MIAMI UNIVERSITY

Self-Driving Autonomous Car for the 2025 American Control Conference QCar Student Competition

Spring 2025

College of Engineering and Computing

Department of Electrical and Computer Engineering

ECE 449: Senior Design Project, Spring Final Report

Student - Zach Copenhaver

Student - Jakob Felts

Student - Fred Levins

Student - Josh Strong

Advisor - David Hartup, Ph.D.

Advisor– Bryan Van Scoy, Ph.D.

5/9/25

Table of Contents

Introduction

The goal of this project is to design a self-driving car that can operate as a self-sufficient taxi service, navigating across a designated track while planning and following a path in a timely and performance-efficient manner. As traditionally defined, a vehicle that is labeled as a 'self-driving' vehicle is one that will be fully autonomous and operational without a human operator by using multiple sensors, cameras, coding, and possibly Artificial Intelligence (AI) (Gillis & Lutkevich, 2024). The car must be able to autonomously drive both safely and efficiently around the track, following traffic safety regulations and coordination as would a manually-driven car. To satisfy this goal, the group must be able to demonstrate and apply autonomous driving fundamentals, as well as gain knowledge and understanding of how the car works to fully program it to follow the rules of the road.

Project Background

Since the 1930's, the concept of a self-driving vehicle has piqued interest across both engineers and drivers alike. Whether it may be a plane or a boat, the key idea of a self-driving transport is to provide an autonomous form of transportation that can avoid hazards, streamline maneuverability, and above all, efficiently travel to a destination without the need for human input. While many automotive companies and research groups have implemented functions such as collision prediction and lane control in commercial vehicles, a fully autonomous vehicle was the main goal amongst researchers.

To promote research and interest in self-driving vehicles, robotics and research company Quanser developed the QCar around 2020, an "open-architecture, scaled model vehicle" used as a learning and research tool to help understand how a self-driving car would work on a smaller

scale. Quanser's mission statement is to "develop engineering platforms that accelerate research and transform educational experiences". They are, in general, an educational product development company focused on producing STEM based tools and inventions to promote student knowledge growth and development. To fulfill this educational intent, the QCar is equipped with various components to allow for user flexibility and testing. As illustrated in Figure 1, the car is powered by an NVIDIA Jetson TX2 that is equipped with a wide range of sensors including LIDAR, 360-degree vision, depth sensor, encoders, and user-expandable IO.

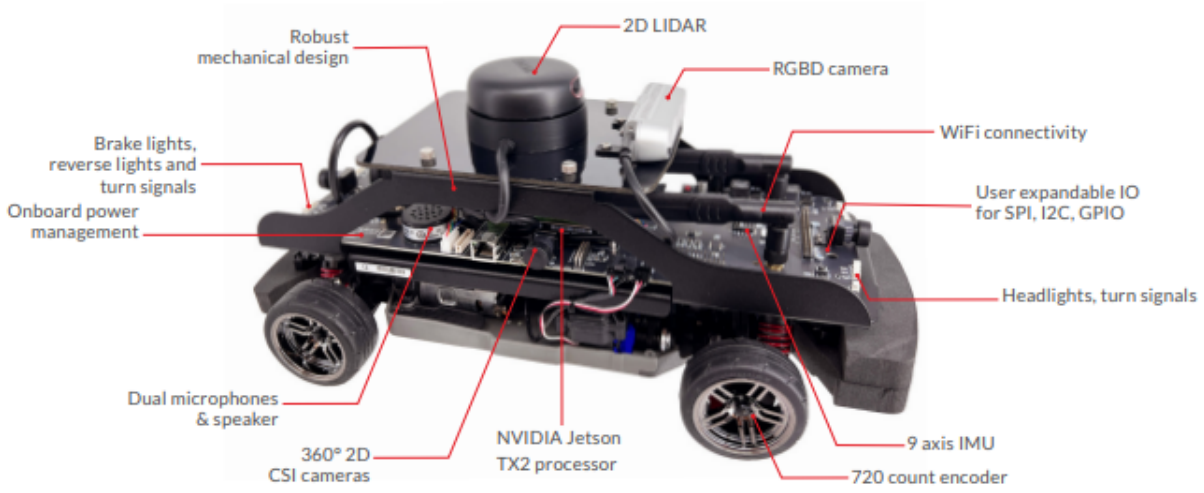


Figure 1. Quanser QCar Product Details (Quanser, 2024)

As a scale-model vehicle, the QCar has been a key area of focus for autonomous performance and testing for many researchers due to its ability to simulate a real-world self-driving environment. Using "...aspects of self-localization, a centralized infrastructure, artificial intelligence for sign recognition and the use of sounds as audible cues" (Pineros), the QCar sparked the interest of the committee for the American Controls Competition, and since 2023, the car has been used as the 'feature vehicle' for the competition's Self-Driving Car Studio due to its interactive approach to autonomous driving.

When considering the inherent risks and implementation of self-driving cars, the importance of object detection is historically noted and completely essential. Due to the importance of real time object detection for various external variables such as stoplights, pedestrians, stop signs, and other traffic aspects; we look to the historic You Only Look Once (YOLO) Algorithm. The YOLO (You Only Look Once: Unified, Real-Time Object Detection, Redmon at. 2015) paper marks the beginning of a groundbreaking approach to object detection by introducing a unified model that treats detection as a single regression problem with one pass through. Unlike traditional methods that rely on region proposal networks and multiple processing stages such as Region-based Convolutional Neural Network (R-CNN), YOLO simplifies detection into a single neural network pass, dramatically improving speed and efficiency. The model's ability to process images in real-time without sacrificing much accuracy due to thresholding made it revolutionary, setting the stage for advancements in real-time object detection algorithms. This work laid the foundation for the YOLO lineage, inspiring numerous improvements and iterations in the field to the current day of verison nine. Due to limitations of the vehicle and python version restrictions, we focus on YOLO version three.

The American Control Conference (ACC) and Quanser

Figure 2. ACC 2025 Denver Logo (https://acc2025.a2c2.org/, 2024)

The American Control Conference (ACC) is an annual event focused on the presentation of advances in control systems theory, applications, education, and overall automation advances. The attendees include researchers, educators, and industry professionals from all over the world looking to gather and gain knowledge from one another. The conference venue for 2025 is Denver, Colorado, known for its technological innovations and academically rich community, providing a strong opportunity to promote and establish collaboration and growth.

The ACC is a platform for presenting state-of-the-art research and finding emerging technologies in areas such as robotics, automation, autonomous systems, and artificial intelligence. The conference normally takes several days with keynote addresses by prominent experts, technical sessions, tutorials, and panel discussions. Participants go into deep discussions on everything from fundamental control theory to implementation in industrial and societal contexts. Overall, it is a multi-event day that works to fit all automation industry professional, student, and hobbyist interests and skills alike.

For students, the ACC offers unrivaled opportunities to learn from and interact with leaders in the field. Dedicated student sessions and competitions create an engaging environment and a strong opportunity for presenting individual work and to compete in events. One of the events at ACC 2025 is the Quanser Self-Driving Car Student Competition, where student teams compete to design and implement their own solutions to a given problem using the QCar.

For the 2025 competition, the car will act as an autonomous taxi service, navigating a designated path to request ‘pick-up’ and ‘drop-off’ locations to follow along. The car will encounter various traffic scenarios, and is expected to follow the rules and regulations of the road through autonomous movement. The competition focuses on practical learning such that students have to design, implement, and test control systems for a scaled self-driving QCar. The

application based aspect of the competition allows for bridging theoretical knowledge and practical application, catering to and promoting the skills of the students. It is open-ended with individual team choices taking prevalence depending on output. There is a tier system to complete to be able to compete in the final in person event.

Participants are judged on various aspects, but with an overall focus on time and efficiency. The overall objective is to have the car demonstrate self-driving, path-planning, and traffic fundamentals while navigating the track, moving between the ‘pick-up’ and ‘drop-off’ points in a timely and performance-efficient manner, with increasing penalties for incorrect or illegal movements. Upon completing the tiers of submissions, the teams will compete to show how and why their development strategy was strongest. The competition does not stop at technical skill but extends to teamwork, problem-solving, and the ability to effectively communicate complex ideas which incorporates the importance of our participation and overall capstone experience.

As our spring semester began and we received more information on the time-line of the competition, we quickly realized that the timeframe of the competition does not kindly line up with the academic timeline. If we wished to fully partake in the competition, we could possibly be working on it through June and early July, which is two months past graduation/semester end. While we still had intent on doing some participation with the competition, we started off running into many issues as the first stage is fully simulation based and our simulation wouldn’t properly operate, even with the support from Quanser. Because of these limitations, we have decided to focus on the physical aspects that we currently have to emulate the final goal of the competition’s taxi-service rather than the virtual simulation aspects.

