle would need to have enough room for $n+1$ vehicles at a time.

In order to calculate specific values for these design metrics, we also set requirements for the latencies of each component in our system, shown in Table 1 below.

Number of cars on one track	3
Max vehicle speed	50 cm/s
Detection latency	100 ms
Computation latency	5 ms
Communication latency	50 ms
Latency distance	7.75 cm
Stopping distance	2 cm
Buffer distance	5 cm
Total following distance	14.75 cm
Length of each vehicle	15.25 cm
Width of each vehicle	10.16 cm
Circumference of one track	120 cm
Radius of one track	19.10 cm
Lane width in each track	15 cm

Table 1: Metrics for track design

Each of the distances were computed using the maximum vehicle speed, v_0, and the individual latencies, t,

$$s_0 = v_0 * (t_{det} + t_{comp} + t_{comm}) \quad (2)$$

This circumference value gave us a lower bound for our design, and we decided to double the circumference to give vehicles more room to operate, instead of always driving at the minimum following distance. We chose to do this in order to better simulate real world traffic.

We decided on a track size of 145 cm by 80 cm with an outer radius of 40 cm, illustrated in Figure 2.

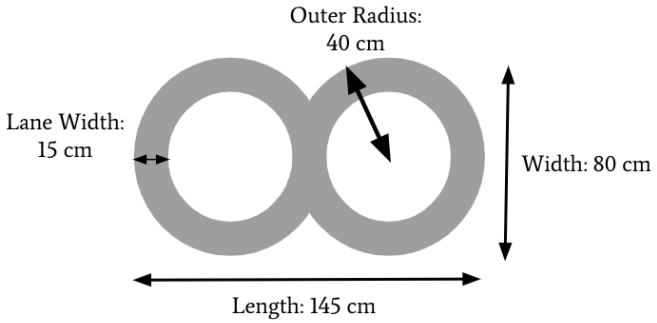

Figure 2: Diagram of final metrics of track design

The individual latencies used in Equation 2 are core to our design requirements as they define how many frames we can process per second and how often we can update each vehicle's paths. Detection latency is defined as the time taken to detect six vehicles within one frame using our ArUco Marker Detection. Computation latency is the time taken for server to localize the vehicles and perform the path planning computation. Finally, the communication latency is the time taken for the server to communicate data to the vehicles. We estimated these latencies in Table 1 based on preliminary testing of individual components of our system. These latencies are important goals for the project and are necessary for constituting success.

Another project requirement is to validate the performance of our localization method, which uses object detection and homography. We aim to have at most a 3 cm precision accuracy when detecting a vehicle's location. This is less than $\frac{1}{3}$ of the vehicle width and less than $\frac{1}{5}$ of the vehicle's length, so it should be accurate enough to use in our driver models.

After realizing that our motor control of each vehicle might be imprecise, we also added a requirement to ensure the vehicles stay within the circular track to a reasonable extent. Our track lanes are 20 cm wide, leaving a 5 cm margin of error on either side of each vehicle. Ensuring that the vehicles stay within the track lanes will validate a deviation of at most 5 cm from the center of the lane.

Based on our latency requirements we would require the detection algorithm to detect all 6 cars at least 99% of the time. Failing to detect a car in one processing iteration will result in the car travelling at least the latency distance before the vehicle can be detected in the next iteration. At our estimated maximum vehicle speed, this would cause a vehicle to cover roughly 5.75 cm. Because we doubled the following distance when determining the circumference of our track, unsuccessfully identifying 1% of the frames should not necessarily cause any collisions, unless back to back frames fail to identify the vehicle.

3 ARCHITECTURE OVERVIEW

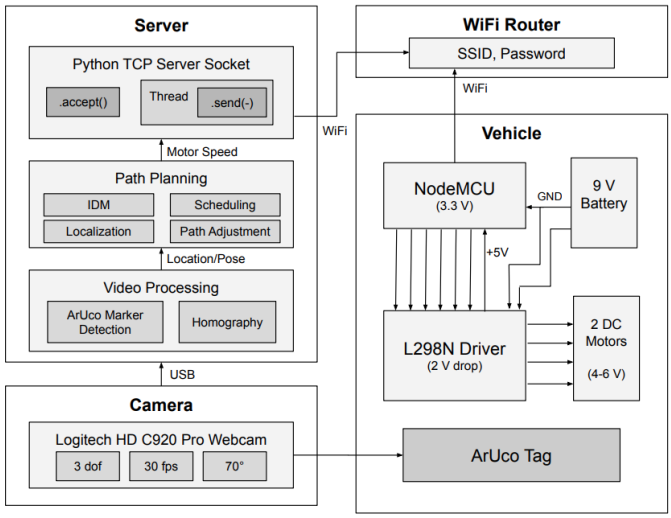

Figure 3: Block Diagram of our system

Our system architecture is broken down into 4 main components: Robotic Vehicles, Object Detection, Path Planning, and Communication. The block diagram in Fig-

ure 3 shows all of the different modules and how they connect/interact with one another.

