



Rapid Development of an Autonomous Vehicle for the SAE AutoDrive Challenge II Competition

Sriram Ashokkumar, Anirudh Jayendra, Sam Tobin, Ariel Leykin, Robert Stegeman, Abhiraj Dashora, Bryan Look, Joseph Koenig, Brian Hu, Mason Crooks, Ishaan Mahajan, Pravin Boopathy, Mukund Krishnakumar, Nevindu Batagoda, Han Wang, Aaron Young, Victor Freire, Glenn Bower, Xiangru Xu, and Dan Negrut University of Wisconsin-Madison

Citation: Ashokkumar, S., Jayendra, A., Tobin, S., Leykin, A. et al., "Rapid Development of an Autonomous Vehicle for the SAE AutoDrive Challenge II Competition," SAE Technical Paper 2024-01-1980, 2024, doi:10.4271/2024-01-1980.

Received: 23 Oct 2023

Revised: 21 Jan 2024

Accepted: 06 Feb 2024

Abstract

The SAE AutoDrive Challenge II is a four-year collegiate competition dedicated to developing a Level 4 autonomous vehicle by 2025. In January 2023, the participating teams each received a Chevy Bolt EUV. Within a span of five months, the second phase of the competition took place in Ann Arbor, MI. The authors of this contribution, who participated in this event as team Wisconsin Autonomous representing the University of Wisconsin–Madison, secured second place in static events and third place in dynamic events. This has been accomplished by reducing reliance on the actual vehicle platform and instead leveraging physical analogs and simulation.

This paper outlines the software and hardware infrastructure of the competing vehicle, touching on issues pertaining sensors, hardware, and the software architecture employed on the autonomous vehicle. We discuss the LiDAR-camera fusion approach for object detection and the three-tier route planning and following systems. One of the defining aspects of our approach was leveraging early simulation and the use of physical analogs, which accelerated the development of the autonomy algorithms. In the process, we established a rapid autonomous vehicle development methodology that will anchor our technical effort in the third stage of the SAE AutoDrive Challenge II competition.

Introduction

Competition Background

The SAE AutoDrive Challenge II [1] is a four-year collegiate competition dedicated to developing a Level 4 autonomous vehicle [2] by 2025. It is organized by the Society of Automotive Engineers (SAE) and sponsored by General Motors, Intel, MathWorks, Cepton, Danlaw, Intrepid, OXTS, Dynamic Map Platform, Hexagon—Novatel, CTech, Nexteer, and Siemens. The competition is divided into stages separated by years, with each year's deliverables depending on the previous ones. Final demonstrations for each year take place at M-City, an autonomous vehicle (AV) test facility in Ann Arbor, Michigan. [3]

Year 1 focused on developing a working perception and control stack and concluded with final demonstrations in June 2022. The teams were given C-Tech tool carts and had to develop platforms to mount sensors for the Perception Challenge, Year 1's primary deliverable. The carts, dubbed the "Perception Carts," can be seen in Fig. 1. MathWorks MATLAB simulation was used to develop and test the control stack.

In January 2023, the team received a factory Chevrolet Bolt EUV. Over the next five months, the team was tasked with integrating and testing the autonomy stack from the "Perception Cart" and simulation onto the vehicle. Year 2's primary goal was to build on the progress made in Year 1 and enable basic automation such that the AV was able to complete four dynamic challenges:

1. Vehicle Integration Challenge: Prove that the car's acceleration, braking, steering, and Autonomous Blue Light (ABL) can be controlled via the Controller Area Network (CAN).
2. Intersection Challenge: Prove that the car can autonomously navigate intersections, given a start location, end location, and a path to follow.
3. Highway Challenge: Prove that the car can autonomously navigate a highway, including lane changes and moving through a construction zone.
4. Perception Challenge: Prove that the car can detect both static and dynamic objects of interest on and off the road while being controlled in manual mode.

FIGURE 1 The “Perception Cart” used in Year 1 of the competition.