Summary

In the fall semester, our group originally intended to compete and participate in the 2025 American Control Conference QCar Student Competition. In preparation for the 2025 competition, our group has acquired a QCar from Quanser to implement and demonstrate self-driving functionality for the competition. However, due to time constraints and lack of flexibility nearing the end of our undergraduate studies, our group focused on a more independent method of research instead of the competition. While we are still working off of the competition's goal of designing the car to act as a self-driving taxi service, it was in our best interest to focus on this research without the competitive nature of the event. However, our group intends to demonstrate the research and our final design at Miami University's 2025 Senior Design Expo near the end of the spring semester, where other students' research and projects will be presented in a formal setting. In this report, our research, solutions, and findings are presented, as well as our final product using the work compiled over the last two semesters.

Project Research

In the fall semester, the team initially had planned to divide this project into two groups, Image Processing and Controls. The way this was planned to work was that the Image Processing group would work on understanding how to effectively retrieve and use what the car is seeing, while the Controls group would focus on understanding how the car's operations work to effectively move on the track. As we progressed through the spring semester, with a solid idea of the end goal in mind, both efforts were combined to accomplish the goals of having the vehicle driving autonomously. Each group went into various routes of research described below.

Image Processing

The car has two types of cameras that we used in our programming. The first type of camera was an 8 Megapixel 2D Camera Serial Interface (CSI) Camera, which can be seen below in Figure 3.

Figure 3. CSI Camera (Quanser, 2024)

The car has four CSI cameras with one placed at the front, left, right, and rear. These cameras have wide angle lenses providing up to a 120° vertical field of view (FOV) and a horizontal FOV of up to 160°. The placement of the four CSI cameras allows the car to have a 360° horizontal FOV with only small blind spots close to the car itself. This can be seen below in Figure 4.

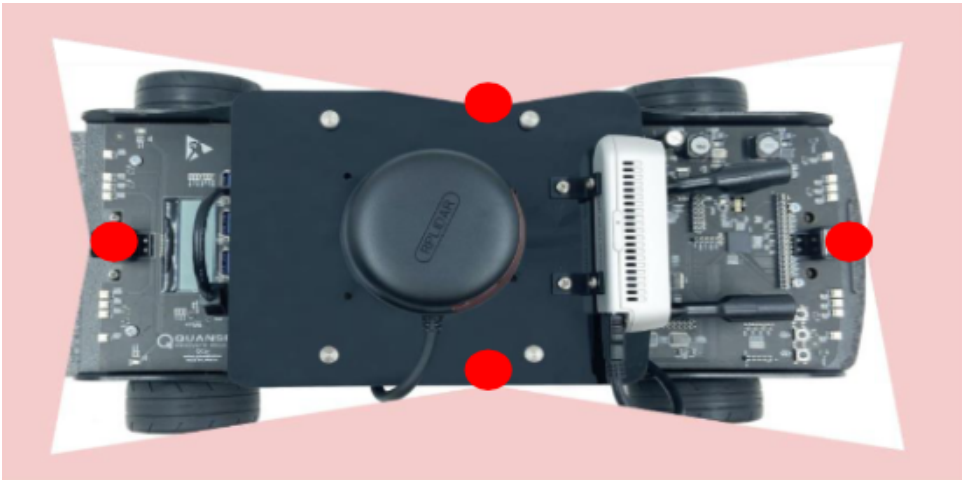


Figure 4. Top-Down View of QCar. Red Dots: CSI Camera Locations. Pink: FOV.

These cameras view images in the red/blue/green (RBG) color space which can later be converted into other color formatting for further image processing. The frame rate of each camera varies depending on its resolution and can range between 20 Hertz (Hz) and 120 Hz. The frame rate decreases as the resolution increases and more pixels need to be processed each frame. The cameras can show up to around 8 million pixels per frame with a resolution around 3280 x 2464 pixels.

We mainly used a Python library called OpenCV for our image processing. OpenCV is an open source computer vision library that provides a large set of optimized image processing and object detection functions and algorithms. We used this library for several applications such as color conversion, creating masks, displaying the camera feed, and detecting objects in the frame.

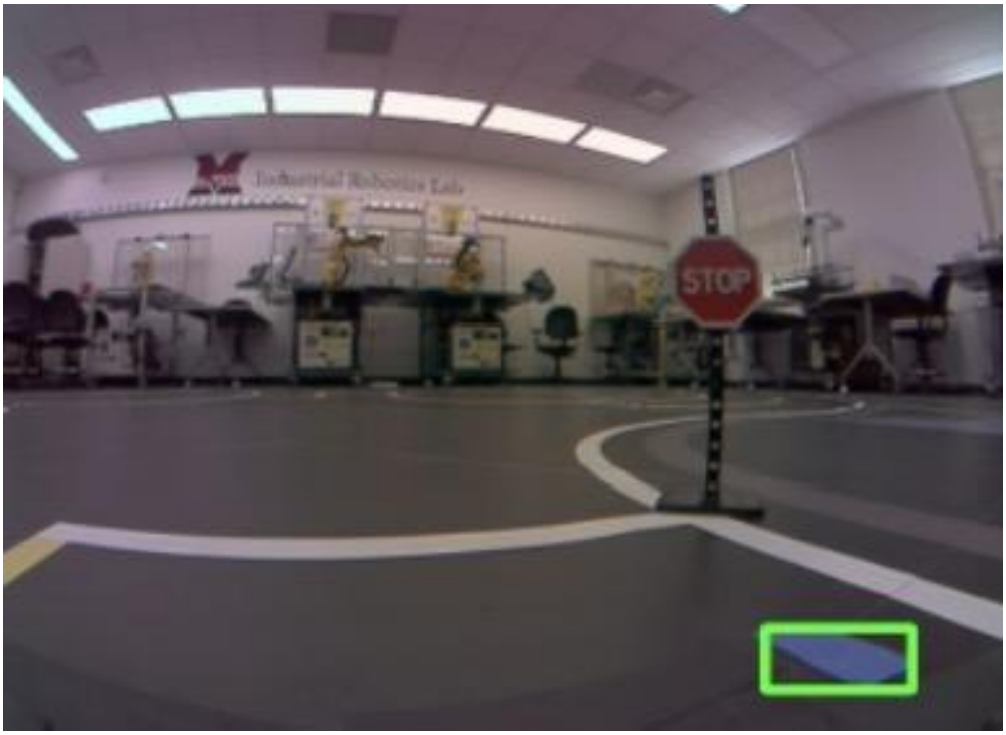


Figure 5: Real life node detection from image feed of car showing blue masking to localize the car

In order to help us with localizing the car, we split the track into 19 different nodes. We then placed each of these nodes into a directed and weighted graph that simulated the track itself.

This graph was used to develop path planning algorithms as well as a finite state machine for the control algorithm as explained later in this paper. The edges were defined as the path between two nodes on the road and were used as the states in our finite state machine. The nodes were placed such that each edge would only include one type of motion, i.e. the car would never have to turn right and then left within the same edge. We placed blue tape on the track to use as landmarks that would mark the location of each node and where each edge would begin and end. When the car detects the blue tape, it prepares to transition to the next state and when it stops detecting the blue tape, it completes the transition and updates the current state.

Controls

Referring to the QCar Supporting Documentation, the car utilizes an inertial frame of reference to monitor the car's position and orientation, known as the "Bicycle Model", as it operates around the track. The planar position (x, y) and yaw orientation (ψ) of the car, as represented in Figure 6, are monitored by drive and steering servo motor, respectively. The drive motor controls the planar movements of the car, being forward and backward, and is built with an integrated cooling fan, which allows it to operate and prevent overheating without the need for external ventilation. Because of this, the motor has no bounds to the values the car can register, but does create error as the value increases. The steering servo has a steering range between -0.5 to 0.5 radians, equivalent to around 28°, with a time constant of 0.16 seconds. The angle is relative to the car's current position, with an angle of 0 being directly straight ahead from the centerpoint of the car. A negative angle corresponds to a right turn, while a positive angle corresponds to a left turn. In practical terms, the servo is able to respond relatively quickly to changes in steering, but may not make precise adjustments.