3.1 Robotic Vehicles

For simplicity, the Vehicle block is representing one of the 6 identical vehicles that will be used in our design. As shown below in Figure 4, each vehicle will have a NodeMCU ESP8266 board as its microcontroller. Similar to an Arduino board, it has digital input/output pins that can be connected to other devices. For our design, 6 of its GPIO pins are used as inputs to the L298N driver, which controls the motors. The connections from the NodeMCU to L298N are the following: ENA-D5, ENB-D6, IN1-D8, IN2-D7, IN3-4, IN4-3. The ENA and ENB pins control the speed of the motors while IN# pins control the direction of the motors. There are 2 digital pins per motor that define its control. The first pin enables or disables the motor moving forward while the second pin controls the same motor moving backward. The motor controller is powered by a 9 V battery that grounds both the NodeMCU and the L298N driver. The module has an built-in 5 V regulator jumper that changes the behavior of the 5 V pin. We have the 5 V jumper enabled so it is used as an output source to power our NodeMCU board. The driver also has a voltage drop of about 2 V so with a 9 V battery, it will have a max voltage of 7 V for the motor terminals. However, the DC motors connected to it run between 4-6 V so the voltage drop does not affect its behavior.

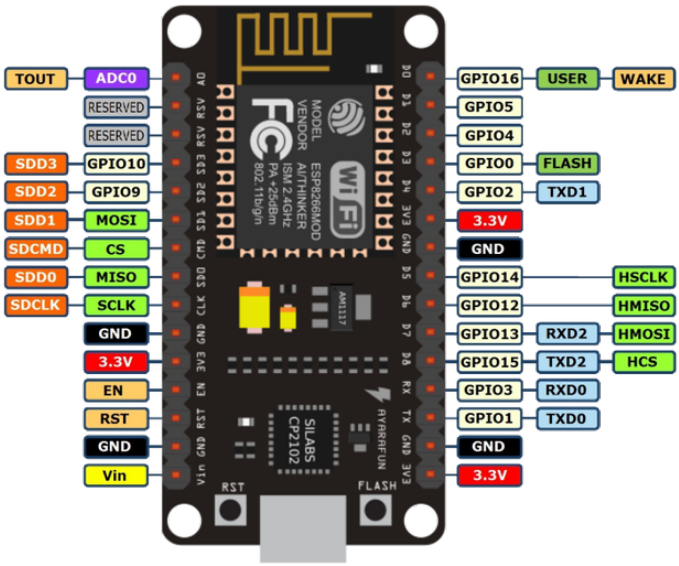

Figure 4: NodeMCU Pinout

3.2 Object Detection

Mounted at the top of each vehicle is an ArUco marker tag, which is a synthetic square marker composed of black and white patterns as shown in Figure 5 below. Used as a QR code, these robust markers will be detected by a Logitech HD C920 Webcam.

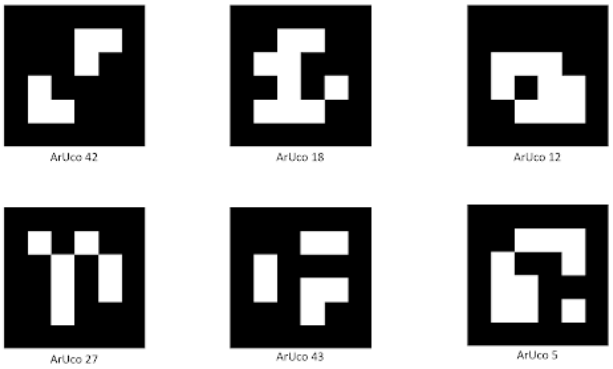

Figure 5: Sample ArUco marker tags

It will be positioned overhead the track to capture all of the vehicles as they circle the track. Connected through USB, this information is fed to the ArUco Marker Detection code to accurately identify all of the vehicles as they move along the track and to determine the location and position of each marker. Homography will then be applied to ensure that the location of each vehicle is accurate, regardless of the camera angle. It will warp the image to the correct perspective to provide the accurate relative location between the vehicles. This information will be given to the path planning module of the server's computation.

3.3 Path Planning

More specifically, each car will use the Intelligent Driver Model (IDM) which is a time-continuous car-following model; this will be used for the non-cooperative case. For the cooperative case, in order to make collective decisions, scheduling algorithms will be added to prevent starvation for vehicles when they meet at the cross center lane. Detection of future path collisions will also be included in determining a vehicle's path. As there may be subtle variability in each car, path adjustment will ensure all cars are moving on their intended path. All of the decisions made for both the cooperative and non-cooperative modes will return as a byte to represent the concatenated speeds of each motor. This will indicate to the vehicle the direction and speed to set.

3.4 Communication

In order to send the motor speed to each vehicle efficiently, the server will need to be connected to all of the vehicles at the same. This is done by connecting the server and the NodeMCU boards on each vehicle through WiFi. This allows the server and multiple client vehicles to be under the same network which reduces connection latency. Making each NodeMCU a client, the server will first connect to all of the clients and create a separate thread for each one. It will keep track of all of them to be able to send

messages to them individually when needed. The server will specify the vehicle it needs to control and only send the speeds of motors to the appropriate NodeMCU. As mentioned in the beginning of this section, the speed of the car will be relayed to the NodeMCU and then the L298N drivers that are connected to the DC motors.

4 DESIGN TRADE STUDIES

4.1 Localization

The main goal for this project is to simulate vehicle-to-vehicle (v2v) communication and to show its advantages in autonomous driving systems. To most accurately simulate v2v communication, we would have to supply all of our robotic cars with their individual sensors and cameras. This way, each car would be able to determine its location and pass on that information to other vehicles nearby. After some consideration, we decided to instead have a single global camera which could detect all the cars at any point in time.