The competition also has static challenges which include project management, presentations, and technical reports. Essentially, these are deliverables that companies require for projects before releasing or pushing them into production. These challenges were answered by a team consisting of 40 members dedicated to AutoDrive Challenge Series II. The team was split with six sub-teams focused on Perception, Localization, Controls, Infrastructure, Hardware, and Operations. The contributions of each team member enabled the team to successfully complete the principal deliverables in Year 2 of the competition. Wisconsin Autonomous [4] placed 2nd in the static challenges and 3rd in the dynamic challenges.

Before the five-month period during which the team transferred the implemented algorithms to the Chevy Bolt, six months were dedicated to developing the perception stack using physical analogs and the control stack within a simulated environment. This paper outlines how the three-tier control stack, designed to meet the challenges of the competitions and equipped with a modular setup for future adaptability, was exclusively established in simulation [5]. A key point in the context of the control stack is that only low-level control required parameter tuning when testing in reality, while higher-level functions, e.g., path generation and decision-making, were solely developed and rigorously tested within the simulation environment. A lesson learned in the process was that autonomy algorithm development can be done in isolation from the vehicle. In limited cases when simulation came short, e.g., the new sensor fusion approach that mitigates the effects of variance in object detection through redundancy and filtering, the team resorted to development using a physical analog rather than the actual vehicle.

Related Work

Developing an autonomous vehicle within a constrained time frame has been done before. In the first iteration of the AutoDrive Challenge competition, teams like the University of Toronto [6] were able to accomplish this, modifying a vehicle similar to our own and securing the top spot. Notably, their approach diverged from ours in that they made limited use of physical analogs and simulation during their development process.

Some other similar competitions include Darpa Grand Challenge [7] and the Urban Challenge [8], which presented teams with the task of creating autonomous vehicles for off-road and urban settings. However, these competitions occurred before 2010, and the algorithms employed for tasks like object detection have since become outdated. Additionally, the technology landscape at that time was different – the simulation ecosystem was less advanced and was sparingly utilized in autonomy development compared to our team’s strategy, which centered around developing the entire control stack within a simulated environment. This allowed us to rigorously test various scenarios, including interactions with stop signs, pedestrians, and other vehicles in simulation.

Examining another significant project – the Bertha/Mercedes autonomous car’s accomplishment of traversing a 100 km distance [9], it is noteworthy that their development process does not mention the use of simulation or the incorporation of physical analogs. This is the exception rather than the norm. The more recent literature highlights the important role played by simulation in the development and testing of autonomous systems. In [10], the authors explore the use of simulation in enhancing the performance, safety, and reliability of autonomous vehicle systems, underscoring how simulation can address potential issues at various development stages. The efficiency of simulation-based testing in creating diverse, complex, and controlled driving scenarios, which are challenging and risky to replicate in real-world testing, is discussed in [11].

These examples underscore the potential for accelerating the development of autonomous vehicles through the effective use of physical analogs and advanced simulation techniques. Leveraging these tools not only expedites the development process but also enhances the safety and robustness of autonomous systems, allowing for thorough testing and algorithm refinement in both virtual and real-world scenarios. This approach is particularly significant as the field of autonomous vehicles continues to evolve.

Sensor Suite

Three criteria defined the process of designing the vehicle sensor suite. First, the sensor configuration needed to be as simple as possible to accommodate the limited development time available for competition tasks while

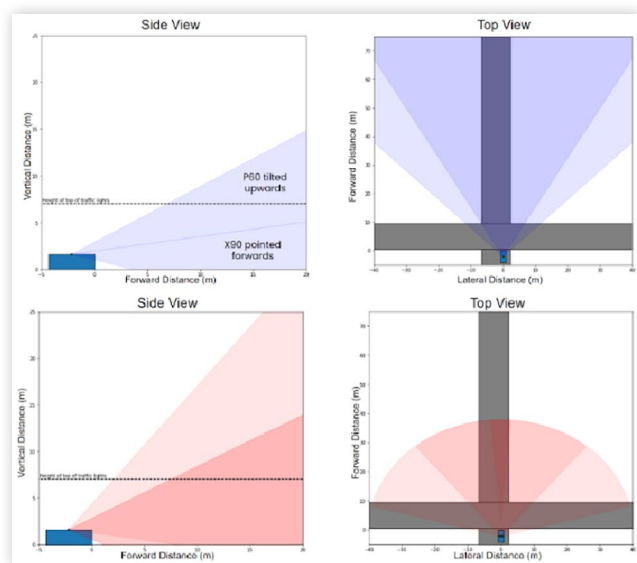
maintaining quality data intake. Second, the Sensor Suite had to have a suitable vertical field of view (FOV) to detect both objects on the road and traffic lights. We needed to see the ground about 3 meters in front of the car and traffic lights within 9 meters, which is approximately the width of a standard intersection. Lastly, the Sensor Suite needed to have a sufficient horizontal FOV to detect dynamic objects before they crossed in front of the vehicle. We found it challenging to determine the exact horizontal FOV needed but aimed for it to be as broad as possible within the other requirements.