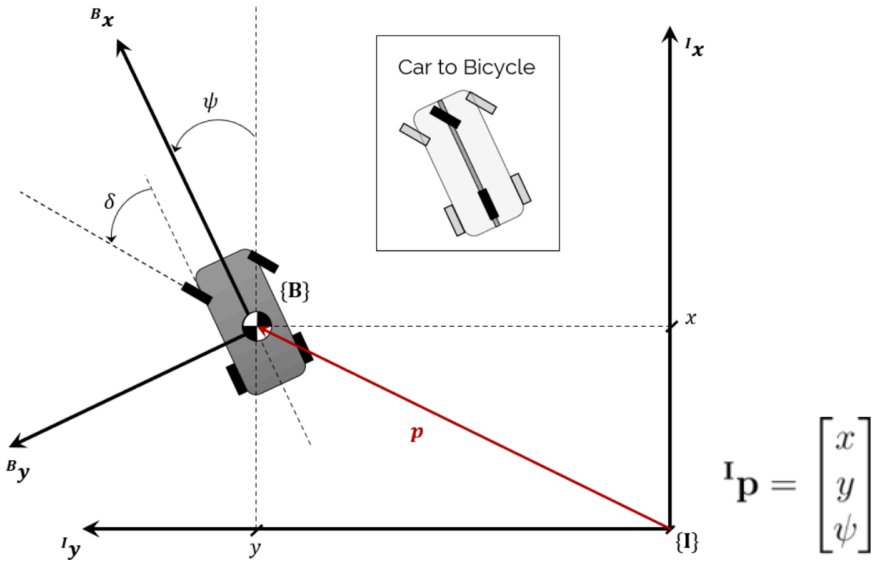


Figure 6. QCar 3-DOF Bicycle Model and Frames of Reference (Quanser, November 2020)

Alongside the drive and steering motors, the QCar is also equipped with the E8T-720-125 transmissive optical encoder from US Digital. This 720-count, pre-gearing optical encoder measures the angular position of the drive motor, using light slits and the time between each slit to generate electrical pulses known as “hardware velocity”. This value is then used to calculate the physical position change in the robot, which helps in returning the car’s velocity.

To adjust the car’s track position as it moves along the track, images taken from the CSI camera are filtered and converted to binary images such that only the white and yellow road lines appear as white, or 1, in the mask; all other colors appear as black, or 0. Then we take certain x-values from the mask and use them as column vectors. We iterate over these vectors to find the index of the last 1, or last white, in the vector. This corresponds to the y-value of the white pixel that is closest to the bottom of that screen in that vector, which is then used to approximate the car’s distance from the line. If this y-value is within an acceptable range, then the car drives straight without changing its angle. But if this y-value is outside the range, the car adjusts its heading based on how far out of the range the white pixels are. The angle that the car will turn at

can be seen in Figure 7 below, where distance is the y-value we calculated, direction is either 1 or -1 and indicates when the car should move towards the line and when it should move away, high is the maximum of the range, and low is the minimum of the range.

```
# car is close to line so turn away from line
if distance >= high:
    angle = .02*direction*(abs(distance - high))
# car is far from line so turn towards line
elif distance < low:
    angle = -.02*direction*(abs(low - distance))
```

Figure 7. Angle Formula

In application, the car will drive straight forward on straight roads, with occasional small turns, while the car will make larger turns when moving along the track's curves. The speed is also adjusted based on the angle of the car, slowing down when turning and moving at normal speed when driving straight. The car can accomodate for both turning directions, so that it can drive in either lane.

Object Detection

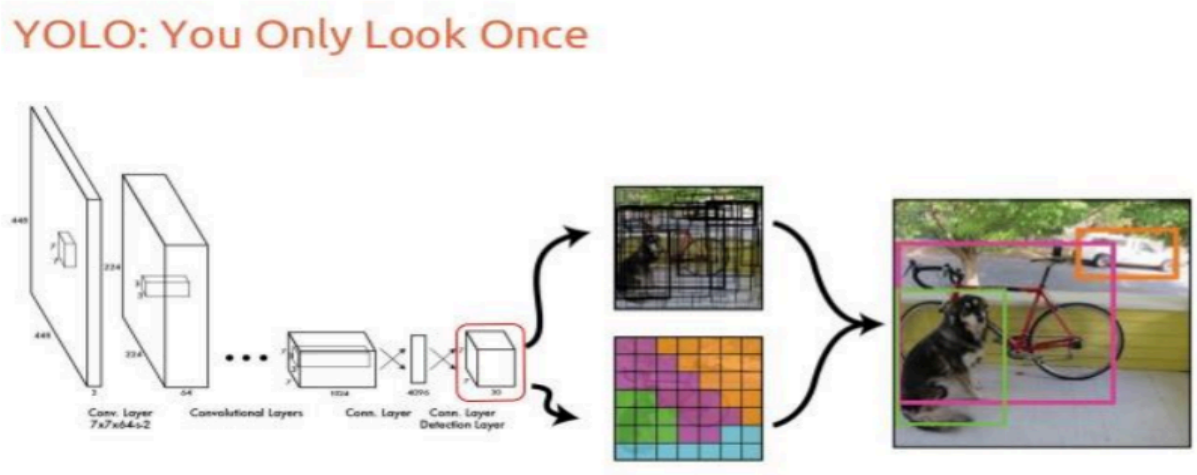


Figure 8. Visual Representation of YOLOv3 ("You Only Look Once: Unified, Real-Time Object Detection"; Redmon, Girshick, Farhadi, and Divvala)

For real-time object detection and classification of stop signs and stop lights, as well as any other scale or needed objects, we implemented a You Only Look Once version three algorithm (YOLOv3) using Common Objects in Context from Microsoft (COCO) dataset-derived pretrained weights and an OpenCV (cv2) Deep Neural Network module (DNN). The algorithm detects and classifies objects using grid cells and anchor boxes with varying scales and aspect ratios to itemize correlated predicted objects while only looking at the image once, allowing for efficiency and speed. It will resize windows as deemed necessary due to such.

When deciding the thresholds, it is important to consider the sparsity of the roadway and model strength, especially in reference to the Non-Max Suppression Threshold (NMS) and confidence threshold (validity), as it can be essential in overfiltering. With an architecture following a single-pass, multi-output layer filtering.

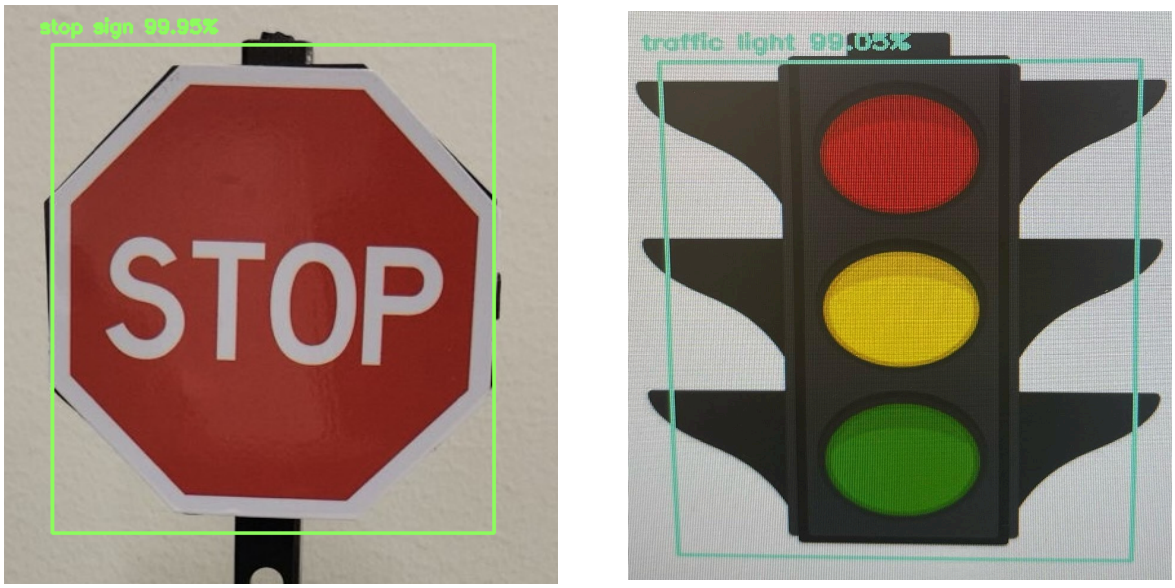


Figure 9 and Figure 10. Output of Test Images ran on YOLOv3 implementation ran on QCar with validity percents

As Figures 9 and 10 show, the combination of the DNN module with a pre-trained YOLOv3 model derived from the COCO dataset allows for strong output with high validity and smooth bounding box creation. This objectively supports the current development and steps toward real-time processing for variable change in a new course and terrain put forth for the 2025 competition by Quanser. Due to current project limitations, version three is required, but within changing roles and Quanser package upgrades, the version will be upgraded as plausible. In total, the object detection strategy allows for recognition and classification of known, trained objects from a DNN that can process and predict those same images to be used for self-driving controls and automation.

