AutoVot: A Vehicle to Vehicle Communication System for Autonomous Driving

Authors: Joel Anyanti, Fausto Leyva, Jeffrey Tsaw
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—Trends point to an autonomous future in the area of transportation. Paramount to the success of such technologies is the addition of safety and robustness features on these vehicles. The goal with this project was to demonstrate the usefulness of Vehicle to Vehicle communication in autonomous vehicles. As a result, we developed AutoVot, an autonomous vehicle convoy that leverages Vehicle to Vehicle Communication to navigate a course with obstacles. The lead car uses a depth camera to detect obstacles along the course to generate a path to navigate to the goal, all while communicating instructions to the following vehicle via Bluetooth. This allows both vehicles to safely navigate the course without collisions, even though the following car has no perception capabilities. Our resulting solution shows promise for communication in the domain of autonomous vehicles.**

***Index Terms*—Autonomous Driving, Object Detection, Vehicle to Vehicle Communication, Nvidia Jetson Nano, Convoy.**

1 INTRODUCTION

The transportation industry has recently turned to autonomous driving solutions to perform transportation tasks previously exclusive to human drivers. Indeed as the technology that enables self-driving becomes more reliable and commercially available, an autonomous future looks to be inevitable. Autonomous driving technology in its current state relies on individual vehicles to make decisions based on self provided sensory input and perception. As impressive as the technology may become, we recognize that coordination between vehicles would provide additional safety benefits and increase the overall efficiency of transportation networks. To demonstrate this claim, we aim to develop an autonomous driving system which leverages vehicle to vehicle communication to perform a coordinated task.

For our development we have chosen to focus on implementing a convoy system between a lead and follow vehicle. The 'lead' vehicle will be fully equipped with image sensing to allow for object detection and path planning. By contrast, the 'trail' vehicle will be blind and rely solely on information from the 'lead' to navigate the course. The two vehicles will be tasked with driving from one end of a track to another while avoiding any obstacles in their path. With this task we intend to highlight the benefits of communication between vehicles and ultimately hope to achieve zero collisions in our verification runs.

2 DESIGN REQUIREMENTS

In order to best accommodate our circumstances we considered the physical attributes of the vehicles to parameterize our testing course. We recognize that at the scale we are developing under, attention to these details are critical to defining thoughtful requirements. In particular we relied on speed estimates for the vehicles to determine the length of the straight away for our course. To estimate the speed of our vehicles we needed to factor in both the payload weight and the RPM of the motors.

Standard DC motors for Arduino based RC vehicles are rated at $200 \pm 10\%$ RPM @ 6v. The wheel diameter for the corresponding motors are 60mm. With the following equations we are able to estimate V_{max} for the vehicles:

$$\omega = 2\pi f \tag{1}$$

$$v = r\omega \tag{2}$$

The resulting velocity we obtain from these equation is $0.64m/s \pm 10\%$ but we must also factor in the added weight of the vehicle components into the overall speed of the vehicle. The required components (camera, sensors, motor shield, Arduino, Jetson Nano, power supply) for the fully equipped lead vehicle are estimated to be around 600g (5.88N) of weight. We project that this will cost around 15% slowdown from the theoretical maximum of the motors. This leaves us with the following: $V_{max} = 0.54m/s \pm 10\%$.

Given these calculations we assume the top speed of the vehicles to be a generous 0.5m/s with the lead car. This projection allows us reasonable margin of error as the weight projects are overestimates and the 'trail' vehicles will have lighter payloads.

In determining the final length of the course, we wanted to maintain a reasonable length for adequate testing. A length that is too long would likely lead to reproducibility errors and difficulty in collecting data. By contrast, A test too short may be unimpressive and less able to illustrate the value proposition. Acknowledging this, we targeted a test run of 1 minute total time for completion of the course.

Given that a vehicle moving at 0.5m/s will reach about 30m maximum distance forgoing acceleration we opted for a 20m straight away course with a width of 1.22m. These values are based on the standard dimension for a track and field race track. A diagram of this track can be seen below in Figure 1.

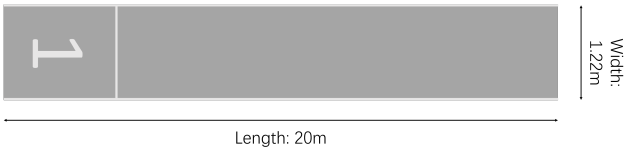


Figure 1: Diagram of task verification track

This length would allow for about 20 seconds of margin time for avoidance and navigation given our expected vehicle velocity. We consider this to be a reasonable amount of time given the requirement of successfully navigating two vehicles to the end of the track. The decision to use a straight away course with no turns is motivated by the fact that such a system would rely much less on localization/-global positioning which would cause greater difficulty in implementation.

Critically we must also acknowledge the vehicle avoidance portion of the development as a driving factor for our requirements. In essence we need to know what the minimum object detection distance is our solution needs to achieve to successfully avoid obstacles in its path. For this we consider the following three sub-distances: latency distance, stopping distance and buffer distance. Latency distance is the distance a vehicle travels over the time required to make one pass through the software pipeline. This includes object detection, path planning and communication. Stopping distance is the distance a vehicle will travel after a stop command before reaching a complete stop. Buffer distance is the distance left between the obstacle and the vehicle after the vehicle has stopped. This breakdown is clearly illustrated in Figure 2 below.

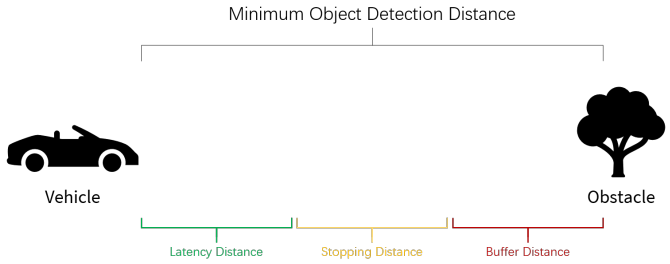


Figure 2: Diagram of object detection distance breakdown

'In order to obtain the stopping distance we estimated the distance the vehicle travels over from its max speed to a full stop. Using the following physics equations in relation the our vehicle parameters,

$$v^2 = v_0^2 + 2a\Delta x \tag{3}$$

$$F = ma \tag{4}$$

The results from this calculation result in a stopping distance of 0.03m as we observe in Table 1.We note that since the buffer distance is a fixed parameter set by the group, there is no corresponding calculation. The numbers provided in Table 1 for the compute task latencies are rough estimates based on research into our methods of choice. We consider these estimates to be generous as they include slack for anticipated worst case conditions. With this we are able to calculate the latency distance using the following equation

$$x_{lat} = v_{max}(t_{det} + t_{plan} + t_{comm}) \tag{5}$$

This leaves us with a latency distance of 0.125m. Thus the overall minimum object detection distance we must target is 0.205m considering all of the sub-distances.

Vehicle Maximum Speed	0.5m/s
Object Detection Latency	100ms
Path Planning Latency	10ms
Communication Latency	100 ±40ms
Latency Distance	0.125m
Stopping Distance	0.03m
Buffer Distance	0.05m
Minimum Object Detection Distance	0.205m
Vehicle Length	0.2794m
Vehicle Width	0.1778m
Track Length	20m
Track Width	1.22m

Table 1: Summary of required metrics

For the autonomous navigation aspect of our project, we need he lead care to be able to detect static obstacles in the path and make planning decisions to successfully maneuver through the obstacles. To accomplish this, we must require that the convoy system maintains 0 collisions through the track, as no crash mitigation will be implemented. Given the track distance and the buffer, stopping, and latency distance, we calculated an object detection latency of 100ms. More specifically, this involves having the on-board stereo camera relay image information to the compute node, which applies an object detection algorithm once every 100ms. This is equivalent to developing an object detection algorithm that is capable of performing at 10fps, assuming negligible latency (i.e a camera latency much less than 100ms) between the stereo camera and the compute node.