The alternative individual camera approach introduces a much more difficult localization problem. Although each car has its own camera and can see its surroundings, it would have no way of determining its global location relative to the entire system. The solution would be to mount GPS sensors on each vehicle, however, GPS coordinates have an accuracy of about 5m which would be too imprecise on our small scale model. Given that each vehicle is 15 cm x 10 cm, it is important to ensure that the location of each vehicle is accurate enough to prevent collisions and miscalculations. While there are algorithms for robot localization such as particle-filtering, we concluded that we would need to invest a lot more time to get the localization accurate enough for our purposes and doing so would only detract from our main goal of exploring the benefits of cooperative autonomous driving. Using a global camera, we would be able to accurately determine every car's location relative to the track, and it greatly simplifies our problem.

4.2 Object Detection

4.2.1 YOLO-V3 and Image Thresholding

The two object detection algorithms we initially considered were YOLO-v3 and image thresholding. The former is a robust method for object detection but we feared that it would not perform well for our purposes. The YOLO-V3 neural network is primarily trained on the COCO dataset which is a dataset of common items such as people and cars. Our robotic cars however do not resemble real cars and so they would be difficult for the network to detect. To solve this problem, we had the option of training the network to recognize our cars but that would require creating our own dataset which may be time consuming.

Furthermore, our project demands that the detection of the vehicles meet mission critical latency requirements. This is important for the vehicles to have enough time to take any necessary actions to avoid collisions, and passing every video frame through a neural network would prove to be too slow for the task. The latter object detection method seemed to be the better candidate for the job. By assigning color codes to each of the cars, we would be able to do some segmenting to identify the vehicles. Though we would have to apply some morphological transformations on the image frames, this sort of computation is a lot cheaper in comparison with that of a neural network.

4.2.2 ArUco Marker Detection

We came across yet another object detection framework, ArUco Marker detection. This detection scheme makes use of QR codes for object detection and boasts high accuracy and speed. Realizing that an image thresholding algorithm would be susceptible to varying light conditions, we chose the ArUco detection because it is a more robust alternative for the conditions of our experiment.

4.2.3 Homography

One problem we noticed in our design was the potential for inconsistencies in our track/camera set-up. When moving our camera and track around for test and demo purposes, we tried to keep the camera directly over the track but there were still slight, inevitable shifts that would cause our view of the track to be more angled than hoped for. This would also lead to inconsistencies with the localized car positions and their assumed location on the track. To combat this problem, we have introduced a pre-processing step that will compute a transformation matrix (homography) to warp all localized car positions to a reference frame. This homography would be computed once before any cars are detected and during the demonstration, all we need to do is apply a matrix multiplication to each car's location in order to map the pixel location to a reference frame. A matrix multiplication is not a computationally expensive operation so this process would fit well into our pipeline.

4.3 Vehicle to Server Communication

In a comparison of cooperative and non-cooperative autonomous driving, the focus is not necessarily to optimize communication between vehicles or even to implement v2v communication. Rather we aim to demonstrate the benefits that cooperation through v2v communication may have. Therefore, it does not affect our proof of concept project if we are simulating v2v communication using a central server or if we actually implement it.

As mentioned in Section 4.1, the global camera system provides the location and orientation of all the cars in our system. For the cooperative case, the data from the global camera system would need to be relayed to all

of the cars to then be communicated to each other as a form of vehicle-to-vehicle communication. However, this communication pipeline adds redundant latencies for sending and receiving data between vehicles. Since the server has all of the system information from the global camera system and it must communicate with each of the vehicles regardless, we decided to eliminate the intermediate step and have all vehicles communicate to a centralized server. This essentially imitates vehicle-to-vehicle communication because each vehicle ultimately makes informed decisions, knowing complete data from all the other vehicles in the system.

4.3.1 NodeMCU

With a centralized system, all of the computation is done on the server and not on each individual car device. Therefore, we wanted to find a microcontroller that can control motors and have multiple of them be connected to the same server. After much research, we decided to use a NodeMCU ESP8266 board for the logic circuitry part. With a WiFi-based firmware, we thought this board would be ideal to create a communication system for one server and multiple clients; the laptop would be the server and each vehicle would be a client. Having one WiFi network that connects all of the devices would simplify the communication processes. One alternative that we looked into was using Bluetooth, as that is one of the other most common ways of wireless connections. However, Bluetooth connections are one-to-one, thus the server would have to disconnect its current connection before making a new connection. Since the cars are almost always in motion, this communication latency could greatly affect how the cars will make decisions.

Though the NodeMCU board is developed from Espressif Systems, it is still compatible with Arduino IDE and can be implemented as an Arduino board. This board can also run on its own compared to an ESP8266-based module that would have to be attached to a separate Arduino board. We also faced a tradeoff between cost and CPU performance with the selection of boards to pick from. Because the computation was done on the server, we were able to pick the cheapest option, not needing a dual-core processor such as the ESP32.

4.3.2 Python Interface

To determine each vehicle's next move, the server would need data from the global camera system that provides information about each vehicle's pose and location relative to the other vehicles. Since this connection system is using OpenCV in Python, we wanted the server to also implement its connection with the client in Python. At first, we looked into Micropython, a software implementation for a microcontroller compatible with Python; this would allow us to utilize Arduino code in a Python interface. After some testing, we found Micropython to be sufficient for our solution, but unnecessary since the Arduino code only needs to be used for the clients (NodeMCU). In other words, since our centralized server is sending messages to the clients, the server can be written in a Python interface and send commands to the clients which has Arduino code uploaded in its firmware. Therefore, Micropython was ultimately excessive in our implementation.