Figure 11. Output of live image feed during car operation showing yolo algorithms recognizing a stop sign, the essential aspect of our production ready system at a relatively high validation rate

Figure 12. Output of live image feed during car operation showing yolo algorithms recognizing two of the eight available classes, stop sign alongside people, both essential to real life self-driving car systems

The above figures, 11 and 12, are showing the real life implementation of the car predicting detected images. The car uses eight classes of detected objects: car, bus, traffic light, truck, motorbike, stop sign, bicycle, person, traffic signs, train, dog, cat, parking meter. These classes are the utmost essential roadway objects, but in our project we truly only use the stop sign class detection. For scale, the other classes would be useful and worthwhile for a project continuation. Inside of the YOLO algorithm, the non max suppression is set to forty percent with a validation threshold of sixty percent. These percentages are relatively arbitrary but universal, the threshold could be increased due to testing seeing around seventy-five percent average detection rate. Using an array of detected objects and the confidence threshold, the objects are determined and bounding boxes are drawn. To further limit over drawing of bounding boxes, the code detects if a bounding box is drawn within the relative same area as a larger prior bounding box. The detected

object classes, alongside the bounding box location, is returned from the YoloDC.py file. Overall, the YOLOv3 algorithm allows for real time object detection of important roadway objects while limiting collisions and possible issues that may occur within open threads.

Path Planning

To incorporate the taxi service goal of our project, we implemented the A* search algorithm to design the fastest and most efficient path for the car to travel between destination points. A* is a graph traversal and pathfinding algorithm that uses designated start and end points to find the shortest path between two nodes in a graph. It is widely used in artificial intelligence and computer science programs, exploring the graph in an informed way as compared to traditional algorithms, such as Dijkstra's algorithm. Despite being more effective, the algorithm uses the advantages of Dijkstra's in its calculation of heuristic values, ensuring the path found is as short as possible, and favoring nodes that are close to the starting point.

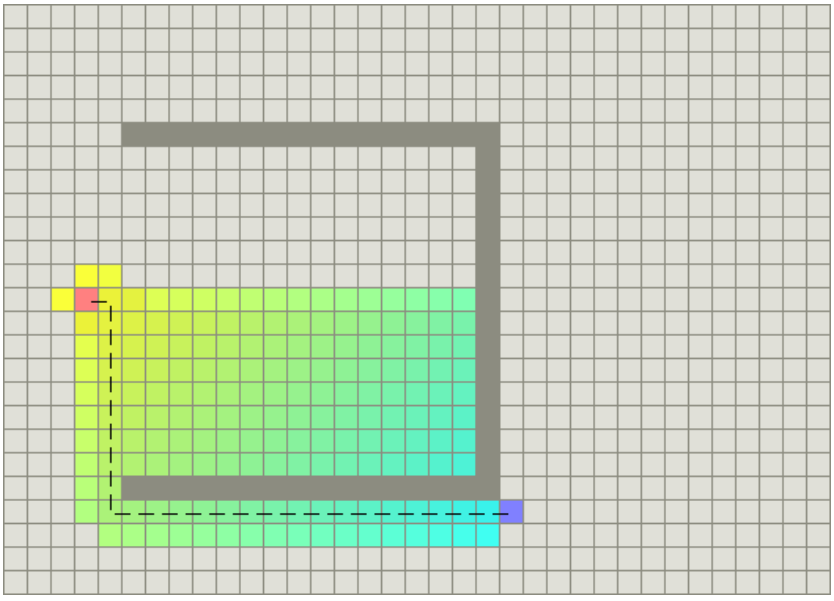
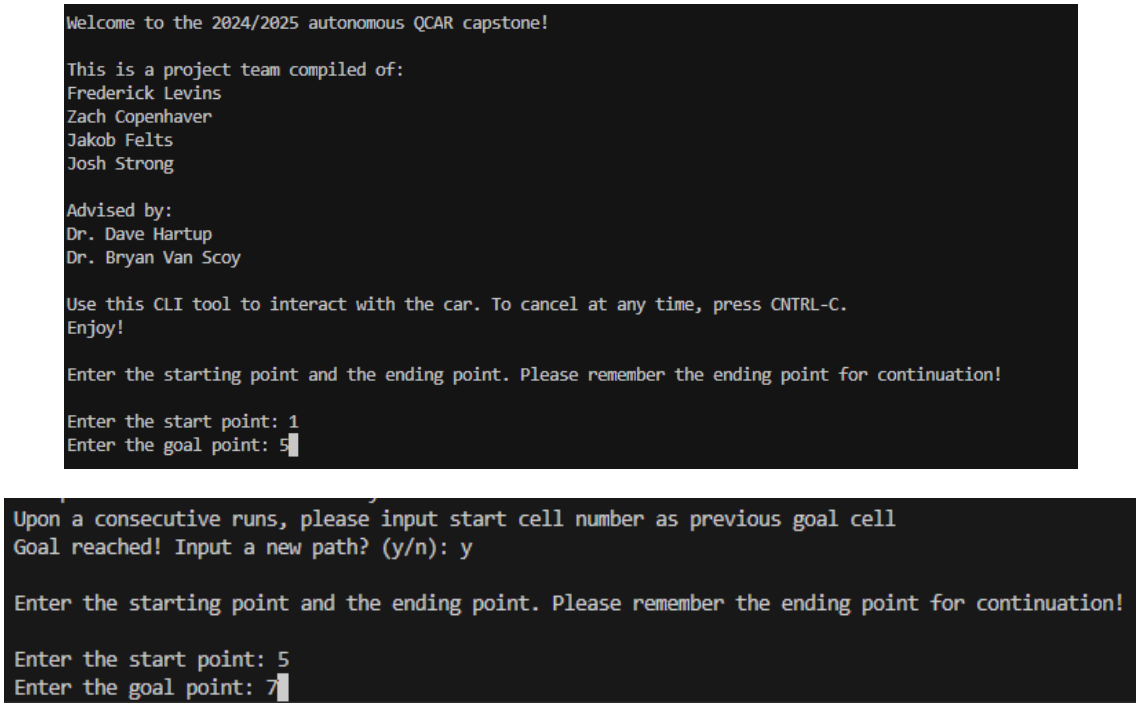


Figure 13. Example implementation of A, with colored tiles ranging from yellow (shortest) to teal (longest) between start and end positions*

By designing a map of values, labeling obstacles and walls as well as the map's individual nodes, these heuristic functions are incorporated into our solution. In A*, each node pattern is traversed, calculating the estimated distance or 'cost' of travel, and returns the path with the lowest cost, or the shortest path available. This cost is represented as a function of n, ***f(n) = g(n) + h(n)***, where g(n) represents the exact cost from the starting point to any node, and h(n) represents the heuristic estimated cost from that node to the goal. A* balances the two as it moves from the starting point to the goal, returning the lowest f(n) value for the path traveled. With this algorithm, our design would be able to adapt and work through various cases to which the car travels, including different paths and start/end nodes the car is designated to travel.

Command Line Interface (CLI) Tool



Figures 14 and 15. CLI tool output for user interaction and usage, figure X shows the initial first run and figure X shows what the continual looping output looks like.

The chosen interface for user interaction for our project is a command line interface tool that uses the output of the terminal as an interactive and easy to use interface for desired end and start nodes and is shown in the above figures 14 and 15. The usage of the command line allows for easy interaction for all individuals so that the project can be more interactive and engaging for users, but more importantly allows for overall easier usage and interaction for all purposes from testing to production. The interface is designed to initially portray aspects of the project, acknowledge the team, be polite, and be fun. After the initial run, it was designed to be simple and easy to use to input a new desired path through a simple integer input for the desired nodes.

The one main failure of the tool is the software bug during the looping for continual input leading to the inability to use the prior end node as the required start node. This issue leads to crashing of the system due to the usage of the start and end nodes within other aspects of the code. Due to this not being able to be implemented, it requires the user to reput in both start and end nodes. To handle confusion or possible misusage, the interface prompts the user to put the prior end node as the start node. If this is not followed, the robot will improperly localize itself during the run.

Miami University Lab Development

One of our goals was to develop a sufficient and sizable lab space for future self-driving car research. The objective of creating this lab is to offer a unique and equitable opportunity for staff and students alike to operate, practice, and test autonomous car theory using the QCar. Through having a dedicated space with a sole and specific purpose of working with the QCar, we are able to increase our standing in the realm of education tool development and theoretical

self-driving analysis. The lab holds two operational QCars with various track parts, traffic signs, and objects required to test self-driving concepts.

In the fall semester, we began this objective by setting up in Hughes 401, cleaning up the room and laying out some of the road mats provided by Quanser. As seen in Figure 16, we laid road mats to create a loop around the lab, with two straight segments connected by two 180° turns. This creates a simple two-lane road loop that we can test the car on. Unfortunately, this was also about the maximum size that the lab room will allow, and we needed to find and set up other spaces if we wanted to test different configurations.