Regarding the nature of the object detection, rather than implementing an object classification algorithm, we aim to overlay the image from the stereo camera with bounding boxes surrounding the object. This would give

a smooth estimate as to where the obstacles in the path are, and thus make planning decisions that produced the least risk of crashing into obstacles. Given the combined stopping and buffer distances is 0.08m, we require any obstacle to be detected within 0.2m in order to maximize the avoidance probability without imposing too strict requirements on the vehicle hardware. Assuming a frequency of object detection performed at 10fps, and a vehicle speed of 0.5m/s, this would give an average 2.4 frames before the vehicle reaches the buffer and stopping distance and jeopardizes colliding with the obstacle. For this reason, we require our object detection algorithm to be able to perform at 90% precision and 95% recall, in order to minimize the probability of missed detections and subsequent collisions. With 95% recall, the probability of not detecting the object before the vehicle reaches the danger area reduces to 0.25%. Notice that we prioritize recall, since we want to minimize false negatives more than false positives. In the worst case, this would translate into having our object detection detect phantom objects on top of actual obstacles, which would not be a problem assuming our planning algorithm is sufficient.

Figure 3: Example bounding box from object detection algorithm

For planning, we simply need an algorithm capable of making navigational decisions based on the object detection algorithm's output and the current camera feed. For this to be successful, we require a latency of 10ms, to allow for the vehicle to dynamically adjust and make smooth turns to navigate around obstacles. The planning stack will be located on the onboard computation node, and communication between the compute node and the motors will be done via an onboard Arduino for simplicity. Using an Arduino as a motor controller would allow the planning stack to output target angles for the wheels to turn rather than PWM signals. This gives greater flexibility to the development of the planning algorithm.

Since we have a lead car equipped with the sensory equipment (camera, distance-tracking sensors) to help it maneuver through the track, we need the lead car to be able to effectively and accurately communicate with the rest of the convoy. We are going to be using Bluetooth as our mode of communication across vehicles which must be reliable and relatively low-latency.

One important assumption we are making is that the Bluetooth conditions are ideal for communication; meaning our track will be free from huge obstructions between the transmission of Bluetooth signals. We are not expecting the latency of Bluetooth messages to exceed 100ms. Upon our research we found that the ideal Bluetooth latency is around 34ms, but can range up to 100-300ms, so we are taking the lower bound since our cars will be in close proximity with a clear path between them. Communication will consist of relaying localization information (map updates) perceived by the lead car, out to the trailing cars. We are expecting the Bluetooth connection to be reliable enough for the lead vehicle to use a broadcasting protocol to accurately send updates on the position of the other vehicles; since we will be sending messages frequently, we expect that a lost packet will not have a significant effect on the vehicles.

3 ARCHITECTURE OVERVIEW

The architecture of the Autovot system consists of 4 major subsystems, vehicle mechanics, object detection, localization/path planning and communication. These subsystems are clearly broken down in the system diagram which can be found illustrated in Figure 9 in the Appendix (see Appendix A). Worth noting regarding the architecture is the two different classes of vehicles that we intend to implement. The key difference between the two designs is that the lead vehicle will be equipped with a slightly more powerful Jetson Nano unit as well as a Intel RealSense camera for perception. The trail vehicles will run identical setup with the exception of the scaled down 2GB version of the Jetson Nano as well as not having any camera unit at all. The reason we opted for the less powerful Nano for the trailing cars is due to the fact that there will be no image processing performed by these vehicles.

3.1 Vehicle Mechanics

The diagram outlining the overall system for Autovot is based on the lead vehicle to fully cover the extent of the development. For simplicity we can assume that the 'trail' vehicles are identical in regards to the core mechanical systems. The core driver for the mechanical system is the Arduino Uno R3 board which will house the logic for controlling the motor systems. To power the Arduino we will leverage the USB connection between the Arduino and Jetson to supply the required 5v. This connection will also be used to relay commands used to steer the vehicle. Our design uses an Adafruit Motor Shield V2 unit to control all 4 of the DC motors individually. This units sits onto of the Arduino to relay control signals over 12C protocol. To power the motors, the raw output from the 7.4V 2S LiPo

batteries is fed into the motor shield unit. This is sufficient to provide the required voltage to the motors and fits in the 5V - 12V range of the motor shield unit. A simple diagram of the shield connected to two DC motors can be seen in Figure 4 below

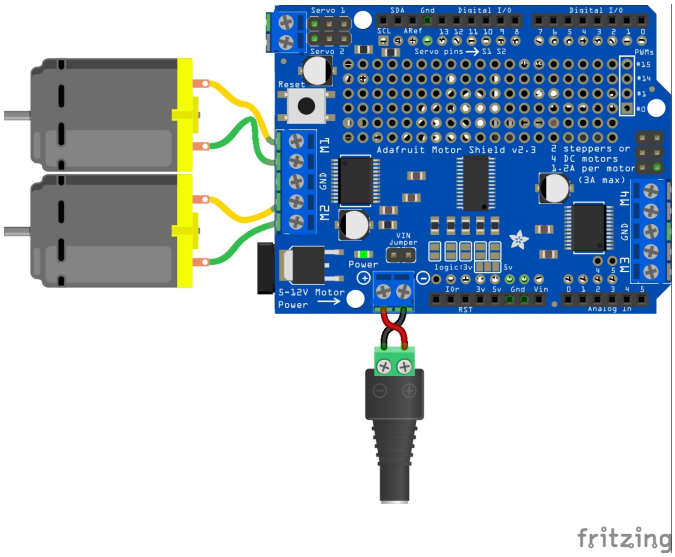


Figure 4: Diagram of the connection between the Adafruit Motor Shield and DC Motors.

3.2 Object Detection

The object detection node relied on an Intel RealSense depth camera to provide RGB-D images to the Jetson Nano. Using MobileNet v1 SSD, as well as the depth image, obstacles' (x, y) locations according to the car's frame are determined, and continually passed to the path planning node.

3.3 Path Planning

Path planning receives information from the localization nodes and object detection nodes, and determines the path that the vehicle should take according to the vehicle's internal map. This internal map is a 10cm x 10cm grid representation of the course. Localization data from the IMU and odometry nodes help determine which grid location the vehicle is currently in. Object locations passed from the object detection node are marked on the map, and the updated map is passed through A* planning algorithm to obtain an updated path. When the vehicle reaches a point in the path where it needs to make a turn, the angle at which to turn the vehicle specified by the path is passed to the control node, which will pass this angle to the Arduino to drive the motors.

Initially we planned on having path planning and object detection handled in the same node, however, we decided to separate the two after migrating from a simple angle heuristic path planning algorithm to A*.

3.4 Localization

For our localization system we used odometry performed by optical encoders on each of the four wheels. The IR octocoupler signal for any given wheel is first digested by the Arduino Pro Micro board which sits on the bottom compartment of the vehicle. This Arduino device interprets the signals and sends wheel RPM packets over UART to the Jetson Nano. This result is then then processed by the Jetson Nano and fused with IMU data to produce the localization results.

3.5 Vehicle-to-Vehicle Communication

Since the 'trail' vehicle does not have any sensory input of its surroundings, it relies on the 'lead' vehicle to propagate information of any structural obstacle it detects. Once the object detection node (running on the lead vehicle) detects an obstacle from the Intel RealSense camera, it produces an object message containing the absolute x, y coordinates (in respect to the vehicle's internal map) of the obstacle. It then publishes these coordinates to the server communication node, which are then sent to the trailing vehicle over Bluetooth connection.

Originally, we had designed the communication between vehicles to follow a piconet structure where the lead vehicle would act as the master node which would relay any object information to the trailing vehicles. After realizing we had limited resources for this project, we decided to transition from a 3 car convoy to a 2 car convoy in order to safely budget our remaining resources. This change in our project made it impractical to create a piconet communication protocol across our vehicles (seeing as there are only 2 vehicles). Instead, we maintain a server-client model between the 'lead' vehicle and the 'trail' vehicle (where the lead vehicle acts as the server and the 'trail' vehicle acts as the client).