The team used a weighted decision matrix to evaluate the different sensor configurations and also a Python script to generate the FOVs of the sensors to decide on the best sensor configuration. The consensus decision was to go with a setup of three Flir BlackFly S, one Cepton P60, and one Cepton X90 LiDAR. Table 1 shows the horizontal and vertical FOVs of the sensors, and Fig. 2 shows the FOV visualization that was used to select the sensors for the final sensor setup.

TABLE 1 The horizontal and vertical FOV for each sensor. Units used are degrees.

Sensor	Horizontal FOV	Vertical FOV
Flir BlackFly S	80	60
Cepton P60	80	60
Cepton X90	90	25

FIGURE 2 The blue rectangle represents the car while the long gray rectangles represent an intersection. The LiDAR FOV is shown in blue on the top left and top right, while the camera FOV is shown in red on the bottom left and bottom right.



Hardware and Mounting

Requirements

In Year 1 of the competition, the team utilized a C-Tech tool cart (Fig. 1) equipped with sensors, computation hardware, and electrical components for development. The hardware included an Nvidia Jetson AGX Xavier and an Intel NUC for computation, which could be conveniently housed within the cart. We continued to use this hardware while using the physical analog to collect data which can be played back on the server to perform processor in the loop (PIL) testing.

FIGURE 3 Vehicle Physical Analog

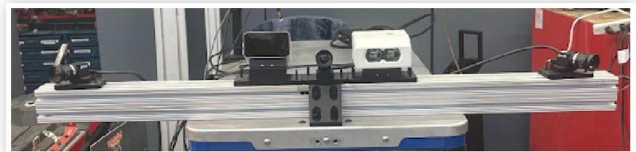


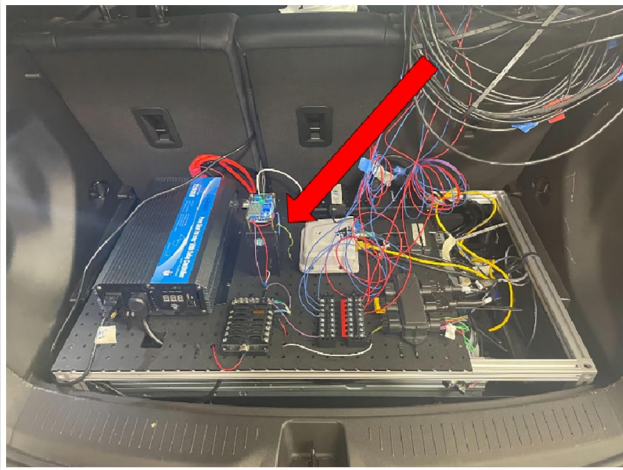
FIGURE 4 Roof rack on the car



Moving into Year 2 with the acquisition of a Chevy Bolt EUV, several new hardware requirements emerged:

- **Mounting and Dismounting Sensors:** The team needed a system that allowed for the attachment and removal of the sensor suite on the vehicle as needed.
- **Autonomous Blue Light (ABL) System:** An ABL system was necessary to indicate the vehicle's operational state (autonomous or manual) visibly to both occupants inside and observers outside the vehicle.
- **Upgraded Computer System:** The team had to install a more powerful server provided by Intel, which required a custom server rack.
- **Additional Sensors:** The hardware had to accommodate more sensors, including LiDAR and IMU antennas, and allow for adjustments in the angles of the Sensor Suite.