Figure 16. Road Setup in Hughes 401

As we entered into the spring semester, more information was released about the 2025 AAC competition, including the team goals, track specifications, and how the car and its environment was to be utilized. This changed a lot of the landscape and setup from our development at the time, and more details and considerations were added that Hughes could not accommodate for. In our goal of continuous development of the car, we sought after various spaces to move our equipment into, providing enough space and openness to practice our designs. Spaces such as the basement of Goggins Arena, the court of Millet, and the satellite

farm were considered for this, but to allow ease of accessibility for not only our group, but also future research groups, a space in the engineering block was desired. We eventually decided on the Industrial Robotics Lab in the Engineering Building (EGB268), which allows enough space for the full size of the track, computer setups, and as mentioned, is located in the heart of the engineering college. Because of this, the lab allows for a more dynamic and open opportunity for research and development, and can continue to be added to with varying different vehicles, obstacles, concepts, ethical considerations, and variable changes in the environment. Overall, the lab is hoped to be open and available to all to learn and develop at Miami University.

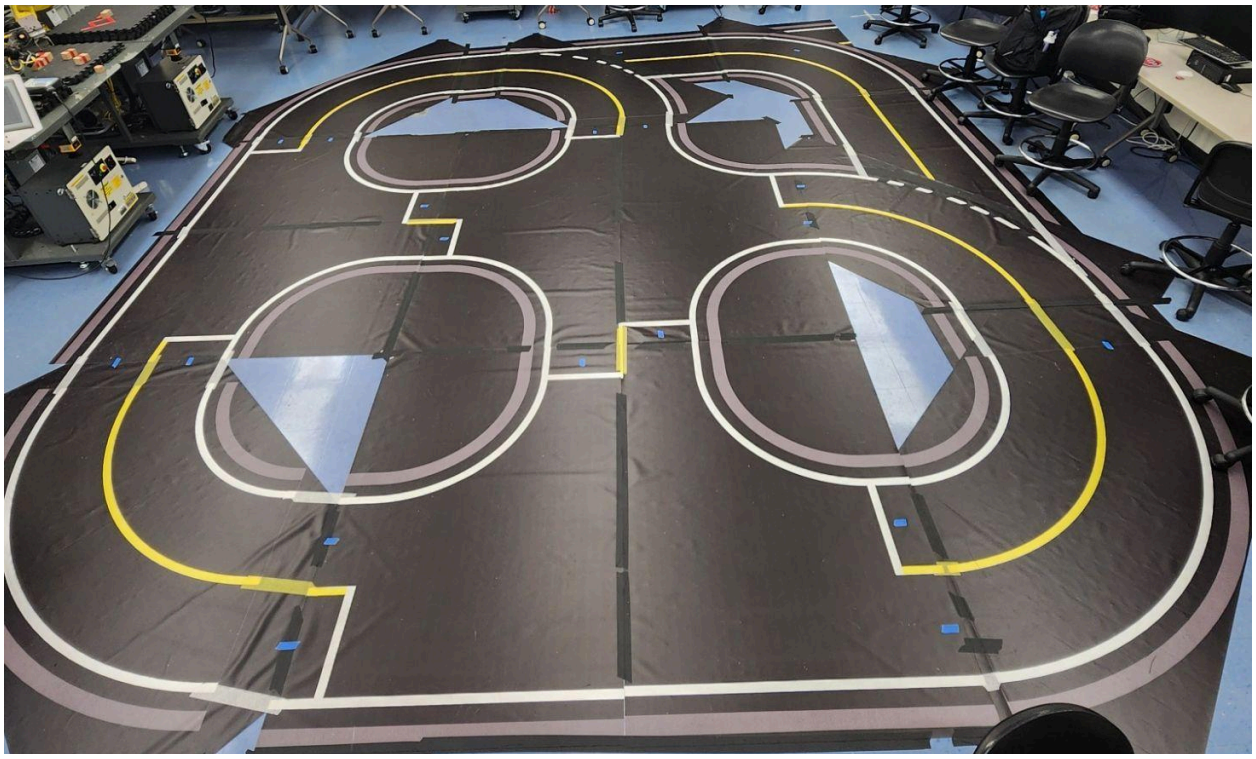


Figure 17. Road setup in the Industrial Robotics Lab in EGB268

Solution Process

Trade-off Table of Cameras

We quickly realized that using multiple camera feeds at once led to a large decrease in computing speed. We also realized that there were multiple different types of color formatting

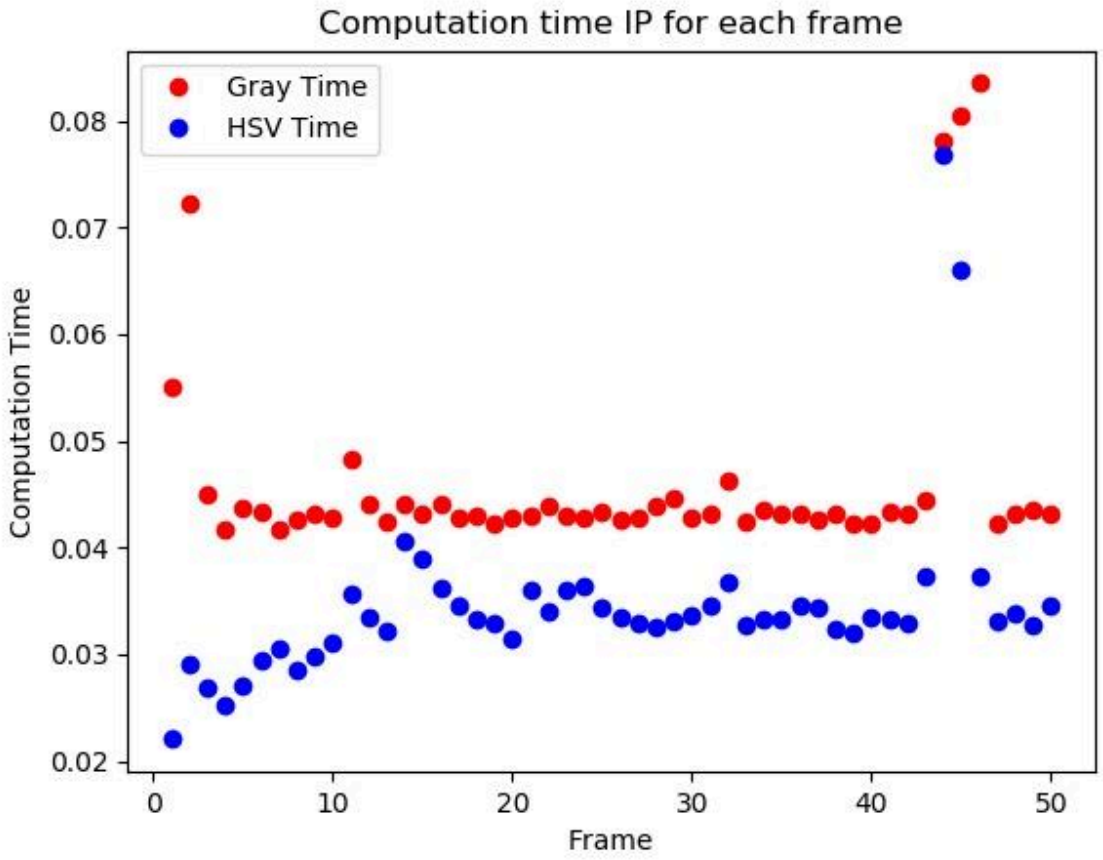
that could be used to process the images. We needed to decide between using a binary color space, where all pixels are either black or white, or a Hue, Saturation, and Value (HSV) color space, where all pixels used the above three values to determine color. In Table 1, we compared the performance of both cameras as well as the performance when using HSV and Binary color formatting to determine the type that we should use.

Criteria / Camera	**Time Efficiency**	**Visibility**	**Complexity**	**Usefulness**
RGBD	Low (due to multiple cameras in use)	Medium (all colours RGB & grayscale depth)	High (two separate images of data)	High (depth capabilities)
CSI: HSV	High	High (all colors)	High (3 channels)	Medium
CSI: Binary	Medium (due to filtering out color)	Low (only black or white)	Low (only black or white)	High (more supported)

Table 1. Trade-Offs of the RGBD Camera and the Color Formatting of the CSI cameras.

An unexpected trade-off comes from the fact that the CSI cameras are somewhat close to the ground and cannot see very far ahead; everything in the image blends together and becomes very difficult to detect. In contrast, the RGBD camera is mounted higher up, on top of the car, so it can actually see farther more accurately in addition to calculating depth.

We needed to decide which color formatting we were going to use. We timed our color formatting with both HSV and Binary conversions with the results seen in Graph 1.



Graph 1. Computation Time per Frame for Both HSV and Binary.

The first step of converting RGB to Binary is converting the color into a grayscale format, where every pixel has a value between 0 and 255, with 0 being black and 255 being white. Then we filter the image using a threshold value such that every pixel with a value above the threshold becomes 255 (white) and every pixel below becomes 0 (black). As you can see in the graph above, converting to HSV color formatting was consistently faster than converting to Binary. HSV also had another advantage in that it was able to view and process several different colors on the same feed while binary only lets us detect a few similar colors. If we wanted to detect red and blue, we could create a mask for red and a mask for blue and apply them both to the same image. If we wanted to do the same with a binary image, then we would need to find a

threshold value such that all shades of both red and blue we want to search for are on the same side of the threshold.