4 DESIGN TRADE STUDIES

4.1 Vehicle Mechanics

4.1.1 RC Vehicle

One of the most critical components of this development is the vehicle platform on which to build on. Perhaps the more obvious solution would have been retrofitting an existing RC car to fit the requirements of this project; However, as we found out this would prove to be a more challenging solution. An off the shelf RC vehicle would provide us with a reliable platform but we must also acknowledge the setbacks of this option. Namely, it would be difficult to integrate such a system easily with the rest of our components and this would have also eaten up the lion's share of the budget. Traditional RC car systems rely on radio technologies (typically on a 2.4 GHz band). This means that we would have needed to reconfigure the radio system to talk to the compute systems in order to allow for navigation

of the vehicle. Furthermore, it would have been difficult to mount various components such as sensors and cameras on a traditional RC vehicle system. For this reason we opted for an in house solution to provide this requirement. Developing our system from the ground up meant we had control of various parameters such as vehicle speed, size, power draw, weight and interoperability. Indeed, the ability to directly define how the vehicle would communicate with the rest of the vehicle systems was the most convincing advantage that a custom solution provided. Nevertheless, we cannot ignore some of the disadvantages to a custom solution. Most notably the custom solution lead to a loss of development time due to design and fabrication of the vehicles. Above all we recognized the need for flexibility in such a project and this was a solution that provided us with that.

4.1.2 Power Delivery

The initial solution for our power delivery involved the use of 2 separate power supplies for the vehicle system. One 5V supply to power the compute nodes (Nvidia Jetson Nano, Arduino UNO) and one 9V battery supply to power the motors. This solution; however, did not hold up as we expected. The 5V battery supply could not provide a high enough current to run the Jetson Nano camera when we tested it. Furthermore, the 9V battery was had much too low a capacity to drive the 4 motors for a prolonged time period. This lead us to opt for a 7.4V 2S LiPo battery for our power delivery. This battery had a much high charge capacity and also higher current discharge rate. This meant it would be able to provide power to all the systems on the vehicle. This solution did need an additional Universal Battery Eliminator Circuit (UBEC) to work. The UBEC steps down the raw voltage to 5V to power the Jetson Nano. While this solution was more expensive to implement, it was much more reliable and saved some space on the vehicle.

4.1.3 Motor Controls

Initial testing of the vehicle with the L298N Motor drives showed relatively low efficiency performance for controlling all 4 motors. The output voltage that the motors received was much lower than our requirements. Furthermore, this solution was also much more I/O intensive, requiring 12 GPIO pins total for all 4 wheel. The efficiency was especially a concern due to the need to also power the Jetson Nano which draws many amps while performing the object detection. For this reason we switched to the Adafruit Motor Shield V2 for our motor control system. We saw much better power efficiency with this shield and also saved a great deal of wire clutter since the unit mounts on the Arduino. One downside to using this solution was that it was limited to 12V output for the motors. We did not encounter any issues with this as our motors operated on range of 4V to 6V.

4.1.4 Steering System

The first iteration of our vehicle system utilized differential steering with standard wheels for vehicle mobility. When testing this approach, we found difficulty in producing clean turns with the differential steering model. Notably, we saw a lot of drift in the vehicles and also stalling in the wheels. This made it difficult to get reliable data from the odometry sensors as often times the wheels would spin but the vehicle would not actually progress. This model also had a complicated vehicle kinematics which was based around the instantaneous center of rotation [8]. In order to get a much more accurate kinematics model, we would need to either simulate the vehicle physics and run some dynamics testing or run many physical tests. This would give us better parameters for more accurate localization. For these reasons we opted to switch to mecanum omnidirectional wheels to provide mobility. These wheels came with a much more simplified kinematic model [6] and also the more vehicle agility. This model can be seen in Figure 5 below. The wheel diameter did decrease from 65mm to 60mm when we switched to mecanum wheels which meant it was a little less stable on rougher terrains. We also saw that the mecanum wheels did not perform as well as the standard wheels on the track. These issues ended up not giving us too many problems as we opted to switch to an indoor track.

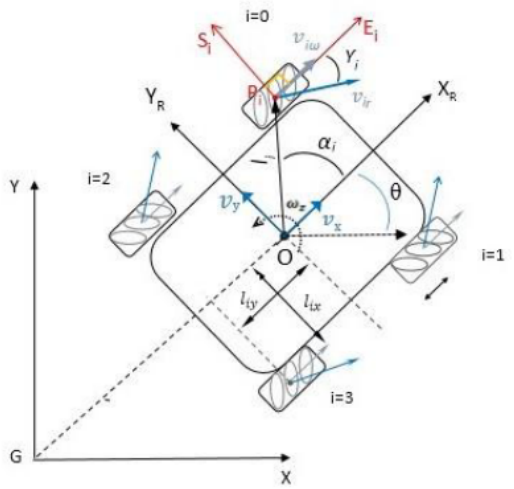


Figure 5: Diagram of Mecanum Wheels Configuration and Posture definition

4.1.5 Vehicle Track

The ultimate goal for this project is to develop a vehicle convoy system that is capable of completing a set course without any collisions. It is therefore crucial for the vehicle track to best reflect the core of this project. Clearly road structures can be much more complex than straight lines and clean angles; However, the focus of the project was less about vehicle mobility and more about communication. In general, a course with curvature would have

required a more robust localization and navigation system. We acknowledge that a global positioning system (GPS) would be useful for such a case; However, at the scale that the project operated such technologies would be too inaccurate (3m tolerance) to pose as a viable solution. Conversely, sticking with straight tracks relaxed these requirements, allowing the group to focus on the communication portion of the project. A straight course is much more predictable and lends to a more simple implementation overall. It is also worth mentioning that a straight course also required much less effort for us to develop as we were able to make use of long hallways for our runs. A more complex solution with turns and other complex road structures would certainly require only more time to develop and test. Above all, the decision was made with the idea of best accommodating the core motivation of the project. While we initially planned to use the university track for our test runs, we saw that the vehicle was much to jittery due to the rough terrain of the track. This lead us to trade off more ideal outdoor lighting conditions for smoother vehicle motion.

4.2 Localization

Our initial design opted for the use of an IMU with open loop odometry to provide localization of the vehicle. With the first few tests of this method, we quickly realized that this would be much to inaccurate for the scale at which we were operating under. The IMU would experience drift even when the unit was entirely still and this would add up over time to produce unusable results. We even found that after 1 minute of sitting still, the IMU would report a displacement of +20cm in the x direction. Even with the use of open loop odometry to estimate locale from expected vehicle behaviors we could not get reliable data. For this reason, we switched to using wheel encoders for our localization solution. We found that these units provided better accuracy and experienced much less drift when reporting location.

4.3 Imaging

For imaging, we considered the following approaches.

One approach we considered was using LiDAR for object detection. This has the advantage of producing point cloud data for the surrounding environment, which can then be passed through an object detection algorithm to exactly outline the shape of the obstacles. LiDAR also has the added advantage of producing information that can easily be converted to 3D maps of the environment and could potentially make object detection and planning easier than with traditional RGB cameras. However, after further consideration, we found that most high quality LiDAR sensors are expensive and beyond the realm of possibility given our current budget. There are small LiDAR sensors within our budget, however, the point cloud information that these sensors provide are not rich enough for object detection algorithms such as VoxelNet to run successfully. Therefore, for these reasons, we decided against using LiDAR for sensing and opted to use a more traditional RGB camera.