The team worked through several iterations to meet these requirements, starting with the development of a mounting platform. We initially considered designing and building a custom roof rack system but later opted for

FIGURE 5 Server rack and the abl system

Yakima JetStream Aerodynamic crossbars as it was more time-efficient. These crossbars provided a versatile solution for mounting sensors, antennas, and the ABL system.

For securing components inside the vehicle, especially the large development server, a custom server rack was developed. This rack also accommodated other necessary components, such as the control system for the ABLs, an inverter, the IMU, CAN buses, and the Intel Server, allowing for future expansion and adaptability. The team developed a control system that utilized the compute platform, control module, and blue lights, with a Finite State Machine (FSM) defining the system's functionality. Federal Signal MPSW9 lights with an SAE J595 Class 1 rating were mounted on the roof rack using custom 3D printed mounts.

Software Architecture

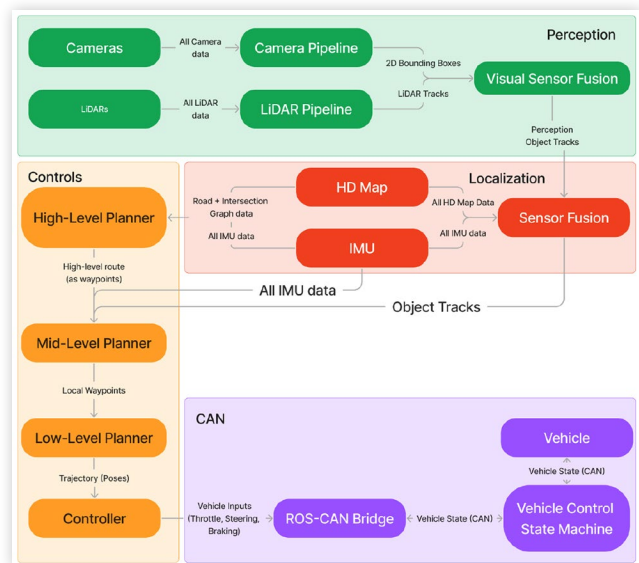
ROS (Robot Operating System)

The team chose to use ROS for the software framework as it is widely used in the industry, which opened the door to tapping into a large body of existing packages for sensor fusion and solution support. The ROS version used on the vehicle was ROS2 Humble. ROS2 Humble was chosen over other options like ROS2 Foxy or ROS1 for several reasons. ROS1 will be phased out in the medium future, and there ROS2 introduces several networking and optimization improvements over ROS1, e.g., support for real time. The choice between Foxy and Humble was more difficult to make. While Foxy has better support for drivers, it reached EOL (End-Of-Life) in April 2023, which made it questionable for our competition. Humble's EOL is May 2027, so there will not be issues of support in the near future. In addition, Humble supports composability which allows for much better optimization. As a result, the decision was made to upgrade to ROS2 Humble. Note that the composability feature of ROS2 enables grouping multiple nodes into a single process, rather than having each node operate as a separate process. This offers

several memory-related benefits, such as eliminating the need to duplicate a message's data for each callback, since the nodes now share the same memory. Instead, only a pointer needs to be copied, which is a less expensive operation. Furthermore, by combining nodes, the overall runtime overhead of the system is reduced since not as many processes need to be launched.

The ROS Stack Solution The ROS stack is divided into four main sections: Perception, Localization, Controls, and CAN, see Fig. 6. The Perception component is responsible for processing both camera and LiDAR data, analyzing it, and organizing the information extracted into an Object Track. More information is provided in the Sensor Fusion section. The Localization component takes in GPS and IMU data from the Novatel PwrPack7 IMU and employs the HD Map to determine the car's location in relation to the surrounding environment. This data is communicated to the high-level planner and fused with the Object Tracks from Perception, which are then sent to the mid-level planner. Depending on the type of object, we determine whether to utilize information from Perception, Localization, or both. For instance, when dealing with a traffic light, we collect current state data from Perception, but prioritize receiving distance data from the HD Map because our location from the GNSS antennas is precise up to 5 cm.

The Controls component is responsible for vehicle control based on dynamic principles and route planning. It uses Object Tracks and GPS/IMU data from the Localization component and road information from the HD Map to determine paths, set local waypoints, and send TBS (Throttle, Braking, Steering) values to the CAN component. More information about its functioning is available in the Route Planning section. The CAN component is responsible for receiving TBS values from the Controls component, translating them into CAN frames using DBC files, and transmitting data onto CAN buses.