With that being said, we started testing with sockets (socket programming) to allow different programs to send and receive data at any given moment. To be compatible with the data received from the camera, we implemented in Python which also provides a socket class that can be easily integrated in our source code. Data would be sent via WiFi by defining a centralized host and port for all the vehicles to connect to.

4.4 Motor Control

Another design change we made was changing from the L293D IC motor controller to a L298N motor driver shield. Though the former can control the speed and direction of 2 motors simultaneously and is compatible with the NodeMCU board, the latter can do the same and also does not overheat with higher voltages. The raw L293D driver chip was able to provide us flexibility to wire it up as desired, however we realized that we weren't using it in some custom-built way but instead a more standard usage of the driver. Therefore, we decided to prioritize consistency between vehicles over flexibility over the chip schematic diagram. For that reason, we decided to use a motor shield, which is a circuit board with soldered connections on it including an on-board voltage regulator and a heat sink for overheating problems; it made the circuit more efficient and safer to use. We also wanted to eliminate any variability in the cars as our main focus is the cooperative and non-cooperative modes and not the cars themselves.

4.5 Steering Geometry

The most common steering mechanisms for vehicles are the Ackermann and the Davis steering systems. These both require a sliding pair of front wheels and a fixed set of back wheels. The defining feature of these approaches is that the steering angle of the front wheels determines where the center of its turning circle is located. Figure 6 below depicts the Ackermann mechanism. This is advantageous because the vehicles are able to control their speed and direction independently. Their speed is naturally defined by how fast their motors are spinning, and their direction is determined by the steering angle, where the specific path can be solved using Ackermann steering geometry.

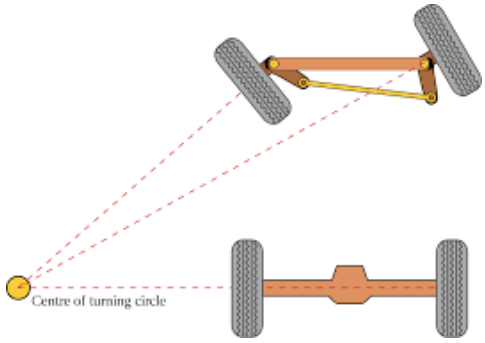

Figure 6: Diagram of Ackermann steering mechanism

The 4-wheel setup with a complex axle system was too expensive to use in the scope of this project. Robotic vehicles with this steering mechanism were primarily the pre-built remote-controlled cars that were either out of our budget range or required a lot of tinkering in order to repurpose them for our project. Thus, we settled for two DC motor-driven rear wheels with a third central wheel for stability, as shown in Figure 7 below.

Figure 7: Underneath view of wheels on vehicle chassis

In this 3-wheel setup, speed and direction are no longer independent. Because the wheels cannot turn, the turning radius of the vehicle is determined by the ratio of speeds in the left and right motors. Creating a speed difference between the two motors would cause the vehicle to turn towards the direction with the slower speed. The drawback of this approach is that we need to create our own geometric equations in order to model the turning point as a function of the two motors' speeds.

4.6 Shape of Track

The paper [**2**] that we wanted to model our project from used a two lane track consisting of concentric circles. However, from a path planning side, this required both longitudinal and lateral movements. Not only would we need a car-following model to simulate highway driving, but we would also need a lane-changing model in order for the vehicles to maneuver around obstacles safely. We decided to change this design into a figure-8 track where each set of cars only travels in their respective circle of the track. This would allow for the vehicles to follow a simple circular path, without changing lanes, and focus on a car-following/object detection model. When making decisions, each car would only have to change it speeds as opposed to both its speed and direction. The differences between the two tracks are shown below in Figure 8.

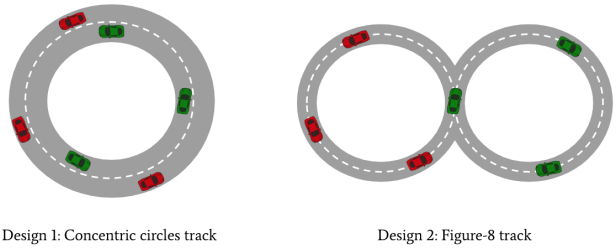

Figure 8: Comparison of Proposed Track Designs

4.7 Car-Following Driver Model

With the new focus on the longitudinal movement, we had a couple car-following models to select from. We decided to use the Intelligent Driver Model instead of the Newell or Gipps's Model. The Gipps Model is a relatively simple model, developed in the 1970s and details the relationship between each vehicle's position, velocity, and braking severity. However, the Intelligent Driver Model is more recently developed, and its main purpose was to improve on Gipps's model, since the latter loses realistic properties in the deterministic limit. The paper [**2**] that attained similar goals to our project also makes use of this driver model, which can give us guidance on how to implement it.

5 SYSTEM DESCRIPTION

5.1 Robotic Vehicles

For 6 cars, we initially decided to go for a thriftier approach and get individual parts to assemble. To reduce the amount of variability between the cars, we also decided on a minimalist approach. This included a NodeMCU (discussed in next section), a L293D IC Motor Controller, 2 DC gear motors, 2 wheels, a wheel ball, 9V battery and a mini breadboard. After assembling the car, we noticed 2 problems with this design. First, the construction of the car was difficult to maintain without any structure to it. Though all the necessary parts were included in this design, there was no frame to keep car intact, which would make it more inconvenient to ensure all of the cars were built the same. As a result, we decided to buy Adafruit's Mini Robot Rover Chassis Kit. The kit included 2 wheels, a support wheel, 2 DC motors, a metal chassis and a top metal plate with mounting hardware. This kit not only gave structure to our cars but also was affordable for our budget. It will make our fleet of cars more uniform and more realistic in the context of our project. In addition, the large flat mount on top of the car was an ideal place to

put an identifiable tag for our camera system. Ultimately, the kit adapted to our overall project better and we are using this hardware design when building the rest of the cars.