However, binary had some key advantages as well. It was much simpler to implement and process. Instead of having three channels to work with, binary formatting only has one, and that channel can only have one of two values, 0 or 255. This makes it much easier to directly access and utilize the value of each pixel. In addition, the three channels of HSV are not the standard red, green, and blue; they are hue, saturation, and value. This means that in order to create a mask, we would need to know the upper and lower bounds of the color in HSV which is less intuitive than RGB and would require much more time spent on trial and error. Another advantage is that binary formatting is widely used and supported. During our research, most image processing applications and algorithms we found converted their images into binary and then ran image detection on them. OpenCV also has several functions that use binary images. In comparison, we found few examples of HSV or even RGB color being used to detect images. As a result, we decided that although slower, it would benefit us more in the long term to use binary formatting instead of HSV.

As for the RGBD camera, it has the same abilities as the CSI cameras to integrate filters like Binary or HSV, but in the previous case our group decided that Binary was the clear decision on what to use. Using binary color scale on the RGB image generated from the RGB Module in the camera generates a similar output to the CSI camera's image. An immediate issue using binary imaging with the RGBD Camera is the amount of excess noise that polluted the camera from the surrounding environment. An example of this pollution can be seen in Figure 18, where the lower right section of the image has a mess of white pixels that interfere with the camera's ability to detect where the track's line is and the left side shows how the lines should be detected.

Figure 18. RGBD Binary Format with Excess Noise

This is one of the reasons that led the group to no longer pursue using the RGBD camera for the line following algorithm, which will be covered in the next section. The other reason to no longer use this camera is because using the depth perspective relied on using multiple cameras, which in turn gave longer processing times to run the programs. As for these reasons we have decided to use the binary formatting with the CSI cameras opposed to with the RGBD camera.

Furthermore, in the fall semester, we decided that we would only use the front CSI camera instead of all four CSI cameras in order to decrease the processing time of each iteration of the control loop. This semester, there were states where the front CSI would not be able to reliably see the road lines such as when turning in the four-way intersection in the middle. We decided that the best way to solve this issue would be to switch to a different camera feed that could see the lines. As seen in Figure 17, there are four rough circles formed by the intersections and bends in the road. When the car is turning left or right through an intersection, it is essentially turning around a point in one of these circles and while the front camera cannot see any lines, the side camera on the same side we are turning towards can see the curve of the circle the car is turning around. When turning right, the right camera can see the curve of the circle and when turning left, the left camera can see the curve of the circle. Therefore, in these states we turn off the front camera and switch to the appropriate side camera. We are still only using one

camera, we are just switching which camera is being used. However, the left side camera does have some issues with detecting the line. When using the left camera, the curve the car is following is much farther away than with the other cameras. We are forced to use a lower threshold for the binary mask in order to see the line properly, however, this leaves the image susceptible to noise from light glare on the road and a threshold high enough to filter out most of the glare makes the line appear choppy and disconnected. The right camera does not have this issue since it is very close to the car, however there is still a small amount of noise caused by the white stop line going across the road.

Figure 19. Left binary mask with low threshold.

Figure 20. Left binary mask with medium threshold.

Although, we do have some noise correction. Instead of checking lines near the edge of the screen, the left and right cameras both check 17 columns every 10 pixels centered on the midpoint of the screen. The mean of their y-values is used as the distance when calculating the angle of the car. Since the line should be near the middle of the screen, we assume that y-values that are too close to the top or bottom of the screen are noisy and remove them from the calculation of the mean. Also, if the variance of these y-values is too high then we know that there is error and can filter it out and recalculate the mean.

Line Following

Since using multiple camera feeds caused significant delay, we decided to use only one CSI camera in the line following algorithm. The track is a simulation with no pedestrians or traffic so we decided that we only needed to have the front CSI camera running. Removing the other three cameras from the code led to much faster processing and fixed several issues with timing that we were having. However, as stated above, there are times when the front camera cannot see any lines so we need to switch to other cameras. Specifically, when we turn right in an intersection, we use the right camera and when we turn left in an intersection we use the left camera.

The default resolution that the example Quanser modules provide is 640 x 480 pixels. For each of the cameras this is more than enough to view everything that is needed to follow the road lines around the track. In order to reduce noise and increase processing speed, we cropped the area in which the image is processed. The front and right cameras are cropped to 640 x 200 pixels while the left camera is cropped to 640 x 80 pixels to counteract the noisy image. This shows as much of the road that the CSI cameras are capable of viewing and results in faster processing times since there are less pixels to process each frame.

As previously mentioned, the algorithm detects the height of the white pixels in certain columns on each frame. Figure 21 shows an example of the binary feed of the front CSI camera when the car is driving straight on a straight section of the road.

Figure 21. Binary Front CSI Camera Feed when Driving Straight.

When we convert the camera feed into binary, we use a threshold such that both the white and yellow road lines become white while the rest becomes black. There is some error that results from lighting and other white objects in the lab, however, we were able to filter this out by increasing the threshold of the binary mask. The car primarily detects the white pixels of the two adjacent lines when driving in a lane. In fact, the car can work perfectly well merely by following only one of these lines. The algorithm has the car attempt to maintain a certain distance from the line by adjusting its heading based on its distance. As seen below in Figure 22, when going around curves, the line on the innermost curve almost disappears from view. This means that the car needs to follow the outermost line when turning. However, when the car is turning left, the outermost line is on the right side of the frame and when turning right, the outermost line is on the left side of the frame. In addition, as stated above, the front camera cannot see any lines when driving through an intersection. Therefore the car cannot only follow one line or side of the frame and needs to change which line it is following based on which direction it is turning and where on the track it is.

Figure 22. Binary Camera Feed when Turning Left.

This is solved using our finite state machine as described in a later section. We know what edge the car is in based on its current state and where it is going. Therefore we can have each individual state define which direction the car should be turning.

The only states in which we cannot use the line following algorithm are those where the car drives straight through an intersection because the lines disappear on all cameras. Instead we had the car drive straight forward with a heading angle of 0. The car is usually not perfectly aligned with the lane so every set amount of frames the car will turn a small amount to correct its heading. The angle and frequency at which it turns are different for each state, since they depend on the pose of the car at the end of the previous state. Once the front camera sees the right hand curve on the other side of the intersection, the car will begin following it. However, this usually makes the car overcorrect and turn left a large amount which would cause it to run over the yellow line in the middle of the upcoming section. To fix this, the car also searches for and follows the yellow line on the left of the screen. The result is that the car corrects itself by following whichever line is visible until it settles in the center of the lane at the end of the intersection.

Most values such as the binary mask threshold, speed and angle constants, and the range we use for determining the car's turn angle, are determined using trial and error. We rigorously tested several different values for each of these parameters until we found values that worked.

The algorithm also logs and tracks its own performance through three methods. First, the binary camera feed is saved as a video at the end of the program. Second, at the end of each frame, the algorithm writes relevant data into a csv file. Both of these methods are primarily used for testing and debugging. Third, the algorithm can reference said prior data while processing. This allows the program to reference its own state several frames ago and provides a dynamic feedback loop.

A Implementation*

To begin our implementation of A*, we first had to create a mapped system of the track that our vehicle would be driving on. The simplest way to do this was by creating a vector in the python programming where we labeled each 'node' on the track, which is every point where the car makes a decision or change from its previous commands, and had a number of attributes associated with that node. The attributes that were used for each node was a name/number, x & y coordinates in a loose grid, a list of nodes that the current node can 'see'/move to, and placeholder values for f, g, h, and a parent node to use with the A* search algorithm. A snippet of our node's class can be seen below in Figure 23.

```
class Cell:
    def __init__(self, name, x, y, reachable, special):
        self.name = name
        self.x, self.y = x, y
        self.special = special
        self.reachable = reachable
        self.g = self.h = self.f = float('inf')
        self.parent = None
```

Figure 23. Class definition for each node on track

Now that the nodes are set up, we can begin running these locations through an A* algorithm. To accomplish this, we decide on a start and end/goal node (which has been created as user inputs), and the program then creates the shortest path between these two nodes by looking at the 'reachable'/neighbor nodes and seeing which path is the least distance to the end node. To calculate the least distance we set the h(n) (distance to the goal) of the A* algorithm to the sum of the absolute difference between the x and y coordinates between the node and the goal. As the distance to the goal then shortened, this h(n) that was being added to the total distance, g(n), also decreased, allowing the algorithm to find the shortest path. This is then repeated until the

algorithm has finally reached the destination, where it will then return the finalized path back to the rest of the program.