FIGURE 6 ROS Pipeline

Intel Server and Autonomy-Toolkit The team received a server from Intel, which sported two Xeon processors and a discrete Intel GPU. We detail below how we used the server for our software stack in a containerized environment. All software components developed by the team are available in public repositories and released under a BSD3 license for unfettered use.

autonomy-toolkit: The objective of the autonomy-toolkit (ATK) [12, 13] is to provide a modular and portable framework for developing, testing, and deploying autonomous algorithms in simulation and reality. This toolkit is a Python package that leverages Docker by wrapping Docker Compose to build and deploy a multi-container system specifically designed for autonomy research. ATK wraps Docker Compose in that all of the functionality of Compose is still available; however, ATK also provides utilities, defaults, documentation, and examples specifically for applications relating to autonomous algorithm development. ATK is open-source, available on GitHub, and has been made available through the Python Package Index (PyPI) [2]. All our software and drivers are developed and imported, respectively, in a custom ATK environment.

OpenVino Integration: Neural networks utilized by the perception stack are deployed on the server using the Intel OpenVINO software kit. Models were trained using PyTorch and exported to ONNX weights, which were then converted to OpenVINO IR format using the Model Optimizer tool. A general C++ class that shares OpenVINO resources (such as the `ov::Core` inference engine) was created for running inference through OpenVINO. Each model that was created, whether that be a classifier or object detection model, has its own class that inherits from the base class and overrides methods to implement pre- and post-processing specific to that model. Model inference was parallelized using the high throughput performance hint and specifying the use of multiple inference streams.

HD Map - ROS Integration: The HD Map provided by Dynamic Map Platform North America (formerly known as Ushr) was integrated into the system to aid in path planning and object detection. Utilizing GPS data about lane lines, centerline information for each lane was developed by the team to guide the vehicle's movement. This approach ensures that the vehicle stays at a safe distance from other lanes while traveling and provides an additional layer of redundancy for lane line tracking by the Perception team. The vehicle's location on the HD Map is determined using a combination of GPS and IMU data. GPS data can narrow the location down to a couple of segments, and the IMU data helps match the car's heading with that of a segment to finalize its positioning. To communicate data from the HD Map to the Controls team, a ROS service is used that runs after all the route information has been processed.

Software Pipeline Optimization *MATLAB-ROS bridge:* To enable quick Software in the Loop testing with the MathWorks simulation environment, we established a communication link between our ROS2 environment and

the MATLAB/Simulink simulation. We achieved this by customizing MathWorks' public MATLAB Docker image to meet our requirements, integrating it into our ATK environment, and utilizing MATLAB's ROS Toolbox to configure the necessary publishers and subscribers for exchanging messages with our ROS2 stack. A significant benefit of using a Docker service instead of running MATLAB locally is that it ensures that everyone's MATLAB environment is consistent and includes the required add-ons, regardless of the platform. This MATLAB-ROS communication infrastructure simplified the testing of control software in a simulated environment for our Controls sub-team.

Composability: We implemented composability to enhance the communication speed between nodes, especially in perception, where we work with large messages like image and point cloud data. Composability provided a non-negligible boost in intra-process memory performance. For example, our camera drivers publish images to our vision and lane line detectors at a rate of 30 frames per second. By using composability, we only need to copy a 4-byte pointer to each of the callbacks instead of copying the entire 3000×4000 resolution image each time it is published. This change has led to considerable computational time savings. Previously, without composability, the rate at which image data moved through the image processing pipeline was capped at around 1-2 Hz. However, with the integration of composability, this rate has been increased to up to 40 Hz. This enhancement has been crucial in making our Perception Stack function efficiently, handling high-resolution image data at improved processing speeds.

Sensor Fusion

A few main criteria were established for the sensor fusion system based on prior experience.