5.2 Communication

With the NodeMCU ESP8266 as our logic board, we promptly started testing its connection with Arduino IDE. In order to do so, we downloaded the CP210x drivers that allows the computer to recognize the serial port of the board when connected through USB. We then set up the Arduino IDE by downloading the ESP8266 community board manager package. With the toolchain setup, we ran into a few issues connecting and getting the computer to recognize the board as a ESP8266 board. After much research and debugging with multiple NodeMCU boards, we realized it was a manufacturer error and finally got it to work with a NodeMCU from a different manufacturer. After the initial connection was made, we decided to test the NodeMCU with the L298N driver shield to test the motors. We connected the 4 output pins of the driver to the 2 motors (2 each), the +12V input and ground pin to the 9V battery, the +5V output pin (with the 12V jumper enabled in the driver) to the NodeMCU and the controller pins (ENA, ENB, IN1, IN2, IN3, IN4) to input ports of the NodeMCU. After testing, we noticed the motors only behave appropriately when the battery and USB cable are both connected to the board. When only the battery is powering the board, the motors behave unexpectedly. As a result, we suspected it is a power issue and are currently working to resolve this issue.

As mentioned in Section 4.3.2, we began testing with Python sockets for the communication system. For testing purposes, we connected both our server (laptop) and the NodeMCU board to the same WiFi network and was able to successfully send data. We tested by running the Python server script on terminal that prompted an input to send to the NodeMCU. For simplicity, we had the NodeMCU respond by blinking its built-in LED and printing the server's message on the serial monitor. We found that by sending a string to the NodeMCU took roughly 6 seconds, which was far too slow for our requirements. We had to reflect on what kind of data we would need to send and realized that with how our track is designed (figure-8) and how the cars are moving, we only need to send data for speed. Therefore, we reduced the number of bits being sent and changed our message type from a string to byte. The NodeMCU was able to receive bytes almost instantaneously; we are still working to get exact metrics on how fast the NodeMCU receives data. Since the client and server are on different devices, we have not figured out a way to relate timestamps across devices. One possible option that we thought about is using the time it takes for a server to send a message and receive a confirmation message back from the client as an upper bound for the communication latency. As an extension, we started testing with multiple clients connected to the network to see how data will be sent and received. To reduce the amount of data being sent, we decided that broadcasting all the data to all of the vehicles would be excessive. Even though this would make communication non-blocking, sending less data to each vehicle would be more efficient and require less computation for each vehicle to know which part of the data being sent is specific to them. Thus, we opted to create separate threads for each client and when the server needs to notify a specific client to move a specific speed, it would only send those bytes to that client. We performed a small trial with 2 clients and was able to successfully send different messages to different clients. We are planning on extending this to multiple NodeMCUs and verifying the communication system as we proceed with our implementation.

5.3 Camera

After researching into potential cameras to buy, we decided on the Logitech HD C920 Pro Webcam. Not only was this camera relatively inexpensive, but it integrated well with Python and OpenCV, which is the language and library that we intended to use for object detection. The camera also has a horizontal field of view of 70 degrees which we thought would be sufficient to view the whole track at a reasonable height. The C920 has good resolution and a frame rate of 30 fps which would suffice for our purposes. We also noticed that a couple of teams in previous years had also made use of the camera so we felt more confident about its performance. Alongside the camera, we also decided to get a mount that could place the camera directly above the track. We found a cost-friendly mount that could clamp onto a table or chair and hover the camera over track. The mount has over 3 degrees of freedom which allowed for much flexibility when configuring the camera setup.

To test the ArUco detection, we started by printing out 12 markers on a grid and assessed the algorithm's speed and accuracy on the markers. We ran the test a couple of times and chose the worst performance as a baseline. We defined the algorithm's accuracy by the percentage of frames where all 12 markers were identified. It was able to achieve 100% accuracy on all tests and a worst case computation time of 70 ms. We then tried spreading out the tags and again testing its performance. We printed out six tags and spread them out at about 2 feet away from each other. We mounted the camera above the markers at approximately 4.5 feet from the ground. The six tags were detected at roughly 28 frames per second which evaluates to about 35 ms for processing one frame. This processing time corroborates evidence from our test with the 12 tags which took about 70 ms to process. Lastly, we tested out the algorithms performance on moving tags. Since we did not yet have our robotic vehicles moving, we moved the tags in front of the camera manually. There were brief moments when the tags were not detected and we suspect that it might be due to a slight blur of the video frame

induced by a moving tag or some error with moving the tags manually. We do recognize the fact that this may lead to problems so we will run a more accurate test once we get the cars moving. In the worst case scenario, we can fall back to the image thresholding method which is more robust to blurring.

5.4 Car-Following Model

The Intelligent Driver Model relates the positions and velocities of individual vehicles based on the the vehicles directly in front. The model defines the following notation [**2**]:

- v_0 is the velocity the vehicle drives at in free traffic,
- s_0 is the minimum following distance necessary between vehicles,
- T is the minimum possible time to the vehicle in front,
- a is the maximum vehicle acceleration,
- b is the comfortable braking deceleration (positive number),
- δ is the acceleration exponent (usually set at 4),
- x_α is the position of the front of vehicle α at time t,
- v_α is the speed of the vehicle α at time t, and
- l_α is the length of the vehicle.