We also considered using a regular RGB camera, such as the Logitech HD Pro Webcam. These cameras produce high quality RGB images that are suitable for a wide variety of object detection algorithms such as YOLO V3, VGG16, etc. These cameras are also traditionally cheaper than depth cameras and easier to replace should something go wrong. However, standard RGB cameras don't provide depth information, and it is difficult to infer depth from these images. This would make planning difficult, as there would be no way to infer how far the obstacles are from the car, and hence, how extreme of a turn to make to avoid the detected obstacle. This would complicate the planning process, and make it hard to achieve our requirement of 0 collisions. For this reason, we decided to pursue a stereo camera approach, which not only produces RGB images, but can also use the stereo nature of the camera to determine depth of objects in an image.

4.4 Object Detection

4.4.1 MobileNet v1 and MobileNet v2

We initially planned on using MobileNet v2 instead of MobileNet v1, due to the higher reported accuracy numbers on the COCO dataset. MobileNet v2 also has less parameters making it more ideal for embedded applications. MobileNet v1 has slightly more parameters [1], but is outdated compared to MobileNet v2 [4]. As a result, we planned to use MobileNet v2, as the lower latency makes it more ideal for our application.

4.4.2 April Tagging

We initially considered using April Tagging for object detection. This would involve placing April Tags on the objects, and using the camera and OpenCV to automatically detect the tags. These April Tags are similar in nature to QR codes, however, they encode less information for faster processing [7]. This approach would simplify object detection, since OpenCV would make it easy to create a bounding box around the detected tag. Furthermore, these tags also have the advantage of being unique, and can help with localization, assuming the order of which the tags should appear in the course is known. However, this approach is less realistic and versatile. Since our goal was to simulate true autonomous behaviour as best as possible, we opted to use a more general approach that relies on neural networks.

4.4.3 VGG16 / YOLO v3

In order to achieve the best possible object detection results, we considered using a variety of algorithms. The first class of algorithms we considered were traditional object classification algorithms such as VGG16 [5], and YOLO v3. While these algorithms are very robust and perform well on relevant datasets such as COCO, we were afraid the onboard Jetson Nano would not have the computation power

to process images at a fast enough rate. At the current latency outlined above, an object would have 2 opportunities to be detected otherwise the RC vehicle would likely collide with it. If these algorithms were used, this window would likely reduce to 1 opportunity at best for each obstacle to be detected. Assuming an accuracy of 95%, the probability of collision would be around 5%, compared to 0.25% if a similarly accurate algorithm, but faster algorithm were used. For these reasons, we decided to move to MobileNet v2, a more lightweight algorithm that still provides similar object detection accuracy at a much lower latency.

4.4.4 Faster R-CNN

We also considered using object detection algorithms that are able to factor in depth information. One such algorithm that we initially considered was Faster R-CNN. This algorithm relies on using a region proposal network (RPN), in order to develop candidate regions of which objects can be located [3]. This allows Faster R-CNN to generate accurate 3D bounding boxes around objects. However, Faster R-CNN and other algorithms that rely on RPNs are slower than algorithms based on Single Shot Detectors (SSD) such as MobileNet. Furthermore, the main advantage of Faster R-CNN was that it was able to output 3D bounding boxes around any detected objects. However, for our project, we realized that the depth parameter of the bounding box is not critical to our planning algorithm, and hence, traditional 2D bounding boxes would suffice.

4.5 Communication

4.5.1 Communication Technologies

We utilize Bluetooth as our mode of communication, but another potential option to communicate between vehicles is WiFi Direct. One clear advantage of WiFi is that it has a long range. In a real world scenario with real cars, WiFi's long range makes it realistic for vehicles to communicate with each other.. WiFi also has desired ad hoc behaviour by seamlessly allowing new nodes to send messages in a network (without having to explicitly pair). The main tradeoffs between the two forms of communication were ease of programmability and practicality. Compared to WiFi direct, Bluetooth is more simple to program and setup (especially since WiFi direct is a more proprietary technology). We implemented the communication between our two vehicles using the pybluez library [2] on both vehicle Nvidia Jetsons. Although Bluetooth has less range and lower speeds, it is a reliable form of communication and for the vehicle track, the range of Bluetooth will be more than enough (our vehicles will be less than 5m away at any given time and current Bluetooth technology can easily range up to 50m). Since range is not an issue and our course will be free from large structural obstacles (such as walls or metal objects), we found that Bluetooth was the practical solution for our form of communication.

4.5.2 Communication Protocols

With a Bluetooth communication network of 2 vehicles, a piconet network became a vestigial component of our overall project design. A piconet consists of one master node and up to seven active slave nodes (all in the radio range of the master). All communication occurs between the master node and one slave node, and time division multiplexing is used as the master switches rapidly between slaves. Since we switched from using 2 trailing vehicles in our convoy to using 1 trailing vehicle, there was no longer a necessity to multiplex between different vehicles when sending messages.

Seeing as we now only have communication between 2 vehicles, we updated our communication format to a Server-Client communication model. In our project, the lead vehicle acts as the server; when the lead vehicle launches its server communication node, it traverses through a list of all nearby vehicles until it comes across the target vehicle (the trail vehicle). Afterwards, a connection with the trail vehicle is maintained until the lead vehicle sends a termination message to the trail vehicle. As for the trail vehicle, it listens for a connection on the same port the lead vehicle uses and accepts a connection once the lead vehicle detects the trail vehicle and connects to the trail vehicle. Once the initial connection between vehicles is made, the trail vehicle will constantly listen for any messages from the lead vehicle. The lead vehicle sends map update messages (detailing the x,y coordinates of obstacles) whenever they are detected, thus allowing the trail vehicle to update its internal map and avoid obstacles using its own A* path planning algorithm.

5 SYSTEM DESCRIPTION

5.1 Vehicle Mechanics

To develop our vehicles we opted for a custom designed chassis over the likes of those offered by some third party vendors. This was motivated by the desire for flexibility and exactness to better fit our overall requirements. We needed the vehicle to be able to hold all of our required components, most notably, the Nvidia Jetson and the Stereo camera. This space requirement ultimately drew us away from pre-existing solutions as space would be a premium with such options. In order to accommodate the required components for our vehicle, we designed a full CAD model of the design for the cars. The vehicle chassis was an 11 x 7in design with customized screw holes for each of the required components. This is done to ensure that there are no issues with components falling off the vehicle in the event of collisions. This design was also updated to included battery holders for the LiPo batteries for added safety and convenience. This CAD model is clearly depicted below as Figure 6:

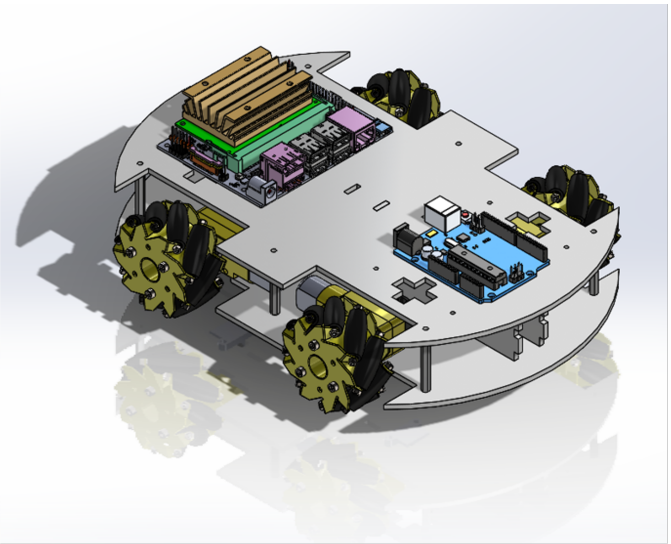


Figure 6: CAD design model for the base vehicle system

As is visible from the CAD model, the vehicle is driven by 4 DC motors. These motors are controlled using the Adafruit Motor Shield and powered by a the 7.4V LiPo battery. The Arduino UNO unit served as the controller for the vehicle steering system. For our implementation the Arduino received commands serially from the Jetson Nano via the shared USB connection. The Jetson was responsible for providing a desired angular velocity for each wheel separately. Since we did not use Electronic Speed Control Circuits (ESCs) for the motors, we used empirical results to attempt to map the command velocity to the actual velocity the wheels achieved. We found that these parameters were most accurate around an 8V charge on the LiPo batteries.