With this generated path, we then needed the vehicle to be able to know where it is along the path/track. For this we had decided to use colored tape along the tracks to signify each of these nodes, and using image processing to search specifically for the color of the tape allows the vehicle to then see when it is approaching/arriving at the next node in the path. During our initial testing we had used red tape (as it was easily accessible to us), but this had errors in detection as well as interfered with Yolo detection of stop signs (as the similar colors resulted in mixed readings). There were also misdetections as many objects in the surrounding environment involved shades of red, resulting in nodes being detected incorrectly. A fix to this was to change from red tape to blue tape, and this led to much more reliable results of detection with little misdetections in the environment.

The next steps were to have the vehicle be able to direct itself between two separate nodes based off of its current path. For this, we had used a loose state system where the vehicle's program would look at its current and next node in the path, and would change its control's state based on how it would get from one node to the next. To do this, each state name would be a combination of the two nodes to signify that the function was specific for transitioning between those two points (Ex. To travel between nodes 5 & 6, the state would be 56). Based on which state the vehicle was in, it would use a mix of control functions to navigate the track to the next point where the current state would be updated again.

The control functions that are used within these states are based on the different navigational paths of follow left line, follow right line, left turn with no line, right turn with no line, and straight (with no line). For the two states that involve following the lines, these were

done by using the line following described in the previous section, as they were simple to implement. For the other three states, it relies on the vehicle guiding itself when the lines can no longer be seen from the forward facing camera. Because of this, we then relied on a few of the other camera perspectives to guide. For the left turn with no line, it uses the left camera to detect the furthest side line to approximately follow the directional curve. From the turn, it will then look for the lines again through the front camera to correct itself back into the lanes. For the right turn with no lines, the vehicle moves slightly to the left and then relies on the right camera to see the edge line of the road to follow until it reaches the next node. Lastly for the straight state, the vehicle corrects itself initially with the side lines and then travels straight making minor adjustments to its heading angle until it can refind the road lines.

Many of these states involve using only a single one of these control functions, but for the cases of going straight it uses the straight function as well as the line following functions to help correct itself when entering and exiting the intersections between the nodes. Each of the states that interact with an intersection also use a wait period where the vehicle will stop before continuing through the intersection to act as an ‘all way stop sign’.

One downside of our mapping though, is that since we are using specific nodes as our mapping system, this limits where we can set pick-up and drop-off locations. As we can only confidently travel and register when we are at the marked nodes, stopping at points along the road are not possible with the current version of the programming.

Feedback Speed Controller

To stabilize the car during operation, a feedback speed controller is implemented using a combination of hardware and software to regulate the motor’s speed. As previously mentioned,

the encoders provide the motor's position and velocity by measuring the drive motor's angular position as it operates, sampled at the previously defined sample time of 0.16 seconds. This generates a position feedback and a velocity feedback signal. The position and velocity feedback signals are integrated over Ts to calculate the cumulative error and velocity. This integration helps to reduce the effects of noise and disturbances. The control algorithm then uses the integrated position and velocity feedback signals to generate the control signal, which is then used to modulate the motor’s speed. Since the system operates in a closed-loop configuration, the feedback signals are continuously monitored and adjusted to stabilize the car's desired speed and position while operating.

By integrating over the sample time and modulating the motors, we are able to implement a feedback speed controller, which precisely monitors and regulates the motor speed, allowing for faster acceleration or deceleration. This controller also allows for precise positioning, which is a key component used in simultaneous localization algorithms for the car to integrate position into smoother speed and motor control.

Final Results

For each of the various routes, the car was able to drive the complete designated path around the track. Given designated start and end nodes, we used the A* path planning algorithm to determine the shortest and most efficient path the car can travel. For each node traveled, various cases are checked, and the car performs the appropriate motions to fulfill this goal, whether it is going straight, going along a curve, or making a turn. To make these curves, we used the front CSI camera to detect the white and yellow road lines and cropped the frame size to only what we need to see on the road. We converted the camera feed into binary images for

easier processing and object detection. We used a threshold that turned both the white and yellow road lines into white pixels and smoothed out a small amount of noise.

Once we had binary images, we determined the y-values of the last white pixels on certain columns and used those to determine the heading of the car. If the lowest white pixels that we are tracking are within a hardcoded range, then we drive straight. If they are not, then the car will turn. The magnitude of the turn depends on how far outside of the acceptable range the white pixels are.

The implementation of the YOLO algorithm, with the objective of real-time object detection with a current focus on stop lights and stop signs, is not fully fulfilled due to Python restrictions. YOLO is light-sensitive, struggles to detect small or crowded objects, has trouble with scale, and is performance-heavy on the car's hardware, making it hard to gather data from the camera. However, our design is able to recognize the essentials of traffic navigation and obstacles in the form of classes trained on image data. For example, using images of stop signs, the car is trained on what a stop sign looks like, and how to identify it using its cameras. This pattern is also used for people, lights, and nodes within the car's range of vision, which are then used to condition and adjust the car's stopping, driving, and turning while in runtime.

Finally, we use encoder counts to adjust the speed of the car based on its angle. The car slows down when it turns or makes large heading adjustments and speeds up when driving straight or with small heading adjustments. This enables the car to travel quickly while not compromising its steering and prevents it from significantly wobbling back and forth as it drives.

Ethics

Autonomous vehicles have been a vital part of research and development for many years, testing in various cities and shaping societies. These vehicles are designed with the reaction and processing capabilities of a human driver, and can autonomously navigate and respond to traffic patterns. This helps to improve road safety, reduce traffic congestion, and increase accessibility for travel for those who cannot drive. However, the idea of integrating machine learning into an activity that requires fast thinking, coordination, and absolute focus at all times creates a lot of ethical concerns, given the idea of simulating a self-driving car driving on a road.

In our development of the scale-model QCar, we have encountered various issues in the car's operation, including staying in-between the lines, recognizing objects, and following traffic rules, such as stopping at the stop sign. Our encounters with these issues help us reflect on the development of larger-scale self-driving vehicles, and the issues they may show in real-world environments. Even with professional testing and learning, accidents and mistakes can not always be prevented. This includes merging into lanes with cars, driving through red lights or stop signs, veering off-road, and many other traffic violations that can result in other drivers or bystanders getting severely injured. As engineers, we abide by a set of engineering standards that enforce safety and protection of others and our work while striving for innovation and continuous development. Citing from the IEEE Code of Ethics, "...in recognition of the importance of our technologies in affecting the quality of life throughout the world, and in accepting a personal obligation to our profession, its members and the communities we serve, do hereby commit ourselves to the highest ethical and professional conduct…" (IEEE 2020). By following these rules and recognizing ethical concerns in our research, we strive to set an

example for future research groups, where a self-driving car can follow traffic patterns and rules to the best of its ability.

Trade-Offs

One aspect we have discussed as a team is integrating simultaneous localization into the controls to help drive the vehicle. One part of the competition is that we are given the layout of the track beforehand in the form of the simulation, meaning that we have the whole track mapped out. Our team planned to have the vehicle recognize itself within the mapped out track, so it can then predict where it is supposed to be and can adjust itself accordingly based on its encoder values. This would allow the car to navigate and perform more efficiently, driving faster while operating across the map. A technique we considered for this is the Simultaneous Localization and Mapping (SLAM) algorithm, where the car would create a map of its environment using its LIDAR while determining its own position within the map. We decided against this approach for two reasons. First, since we have a map of the track given to us, there would be no need to map the track while localizing at the same time. Second, the track is flat with few three dimensional objects for the LIDAR to bounce off of. The lines are all drawn on the track and the only three dimensional objects on the track are the car and a handful of stop signs so we decided that the LIDAR would not be very useful.

There were several ways we attempted to control the car when the normal line following algorithm could not be used. We could have used the encoders to reference how far the car had moved and turn based on where it should be at that distance. We also tried to have the car turn based on how long it has been since the state began. Both of these attempts caused many issues. Both of them are entirely dependent on the pose of the car at the start of the state, and if this state

is not the first state the car executes, then the pose will be different every trial. Therefore, the actual location of the car after it moved the same distance would be different every time. We lacked a good way to initialize the car's starting pose which made it difficult to utilize prior control data. The timing attempt especially ran into issues because the speed of the car also depends on the car's battery voltage. At high voltages, the car runs faster while it slows down at lower voltages and this can change very quickly. These two solutions could work until ideal cases, but were not good at adjusting themselves and dealing with variation. In contrast, the line following algorithm is very good at automatically correcting heading errors as long as the car is not completely outside of the lane. We decided that it would be better if we used line following as much as possible since it was the most reliable and consistent. It was then that we noticed that for left and right turns in an intersection, the car turns around a white curve that the side cameras could see, even if the front could not. So we altered the line following algorithm for those states.