The model defines the net distance between two vehicles as,

$$s_\alpha := x_{\alpha-1} - x_\alpha - l_{\alpha-1} \tag{3}$$

and the approaching rate between two vehicles as,

$$\Delta v_\alpha := v_\alpha - v_{\alpha-1}. \tag{4}$$

Using Equations 1 and 2, the model is able to define the function of acceleration of an individual vehicle as a function of time,

$$acc_\alpha = a\left[1 - \left(\frac{v_\alpha}{v_0}\right)^\delta - \left(\frac{s_0 + v_\alpha T + \frac{v_\alpha \Delta v_\alpha}{2\sqrt{ab}}}{s_\alpha}\right)^2\right] \tag{5}$$

Due to some obstacles with powering the circuits onboard the vehicles, we have yet to test these parameters to define maximum acceleration, minimum time to vehicle in front, etc. Because the robotic vehicles in our simulation are not robo-taxis with passengers, we are able to increase our maximum acceleration and have harsher braking in order to loosen our constraints slightly.

5.5 Information Constraints

For the **non-cooperative** case, since we decided not to include individual sensors on the vehicles, we needed our constraints to accurately simulate what vehicles would "see" in a non-cooperative setting. We will enforce a radius around each vehicle and when making its decision, only expose the vehicle to the obstacles within the radius. The server, which will be performing the computation and decision-making on behalf of the vehicle, will plan based on this constrained information and relay the decision to the car.

For the **cooperative** case, the information available to each vehicle will also be bound by some radius. But a key difference is that each vehicle will also have information on the other vehicles' intended paths, which provides a much more thorough picture of the moving parts in the system. This extra knowledge does not benefit the vehicles in our case with a figure-8 track because each vehicle only has one path to travel in, thus the speed of the vehicle will also denote the vehicle's intended path. However, if we were to consider our previous track design with two lanes, a vehicle could continue in its own lane or change lanes. Even if a vehicle is slowing down, its intended path could be to come to a stop or to switch into the other lane. Knowing if/when a vehicle would change into a lane in the cooperative case would allow other cars to prepare ahead of time and slow down rather than react abruptly to the lane change. While this is important for real-world advantages to cooperative autonomous driving, we probably will not be able to explicitly demonstrate it in this project.

5.6 Scheduling Algorithm

A key difference in the cooperative case that will be highlighted in the figure-8 demo is that the vehicles will be able to make a collective decision that optimizes the entire system's goals. When examining contention in the shared center lane in the non-cooperative case, vehicles in the left track would view a stopped car in the right track as an obstacle that does not interfere with their path through the center lane. Thus, each car in the left track would continue through the center lane, starving the vehicles in the right track. However, to optimize the throughput in the overall system, we can implement a scheduling algorithm on the server side that will ensure fairness by granting cars that have been waiting longer by the center lane a higher priority to pass through.

This algorithm is similar to protocols defining how resources are shared between threads fairly, except in this case, the mutually exclusive resource is access to the center lane and each thread's priority is dynamic. We intend to devise this algorithm from scratch, and it could be as simple as keeping a running counter for number of ticks each vehicle has been waiting. However, the algorithm might include some minimum waiting threshold for a vehicle be-

fore its priority is elevated, and the solution would require a detailed driving model for the interaction between cars around the center lane in order for the yielding cars to slow down and allow the prioritized cars to accelerate through the center lane. It will be difficult to formally prove that this algorithm is optimal, therefore we will focus on just being able to achieve improvements in our throughput.

6 PROJECT MANAGEMENT

6.1 Schedule

Refer to Figures 9 and 10 for our project's schedules. As shown, there are 2 schedules: our initial plan (Figure 9) and our updated plan after making a few design changes (Figure 10). As we began implementing and testing our designs, we discovered better alternatives and improvements to our original design. Some of these updates including using the ArUco Object Detection, the shape and size of the track, the motion and interactions between the car and hardware implementation of the vehicles. Though we made changes to our plan, we anticipated this from the beginning and allocated enough time for implementation and testing. In both schedules, it is apparent from the different colors the parallelism in our work.

6.2 Team Member Responsibilities

By the nature of our project, we split the responsibilities into 3 parts: Robotic Vehicles/Communication, Object Detection and Path Planning. Serris is taking lead on the design of the robotic vehicles and how they will communicate with one another. This includes deciding on tradeoffs in order to maximize control over vehicles while minimizing communication latency. She will also work with Kylee to test the vehicles with various motor speeds to establish metrics for the driving models.

Tito is in charge of the global camera system. His tasks include deciding on an object detection model for the scope of our project. He will also work on incorporating homography or any other solution to ensure accurate localization of the vehicles. Tito will get metrics for testing this model and also help Kylee with the Path Planning algorithm.

Lastly, Kylee is in charge of the path planning algorithms for both the cooperative and non-cooperative modes as well as other computation necessary for keeping the vehicles on track. This entails being able to find the relative positions of the vehicles and the track. It also requires implementing the Intelligent Driver Model, and devising then implementing a scheduling algorithm for the cooperative case.

6.3 Budget

Refer to Table 2 for our project's budget. This includes components we have already purchased and components we plan on purchasing. As shown below, there are some parts that overlap in functionality including individual parts for a car and motor driver. As mentioned in Section 5, these purchases were due to design changes that we thought would simplify and benefit our project.