5.2 Camera

After researching potential stereo cameras that would suit our requirements, we decided to use an Intel RealSense Camera that we already owned. Not only would this relieve pressure on our budget, but it is also easily integrated with ROS, Python, and OpenCV. The camera also has a horizontal field of view of 86 degrees and a frame rate of up to 90fps, both of which exceed the necessary capabilities for this project. This camera also has the added advantage of being familiar to us, as most of us have a bit of experience integrating the camera with Python in the past. We also engineered the body of our RC vehicles to ensure the camera would fit on the car. Interfacing between the Intel RealSense and the Jetson is well documented and fairly straightforward to implement.

5.3 Localization

Our localization system consisted of a combination of wheel odometry and IMU data. For the wheel odometry we used IR Optical Octocouplers to obtain RPM for each wheel. This was done by using Interrupt Service Routines (ISRs) on the Arduino Pro Micro. In particular, the interrupts for each wheel are triggered by the falling edge of the optical encoder signal. The RPMs for each wheel are continuously updated by measuring the time between interruptions while factoring how many slots made up the entire wheel encoder. At a fixed rate the Pro Micro device will report the RPM data as a structured packet over UART to the Jetson Nano. This packet reports only the vehicle RPM and not the direction of the wheels since the encoders are one-way. Therefore we added the expected direction of the wheels by assuming that the control signal and physical system were in alignment. We found that this technique worked well despite possible errors due to switching delay in control versus physical. The RPM data is then fed into the kinematic model of the vehicle to derive longitudinal and lateral velocity. These velocities are then used to provide x and y displacements by performing discrete integration. Notably, we consider velocity to be constant at zero for this calculation. It is worth noting that the previous method for localization included the use of an IMU to calculate the vehicle displacements. The IMU we used for this task was a BNO055 Adafruit 9DoF board. With some testing we soon realized that the IMUs were too inaccurate for our use case. This is mostly due to the scale of the vehicle as relates to the amount of drift in the sensor. For this reason, we opted to only use the IMU for providing feedback to the control system. The reported yaw of the vehicle from the IMU was used to check whether the vehicle was moving the in desired direction to some tolerance.

5.4 Object Detection

Object detection in the lead vehicle relied on an Intel RealSense Depth Camera. This camera provides RGB-D information to the object detection node. The RGB image is passed unfiltered through MobileNet v1 SSD to extract bounding boxes around any obstacles detected. In order to achieve optimal performance, MobileNet v1 was first converted to a TensorRT format that is more performant on embedded computing devices such as the Jetson Nano. Initially we intended to use MobileNet v2, however, after initial testing, we found that the smaller MobileNet v2 network had greater difficulty identifying small objects consistently compared with MobileNet v1. Once the bounding boxes are extracted, the RGB image and the depth image are pixel-aligned in order to accurately extract the depth at the center of the detected obstacle, and extrapolate the obstacle's location in the vehicle's reference frame. This location is then passed to the path planning node to update the vehicle's path.

In the event that the depth image outputs an unknown depth (value of 0), the last seen valid depth value is used in its place in order to calculate an approximate for the detected object's location. We found that in less than optimal lighting conditions the unknown depth issue was more common.

5.5 Path Planning

Once an obstacle is detected, a new path must be computed in case the vehicle's current path collides with the obstacle. Localization data is used to convert the detected obstacle's location in the vehicle's reference frame to the global reference frame. Once the global location of the obstacle is known, it's corresponding location in the vehicle's internal map is marked, along grid all grid locations within a 5×5 square centered around the obstacle's location is marked as well. Once the internal map is updated, it is then passed through A* planning algorithm to obtain the updated path. An example of this path can be seen in Figure 7 below, where the obstacles are marked in red, and the new computed path is marked in green. As the vehicle approaches the detected obstacle, the weights for the corresponding obstacle locations increase, forcing A* to find a path that avoids any grid location close to the obstacle.

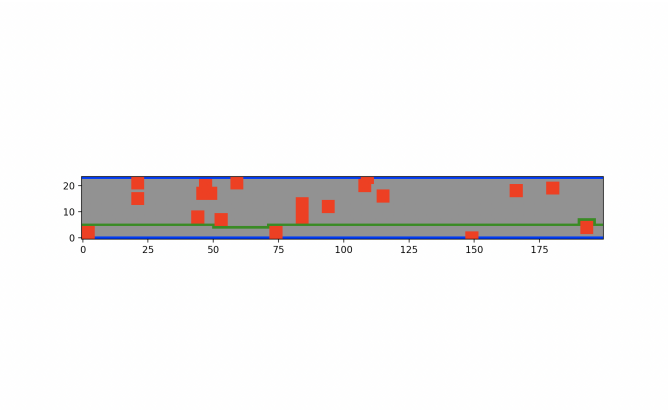


Figure 7: Example Path Planning Map results

In order to keep our planning algorithm more flexible, control signals are sent to the control node only when the vehicle must make a turn. When this occurs, the angle at which to turn the vehicle specified by the current path is passed to the control node, which will communicate with the Arduino and Adafruit motor shield to drive the vehicle.

Path planning for the following vehicle works in a similar way. However, instead of having the object detection node pass local locations of obstacle coordinates, the following vehicle receives obstacle coordinates in the global reference frame over Bluetooth. The following vehicle is then able to mark the same 5×5 grid in its internal map as the lead vehicle, and run A* to update its path in the same way as the lead vehicle.

Initially, we wanted to pursue a more simple path planning algorithm to offset the anticipated object detection latency. Our original algorithm simply calculated an angle at which to move the vehicle in order to avoid the closest obstacle, but did not maintain an internal map or goal. We felt that this approach was unrealistic and could potentially have a lot of variance depending on the placement of the obstacles, and would also make it difficult for the following vehicle to replicate the path. For these reasons, we decided to use a more robust solution, which was made possible by the low latency of our object detection stack.

6 TEST & VALIDATION

In this section, include tests that you used to evaluate your design implementation and compare to the theoretical design trade-offs. Do your measurements match the theoretical predictions?

You should partition with the subsection format to group your discussion in terms of results to validate qualitative and quantitative product requirements and/or design specifications. Include test results for design specifications that help further validate the design performance.

6.1 Results for Object Detection

To test the object detection capabilities of the system, we first compared both MobileNet v1 and MobileNet v2 and their respective performance on a small dataset of 20 images of bottles at a range of 0.2m to 1.2m away from the camera. The results can be summarized below. Note that we only count an object as detected if the object was able to maintain being detected for at least 2 seconds.

MobileNet v1	MobileNet v2
95%	75%

As a result, of we decided to implement Mobilenet v1 instead of MobileNet v2.

When testing object detection capabilities of the vehicle, we collected a dataset of 50 images by placing bottles at various orientations around a static vehicle, with the camera mounted, at a distance of 0.4m. We used 0.4m as the baseline distance, since it is the minimum distance at which an obstacle must be detected in order to successfully avoid it when the obstacle is moving. We recorded precision and recall on this data set. The results are summarized below. We also experimented with using wooden blocks as another class of obstacles. The results can also be seen in the table below.

Object Type	Precision	Recall
Bottles	92%	98%
Wooden Blocks	92%	60%

The performance with wooden blocks had similar accuracy, however, there were significant false negatives, leading to a lower recall score. For this reason we decided to only use bottles as the obstacles in our course to mitigate the chances of obstacles on the course not being detected.