Future Work

In our original solution from the fall semester, the team's overall goal was to compete in the 2025 ACC Self-Driving Car Student Competition that will be held in Denver in Summer 2025. Although our work was not fully concrete in relation to the competition, our team wanted to make our design open-ended and augmentative for incoming research groups and potential events to work towards. For reference, our team looked at the previous 2024 competition as a baseline of what our vehicle should be able to do and then when more is announced we will modify our existing program to suit the changes. Then, when the 2025 competition and its themes were fully revealed, we switched our focus to that, adding to the work we've done as a group before the endgame was shown.

To summarize the design process, the first aspect of the competition is the simulation portion during Stage 1. However, we were not able to get the simulation fully operational, and due to the time constraints and the current direction our group's solutions were pointing towards, we eventually had to move forward without the simulation. But for future years, we hope to pass down our current work as a guidance/assistance tool, so that research groups can work with it for the Quanser competition. As a reference, Figures 24 and 25 shown below help show general insight into this timeline and the important dates correlated of the 2025 competition:

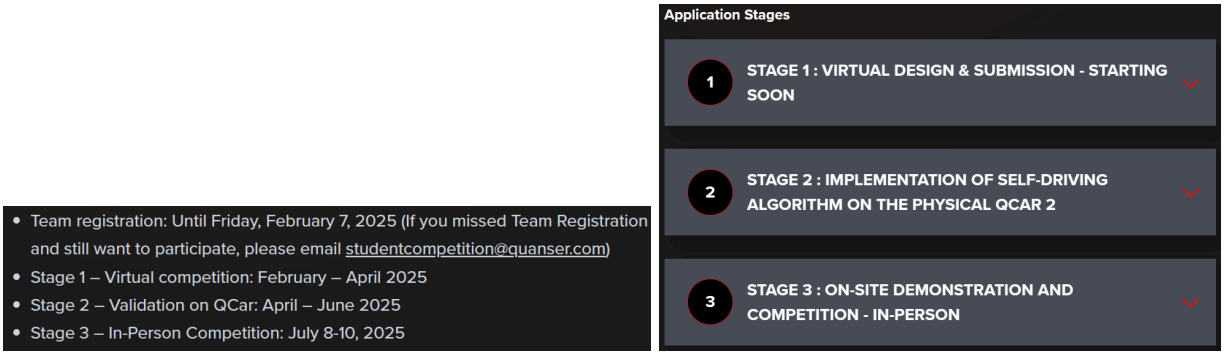


Figure 24 and Figure 25. 2025 ACC Self-Driving Car Student Competition Timeline and Objectives (Quanser, 2025)

While our team has successfully designed the vehicle to autonomously drive around the track without any stops or interference, there are various implementations we have incorporated to create an open-ended approach for setting up the car for the competition. For example, the team's YOLO algorithm detects images such as stop lights/signs, and the application is able to run while the vehicle has been driving on the track. The car is able to recognize the traffic signs and nodes for the car to act appropriately by stopping on stop signs/lines and waiting before going forward. Here, our YOLO algorithm is refined to make sure that it can clearly register images while driving without interfering with the controls or getting late responses on correcting its path.

Another aspect we have discussed is improving the noise and lighting filtering while enhancing the line detection. Light shining on and reflecting off of the track provides a significant obstacle to our line detection algorithm, as said light is often below our threshold and appears on the binary feed as large white spaces in the middle of the track. This makes it difficult for the algorithm to consistently detect the lines. Random noise is a less common but still important issue. The binary image conversion already does a good job of filtering out random noise but some still appears, especially when considering the white floor and walls of the lab room. In addition, the edges of the road mats can disappear on the camera feed. This can be seen at the top of the left line in Figure 22. This can cause significant error as the algorithm misreads the road lines and drives straight off the track. We hope to be able to resolve this issue in the future with an improved line detection algorithm.

Conclusion and Team Reflections

Concluding this report, our group has been able to implement self-driving functionality using the Quanser QCar. Our research and development of the car initially started with understanding the robot, by studying its documentation and the code used in previous competitions. As we moved away from manual controls, our team was divided to incorporate image processing and controls methods to create an autonomous algorithm for the car. The CSI cameras were used to implement a line following algorithm to stay in-between the lines. The controls used information from these feeds to move along the track and complete a full loop, as well as implementing a feedback speed controller to maintain consistent speed and performance. Combining these efforts helped accomplish the goals of having the vehicle driving autonomously.

At the start of the fall semester, the group didn't have a clear goal of what exactly was to be completed with this project as there was no precedent of it nor did we know what would be needed as the current 2025 competition hasn't been fully announced. Starting from the first team meeting, the group was able to decide on an attainable goal of having the car drive autonomously through the basic track that could fit within the research room. Every team meeting afterwards was a steady check-in with the advisors as the group made goals and deliverables each week to show progress towards the final semester goal, as it was also a point each week to ask questions for clarification on aspects and paths that were hindering the group's progress (examples include when the group was struggling to only use the specified editions of Python 3.6 or when we were getting delays in camera feeds so we were pointed towards cropping the images and different frame rates).

As the semester progressed and the group decided to divide and conquer with the tasks that needed to be completed, the team was able to notice flaws like that one member may have an issue with something while others had already found a workable solution to that problem. This led to some moments of poor communication between the team as well as wasted time when someone spent time on a task that was already completed. The weekly meetings among team members allowed for information to be communicated and spread through each other, so that everyone was also able to be on the same page as to not continue these issues moving forward once they were noticed.

Going into the spring semester, the group was able to work strongly together with a clear idea of the end goal in mind. This helped give us an understanding on how to work more efficiently and effectively together by knowing exactly what to work on, and how to implement it. Despite our lack of focus on the AAC competition, we were still able to accomplish the

competition's goals of designing and testing a self-driving taxi service. We hope that our research can provide a viable outline for future research groups to test and experiment with, and potentially be able to participate in the competition in future years. In the end, our design was made with the idea of accumulative research in mind, so we hope that research groups can take inspiration from our work, and be able to build up from our many months of research and testing.

Works Cited

ACC (2024). https://acc2024.a2c2.org/

ACC (2025). https://acc2025.a2c2.org/

Birnbacher, Dieter & Birnbacher, Wolfgang. "Fully Autonomous Driving: Where Technology and Ethics Meet." IEEE, October 2017, pp. 3-4, https://ieeexplore.ieee.org/document/8070891.

Computer vision engineer. (2023, February 27). *OpenCV tutorial for beginners | FULL COURSE in 3 hours with Python* [Video]. YouTube. https://youtu.be/eDIj5LuIL4A?si=FSZmA_2mRQe2ZeGz

Configuring stereo depth. Luxonis. (n.d.). https://docs.luxonis.com/hardware/platform/depth/configuring-stereo-depth

Dipert, Brian. "DNN-Based Object Detectors." Edge AI and Vision Alliance, 29 Nov. 2022, www.edge-ai-vision.com/2022/11/dnn-based-object-detectors/.

Foead, Daniel, et al. "A Systematic Literature Review of A* Pathfinding." 5th International Conference on Computer Science and Computational Intelligence 2020, 19 February 2021, pp. 507-514, https://www.sciencedirect.com/science/article/pii/S1877050921000399.

Gaffar, Syed Abdul. "Implementation of Yolov3: Simplified." Analytics Vidhya, 1 July 2021, www.analyticsvidhya.com/blog/2021/06/implementation-of-yolov3-simplified/.

Gillis, A. S., & Lutkevich, B. (2024, June 20). What is a self-driving car?. Search Enterprise AI. https://www.techtarget.com/searchenterpriseai/definition/driverless-car

Horner, Hayden. “Engineering Standards: What Are They and Why They Matter?” Engineering Institute of Technology, 7 Apr. 2025, www.eit.edu.au/engineering-standards-what-is-it-and-why-it-matters/.

IEEE Board of Directors. “IEEE Code of Standards.” *IEEE*, June 2020, https://www.ieee.org/about/corporate/governance/p7-8.html.

Kardell, Jeff. “An Introduction to Feedback Devices for VFD and Servo Applications.” *KEB*, 8 May 2024, www.kebamerica.com/blog/introduction-to-feedback-devices-for-vfd-and-servo-applications/.

McKay, Jackson. “Position Feedback: What You Need to Know.” *ADVANCED Motion Controls*, 24 June 2024, www.a-m-c.com/position-feedback-need-know/.

Meel, Vidushi. “YOLOv3: Real-Time Object Detection Algorithm (Guide).” *Viso.Ai*, 4 Oct. 2024, viso.ai/deep-learning/yolov3-overview/.

OpenCV (n.d.). https://opencv.org/about/

Patel, Amit. “Introduction to A*.” Stanford CS Theory, 1997, https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html.

Pineros, John. “From Concept to Reality, The Student Self-Driving Competition.” Quanser, 21 Dec. 2023, www.quanser.com/blog/research-studio/from-concept-to-reality-the-student-self-driving-competition/.

Redmon, Joseph, et al. “You only look once: Unified, real-time object detection.” 2016 I EEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, pp. 779–788, https://doi.org/10.1109/cvpr.2016.91.