6.4 Risk Management

Our project involved many components which none of us had much experience with. In order to mitigate the budget risk, we decided to get the bare minimum that we needed of each component in order to begin preliminary testing and assess its performance. After researching possible robot car models, we settled on two relatively cheap robot kits. One of them was slightly cheaper than the other but also wasn't as sturdy. We decided to get one of each and compare the two before choosing a design for the rest of the cars.

We realized that our solution approach was very reliant on the communication between the server and the cars so we decided to test out the simplest scenario of server communication. Since the server will be connecting with all the nodes we decided to get two NodeMCUs which was the minimum amount needed to test out multi-client server communication. We reasoned that once we were able to communicate to two NodeMCUs, we could then scale out to multiple clients. We made the order for our parts as soon as possible so we could get familiar with the different components of our project. Doing so allowed us to discover problems early on which allowed us to make the necessary design changes. For example, we noticed that it took roughly 6 seconds for the server to communicate a string to the NodeMCU while it took practically no time to communicate a byte. These tests allowed us to reconsider our design for what and how we were communicating in order to meet our requirements. When calculating our budget, we took into account the possibility of damaged parts so we made allowances for extra components if needed. We plan to have 6 cars on the track but budgeted for 8. We also made the decision to do all the path planning on the server rather than on the microcontroller. This lessens the load on the microcontroller enabling us to get a less powerful and cheaper device like the NodeMCU as opposed to a more capable but expensive Raspberry Pi.

For every decision made, we tried to have at least one other fallback option. Instead of the NodeMCU microcontroller, we also held the option of an ESP32 chip mounted on an Arduino Uno for the server-client communication. We also considered communication via Bluetooth. For the object detection, we had 3 options in mind; the YOLO-v3 neural network, Image thresholding, and ArUco marker detection. Upon hearing of the ArUco detection library, we immediately began testing its performance on our project setting. We tested the detection speed and accuracy and confirmed that they met the requirements. When benchmarking the performance of the object detection library,

we focused on the worst case performance. We kept track of the longest amount of time it took to detect a set of markers, across multiple test iterations.

7 RELATED WORK

The Prorok lab supervised by a Cambridge University professor, Amanda Prorok, also worked on a similar project [**1**]. The project involved an experimental testbed consisting of 16 miniature RC vehicles. Their experiment was executed on a multi-lane track with an obstacle placed on one of the lanes. The demonstration showed how the communication between vehicles could prevent a buildup of traffic on the blocked lane. Their primary goal was on safety as they tested the fleet in both driving modes with normal and aggressive driving behaviors. In their experiments, from non-cooperative to cooperative driving, they were able to see a 35% improvement of traffic flow with normal driving and a 45% improvement with aggressive driving.

8 SUMMARY

From our initial solution approach, our project has undergone many design changes. However, they are a result of refining our design requirements and simplifying the problem to focus on our main goal: comparing the effects between cooperative and non-cooperative autonomous driving. To get an accurate measurement of our system's performance, we immediately began implementing and testing our designs to make the necessary design changes early on. From what we have encountered, we have been able to address most of our design challenges with concrete metrics through testing or potential alternatives as we continue to experiment. Overall, we believe we have made good progress in our project, and it will continue to advance as we finalize our project.

References

[1] Nicholas Hyldmar, Yijun He, Amanda Prorok. *A Fleet of Miniature Cars for Experiments in Cooperative Driving.* Paper presented at the International Conference on Robotics and Automation (ICRA 2019). Montréal, Canada, 2019.

[2] Treiber, Martin, Hennecke, Ansgar, Helbing, Dirk. "Congested traffic states in empirical observations and microscopic simulations", Physical Review E, 62 (2): 1805–1824. 2000.