6.2 Path Planning

As a result of transitioning from the angle heuristic algorithm, we didn't develop any tests for simulation, since A* is a provable algorithm that finds a path from a start state to a goal state while actively avoiding marked locations. We simply tested the robustness of the algorithm on

our map by generating a path for a random sequence of obstacles and visually seeing if the path ever intersected with marked obstacles. An example of this is seen in section 5.4. With 10 random locations of 20 obstacles , A* was indeed able to determine a path that avoided the obstacles and reached the goal position.

Obstacle Number	Success Rate
1	5/5
2	4/5
3	4/5
4	3/5
5	3/5

As the number of obstacles increased, the vehicle had more collisions with obstacles. We reasoned that this was not a fault with the path planning, but rather an issue with the drift of the vehicles. Every turn that the vehicle made to avoid an obstacle, would cause the orientation to change a little bit leading to non-negligible drift in the vehicle's movement, meaning it failed to follow the path specified by A*.

6.3 Bluetooth Communication

The trail vehicle dependents greatly on the Bluetooth connection which is formed with the lead vehicle. Therefore, we had to ensure that the connection between the two vehicles was **reliable** and **fast** in order for the trail vehicle to receive accurate information about the detected obstacles the lead vehicle encounters.

In the spectrum of reliability, we found our early tests of the Bluetooth connection showed that for our purposes, the connection will always be reliable. The first preliminary test we performed was on the integrity of an formed connection between the two vehicles (with the goal of discovering how likely the connection was to break once already formed); we connected both vehicles at a close range of 0.5m and separated both vehicles and found that the connection was broken only after a distance greater than 20m. This made us feel safe about the connection integrity since our course is 5m long and both vehicles will always be within 0.5m of each other.

Next, we wanted to test how easily the vehicles could connect to each other. Since both of our vehicles start off statically at the beginning of the course, we just needed to ensure that after a certain time, they would connect to each other. We tested the starting points at different distances and found that the vehicles always connected to each other after about 12 -16 seconds, and the distance did not affect the connection time. Although an interesting finding we came across was that the connection was not formed at a distance of 5m if there was a large structure (such as a wall) in between the two vehicles. Thus, we were confident that a connection in an open course will always be formed within a range of 1m between both vehicles.

Finally, we wanted to ensure that the delivery of messages between the vehicles was fast. In our original design report we were anticipating a latency of about 140ms but after performing timing tests of delivering messages, we found that the rate of sending messages was about 12.5 messages per second or a communication latency of 80ms as we see in Figure 8. We recorded the time it took 100 messages to reach the trail vehicle (client) at different lengths ranging from [0.5m, 6.5m] and found a negligible drop in message rate when increasing the distance. We hypothesize that the message rate would drop more drastically given distances larger than 10m and objects obstructing the vehicles, but such scenarios are out of scope for our project. Our track only covers 5m, so the rate of message delivery will reliably be around 12.5 messages/s.

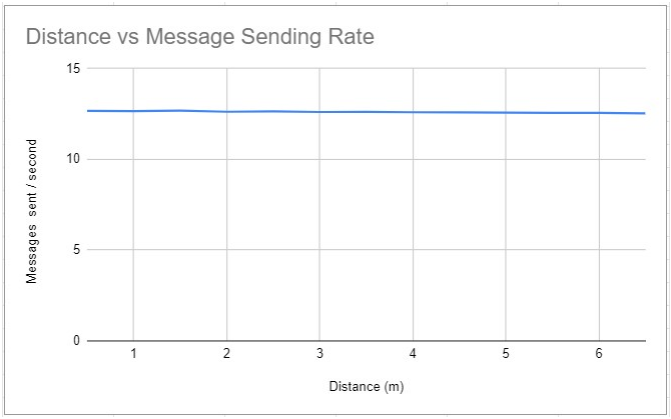


Figure 8: Effect of Distance on Message Sending Rate

7 PROJECT MANAGEMENT

7.1 Schedule

Our full schedule can be seen in the Appendix in Figures 10 and 11 in Appendix B. We decided to work on the separate subsystems in parallel, and integrate the entire systme during the last two weeks. We were on schedule for majority of the project, but ran into some issues during the integration phase, where integration took longer than expected. Part of the reason was porting some code written locally onto the Jetson Nano was an adjustment, and also converting all the systems to ROS in order to communicate with one another also took some tinkering to get working. In the end, integration took about 2-3 weeks in total, and we had to work longer to get it to work properly.

7.2 Team Member Responsibilities

Joel was in charge of vehicle mechanics, including the assembly, and design of both vehicles. He also developed the systems for localization and odometry that were integral to the project. Joel also worked with Jeffrey on developing the path planning algorithm, and Fausto on developing the communication framework.

Jeffrey was in charge of object detection and path planning. He tested and developed the system that determined

local locations of detected objects. He also worked closely with Joel to tune and create the path planning algorithm.

Fausto developed and tested the communication framework and integrated Bluetooth with the system. He also worked with Joel and Jeffrey to determine what should be sent to make planning for the following vehicle the easiest.

The whole team worked together in lab and integrated the subsystems together. We also all worked together to debug any integration bugs and test the system as a whole.

7.3 Budget

The overall budget for the project is broken down in Table 2 in Appendix C. Here we have categorized the parts as follows. Parts that were purchased and unused in the final solution are in yellow, parts that were purchased and used in the final solution are in green while the parts already owned or borrowed are in blue. For each category we also provide the total cost for each. This cost is broken down similarly as seen in Table 3 in Appendix C. Note that the overall cost for the entire project came to $960.60 total.

7.4 Risk Management

The primary risk came from the design. Our original design didn't account for the possibility that the vehicle would drift significantly. As a result, our initial solution to localization involving IMU data and open-loop feedback proved insufficient. As a result, we transitioned to using Optical optocouplers as wheel encoders to estimate the wheel RPM and extract more accurate localization data with an updated vehicle model. After making this change, and implementing a simple PID controller for the wheel velocities, we were able to create somewhat accurate localization data for the vehicle. We also integrated other solutions to mitigate drift in localization data, including a ramp up period that ramps the wheel RPMs to the desired results to avoid spinouts and slippage in the wheels. However, persistent mechanical issues and an inaccurate vehicle model limited our capabilities to mitigate vehicle drift. This problem affected our scheduling as well, as we did not expect the vehicle mechanics to account for a majority of the work in the project.

In our initial review of our design, we feared that the angle heuristic algorithm would be difficult to implement, as the object detection could not update the plan fast enough to comfortably allow the vehicle to avoid the detected obstacle. As a result, we opted to transition to Mecanum wheels for increased agility, as well as better grip on the track when compared with standard wheels.

Another risk was moving from MobileNet v2 to MobileNet v1, as a result of the former's inability to consistently detect smaller objects. This wasn't a huge change, although it did mean we needed to research into how to port larger networks on to the Jetson Nano, since MobileNet v1 is larger than MobileNet v2. We opted not to move to our fallback solution of using April Tags, as we felt like the object detection using MobileNet v1 in TensorRT form provided along with the depth image from the Intel RealSense camera supplied accurate enough results that we didn't need to move to an alternative solution.

Finally, the main challenge in our project was not the individual subsystems, but integration. We initially thought that the integration of the subsystems would be relatively quick, however, majority of the time was spent bug fixing how the ROS nodes were interacting with each other. Overall, we needed to allot more time to vehicle drift in order to achieve better results.

7.

8 ETHICAL ISSUES

The goal of this project was to demonstrate the effectiveness of vehicle to vehicle communication for autonomous navigation purposes. Although this technology could potentially change the landscape of transportation, there are possible ethical issues that may arise.

First and foremost, one of the most significant challenges with this product is security of the network. Any network that the vehicles must communicate to-and-from must be secure, otherwise malicious users will be able to send hostile messages to control and manipulate the vehicles. Approaches to mitigate this would be to develop a custom communication protocol to make the network more secure, however, this introduces overheads into the communication which may delay time sensitive decisions that the cars will make.

Furthermore, autonomous navigation also has the possibility of introducing bias into object detection. The data sets that MobileNet v1 and other object detection algorithms might be inherently biased to certain objects that are common in some areas and not in others. This would make object detection biased to some neighborhoods and not others which will impact performance and accessibility. There are well documented cases in literature where facial recognition algorithms are biased to lighter skin individuals resulting from an under-representation of darker skinned individuals. In order to correct for this possibility, the data sets that the object detection algorithm is trained on should be carefully scrutinized for potential sources of bias.

We also would like to note that V2V and autonomous navigation in general is still in its infant stages in development. There are many barriers to making it widespread, and thus, potential for other ethical issues to arise. However, as this technology continues to develop, engineers must pay attention to the accessibility of this technology to different demographics, and ensure it doesn't discriminate others.

9 RELATED WORK

There are a lot of projects online that deal with autonomous navigation of RC vehicles. However, most of

these projects involve a single car navigating through a course, rather than a convoy. One notable project was one produced by students at WPI also created a similar project that involved autonomous navigation of RC vehicles. However, unlike our project they relied on LiDAR and other forms of odometry using EKF to perfect localization. Our project not only performs object detection differently, but also attempts to navigate a convoy of vehicle, rather than an isolated RC vehicle. Other projects have also used similar object detection approaches, but all rely on more complex path planning approaches, rather than a simple bounding box approach we decided to employ.

10 SUMMARY

Our project sought to demonstrate the effectiveness of V2V by implementing an autonomous vehicle convoy. We were able to develop a two car system to navigate a short track with obstacles, where only the lead vehicle has perception capabilities. Due to limitations with vehicle drift, we had to loosen our requirements and reduce both the track distance and the number of obstacles. Given more time, we would have liked to introduce a third car to the convoy that the lead car would communicate with, as well as implement better control algorithms to help mitigate vehicle drift.

10.1 Lessons Learned

There were many challenges that we faced in the development of this project. The first was expecting our vehicle to behave as we expected it on paper. We expected by driving the motors with the same voltage, the vehicle motors should move in the same way making the vehicle go straight. However, we found out this is an unrealistic model to have, as every wire and motor has slightly different electrical characteristics making all of the hardware components behave slightly differently, making it extremely hard for the vehicle to go straight. When dealing with hardware, we learned that we needed to spend more time developing control algorithms to help mitigate these manufacturing differences.

We also learned a lot about integrating code with the Jetson Nano. Neural networks off the shelf can't directly and efficiently integrate with the Jetson Nano due to the large graph size. In order to make networks more performant, we had to convert our neural network to TensorRT format.

Glossary of Acronyms

Include an alphabetized list of acronyms if you have lots of these included in your document. Otherwise define the acronyms inline.

- IMU - Inertial Measurement Unit
- SSD - Single Shot Detection
- UBEC – Universal Battery Eliminator Circuit
- V2V - Vehicle to Vehicle
- YOLO - You Only Look Once

References

[1] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: (2017). arXiv: 1704.04861 [cs.CV].

[2] Albert Huang and Larry Rudolph. "Bluetooth for Programmers". In: (), p. 77.

[3] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].

[4] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: (2019). arXiv: 1801.04381 [cs.CV].

[5] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: (2015). arXiv: 1409.1556 [cs.CV].

[6] Hamid Taheri, Bing Qiao, and Nurallah Ghaeminezhad. "Kinematic Model of a Four Mecanum Wheeled Mobile Robot". In: *International Journal of Computer Applications* 113.3 (Mar. 18, 2015), pp. 6–9. ISSN: 09758887. DOI: 10.5120/19804-1586. URL: http://research.ijcaonline.org/volume113/number3/pxc3901586.pdf (visited on 05/14/2021).

[7] John Wang and Edwin Olson. "AprilTag 2: Efficient and robust fiducial detection". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016.

[8] X. Wu, M. Xu, and L. Wang. "Differential speed steering control for four-wheel independent driving electric vehicle". In: (2013), pp. 1–6. DOI: 10.1109/ISIE.2013.6563667.

11 Appendix

11.1 Appendix A

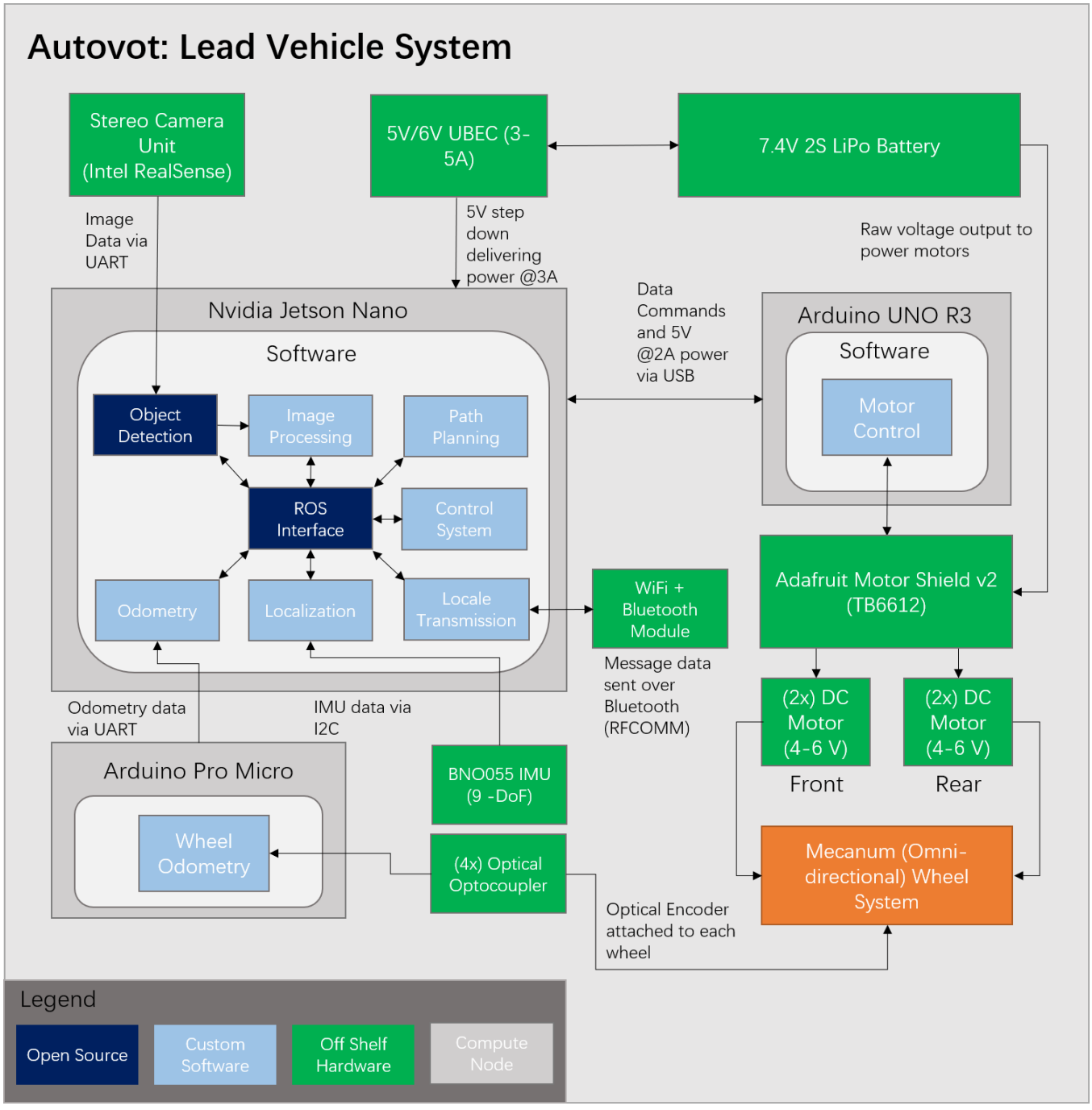
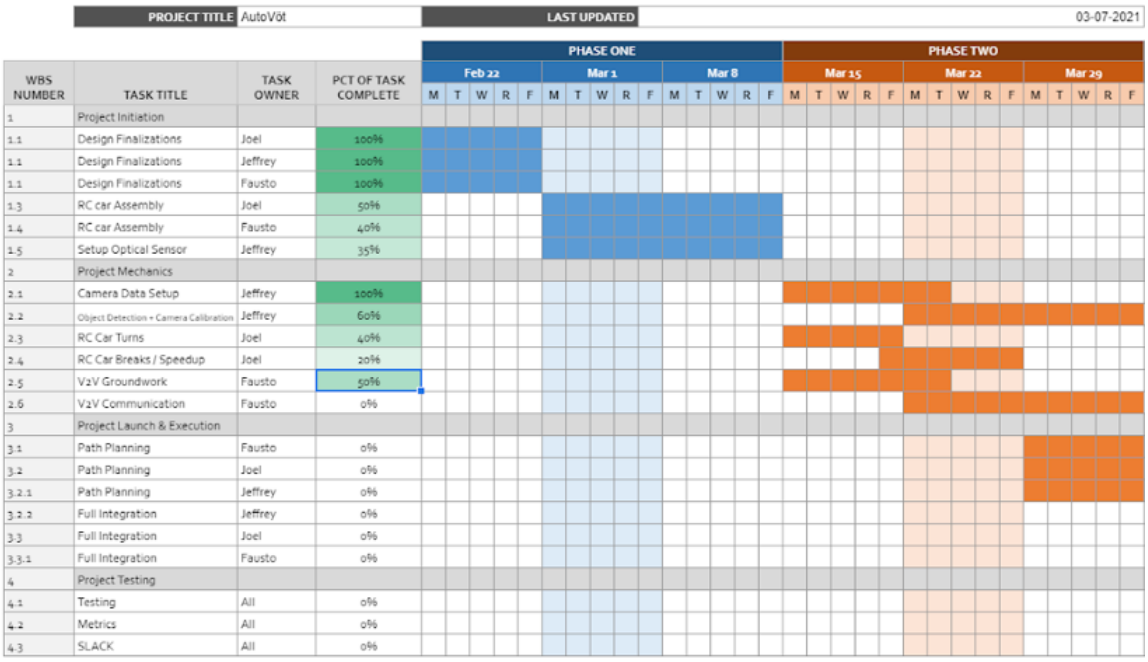