Appendix A

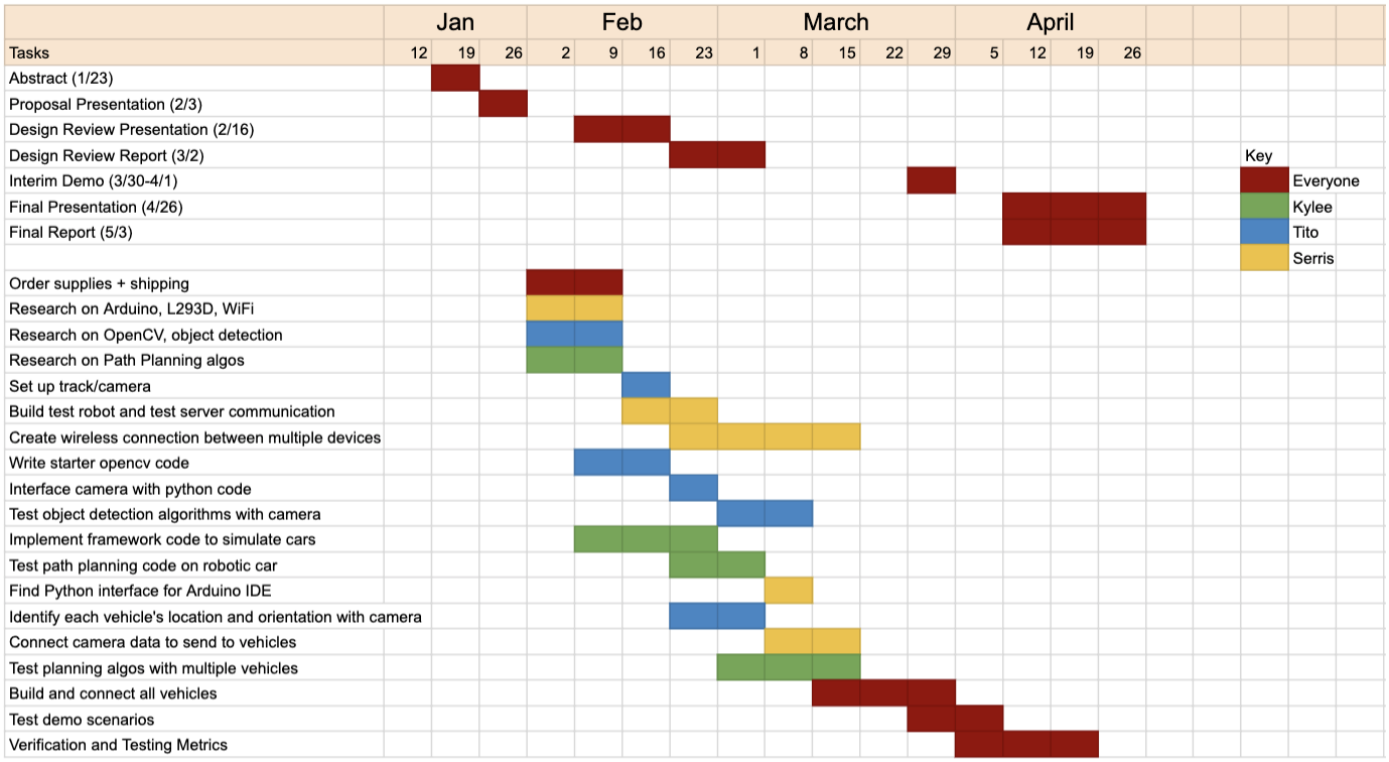

Figure 9: Initial Gantt Chart

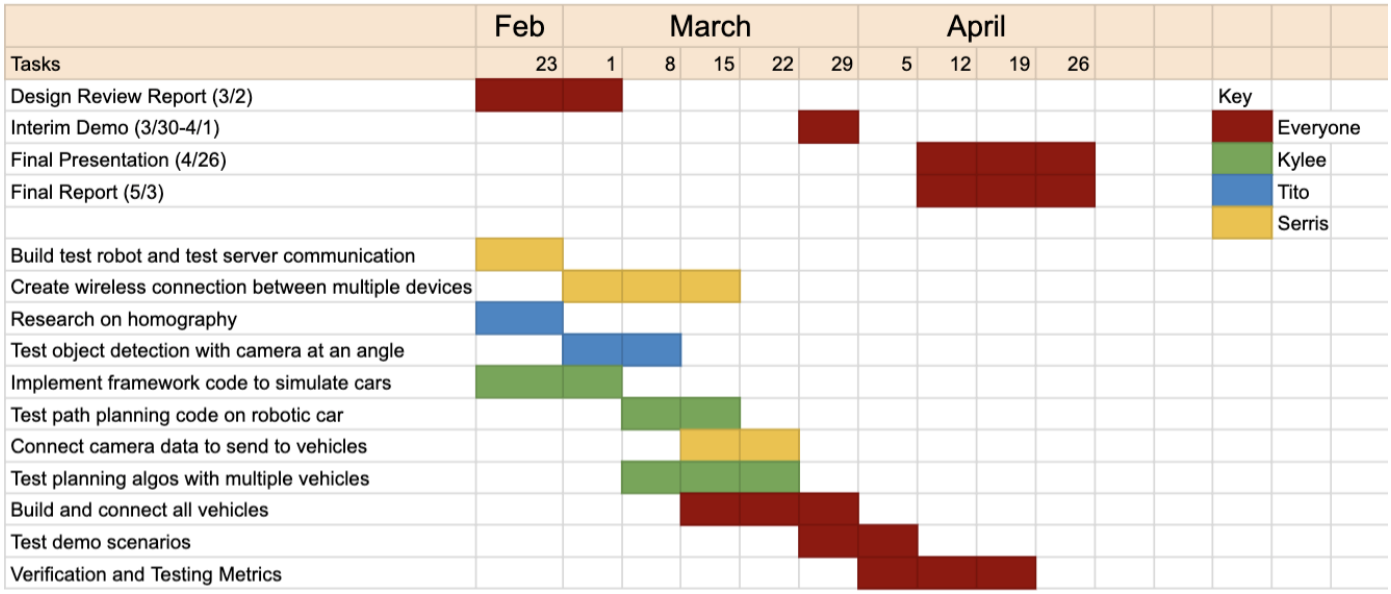

Figure 10: Updated Gantt Chart after design changes

Purchased			
Item	Quantity per Order	Order Quantity	Total Price
Mini Robot Chassis Kit	1	1	29.95
L293D IC	10	1	8.00
DC Motors + Wheels	4	1	14.59
Roller Ball	1	1	5.68
Mini Breadboard	6	2	13.96
NodeMCU	3	3	38.97
9 Volt Batteries	8	1	10.99
Logitech HD Pro Webcam C290	1	1	59.99
Logitech Webcam Mount	1	1	19.98
L298N Motor Controllers	5	1	13.99
On and Off Switches	12	1	9.88
TOTAL			**225.98**

Need to Purchase			
Item	Quantity per Order	Order Quantity	Total Price
Mini Robot Chassis Kit	1	5	149.75
L298N Motor Controllers	5	1	13.99
TOTAL			**163.74**

Used from Lab
Item
Jumper Wires
Battery Clips

Total Budget	
State	Price
Purchased	225.98
Need to Purchase	163.74
TOTAL	**389.72**

Table 2: Budget of tools needed for the project