Figure 9: Architecture Diagram for the lead vehicles

Appendix B



PROJECT TITLE	AutoVöt	LAST UPDATED	03-07-2021

WBS NUMBER	TASK TITLE	TASK OWNER	PCT OF TASK COMPLETE
1	Project Initiation		
1.1	Design Finalizations	Joel	100%
1.1	Design Finalizations	Jeffrey	100%
1.1	Design Finalizations	Fausto	100%
1.3	RC car Assembly	Joel	50%
1.4	RC car Assembly	Fausto	40%
1.5	Setup Optical Sensor	Jeffrey	35%
2	Project Mechanics		
2.1	Camera Data Setup	Jeffrey	100%
2.2	Object Detection + Camera Calibration	Jeffrey	60%
2.3	RC Car Turns	Joel	40%
2.4	RC Car Breaks / Speedup	Joel	20%
2.5	V2V Groundwork	Fausto	50%
2.6	V2V Communication	Fausto	0%
3	Project Launch & Execution		
3.1	Path Planning	Fausto	0%
3.2	Path Planning	Joel	0%
3.2.1	Path Planning	Jeffrey	0%
3.2.2	Full Integration	Jeffrey	0%
3.3	Full Integration	Joel	0%
3.3.1	Full Integration	Fausto	0%
4	Project Testing		
4.1	Testing	All	0%
4.2	Metrics	All	0%
4.3	SLACK	All	0%

Figure 10: Gantt Chart Part 1

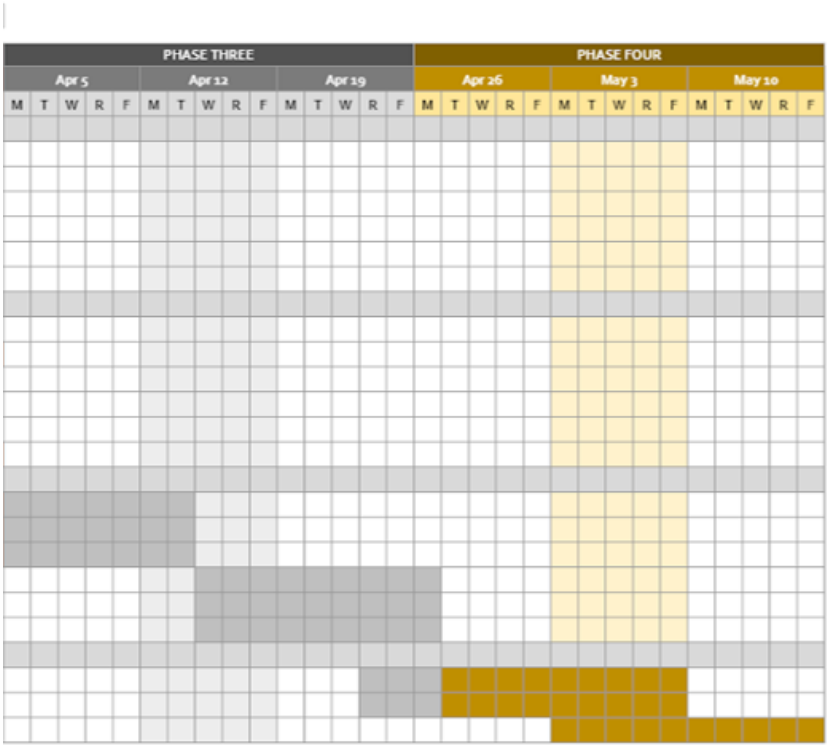


Figure 11: Gantt Chart Part 2

Appendix C

Part	Price	Qty	Subtotal
Wheel + DC Motor Kit (x4)	16.99	3	50.97
9v Power Supply Holder	8.98	1	8.98
5v Power Supply	15.95	3	47.85
Jeston Nano (2GB)	59.00	2	118.00
Micro SD Card 32 GB (x3)	15.99	1	15.99
USB to Arduino Cable (x5)	13.27	1	13.27
9-DOF Absolute Orientation IMU Fusion Breakout	38.95	3	116.85
Metric Steel Pan Head Screws M3 (x100)	2.96	1	2.96
Steel Hex Nut M3 (x100)	0.88	1	0.88
Female Hex Standoff M3	1.17	20	23.40
Steel Pan Head Phillips Screw, M3 x 0.5 mm Thread, 30 mm Long	4.66	1	4.66
Adafruit Motor Shield V2	24.14	2	48.28
2S 7.4v LiPo Batteries (x2)	29.99	2	59.98
5v/6v UBEC 3A/5A Max	9.90	2	19.80
Mecanum Omni Directional Wheels	21.99	2	43.98
Wheel Encoders + IR Optocouplers	6.79	2	13.58
12 x 24 Acryllic Sheet	7.50	1	7.50
12 x 24 Wooden Sheet	3	4	12
Nvidia Jetson Nano (4GB)	99.00	1	99.00
Arduino Pro Mirco	10.00	2	20.00
Arduino Uno R3	21.49	2	42.98
Wireless NIC Module for Jetson Nano	26.99	1	26.99
Intel RealSense D415	160	1.00	160
Mini Breadboard	1.50	2	3.00
Miscellaneous wires, screws etc.	10	1	10

Table 2: Table with full budget for project

Categories	Total
Purchased (Unused)	107.80
Purchesed (Final Design)	501.13
Owned/Borrowed/Scrounged	351.97
Overall Cost	960.90

Table 3: Table summarizing budget by category of item