- Fully utilize all of the data captured by the Sensor Suite
- Reliably detect objects on the road, even if of an unknown type/class
- Incorporate redundancy into object detection
- Reduce potential noise in detection

Fully utilizing all of the sensor data was deemed to be of utmost importance – as more information is incorporated into object detection, the overall detections becomes more reliable. The second requirement is a direct result of the requirements set forth by the perception challenge for the Year 2 competition. Since the Perception component is being deployed on a moving car, the sensor fusion algorithm should not discard unknown objects that the vehicle could potentially run into/over. The third and fourth requirements, redundancy of detections and smoothing out noise in detections, are a result of the team's past experience of the pitfalls observed due to too much reliance on an object detection

neural network for detections. The new sensor fusion approach will mitigate the effects of variance in object detection via redundancy and filtering.

Sensor Fusion Development

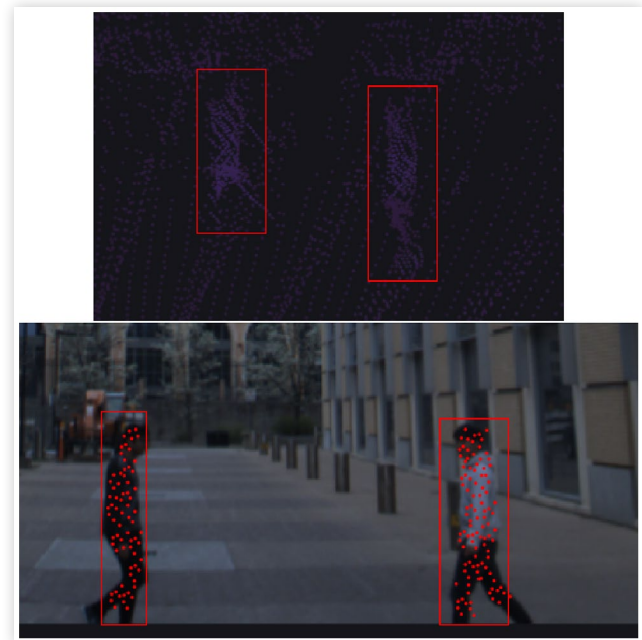
In our effort, it quickly became apparent that to fully utilize all data captured by the Sensor Suite, a pipeline exclusively dedicated to object detection on LiDAR data was necessary. Simultaneously, an object detection pipeline based on image data was also required, as colored image data is crucial for classifying the state and type of many objects such as traffic lights and signs. The result was not only the comprehensive utilization of the Sensor Suite but also an allowance for potential redundancy in object detections.

Another key observation made about the sensor data was that clustering LiDAR point cloud data for object detection is potentially much more reliable and all-encompassing than performing object detection on image data. LiDAR data is not heavily affected by lighting, weather, or other conditions, making the data and detections more consistent [14]. This means that heavier reliance on LiDAR data is likely to reduce the noise and inconsistencies in detections. More importantly, the clustering of LiDAR data is unaffected by the type of object being detected, meaning that comprehensive training data is not required for consistent detections. This is in contrast to object detection and classification using image data, where the detection of an object not present in the training data is highly unlikely. The indifference of LiDAR clustering to object type is crucial for achieving the second requirement of reliably detecting road objects, including potentially unknown ones. It is important to note that LiDAR detections here refer to the detection of an object's presence and not its classification, which will be done by fusing with camera data.

The result of this observation was the proposal of a sensor fusion algorithm called the LiDAR-Based Approach, which prioritizes LiDAR detections and uses camera data solely for classifying clusters found in LiDAR data. Each detected cluster is projected into 2D, and its 2D bounding box is used to obtain a cropped image that provides details of the detected object with colored image data. This cropped image is then fed into a neural network classifier that assigns a class ID from the set of competitive objects or determines that it is outside the set of competitive objects. Figure 7 demonstrates the projection of LiDAR clusters onto images from the camera. The top image shows clusters detected in the 3D point cloud indicated with red bounding boxes, while the bottom image shows the projection onto the 2D space, and a 2D bounding box is projected on the next cluster.

The LiDAR-Based Approach improves upon the Image-Based Approach in that it relies more heavily on LiDAR data, which increases the consistency and reduces the noise of detections. Additionally, for reasons described earlier, there is a very high probability of detecting unknown objects in the road. However, this algorithm suffers from similar problems as the Image-Based

FIGURE 7 Visual demonstration of projecting LiDAR clusters onto the image as in the LiDAR-Based Approach



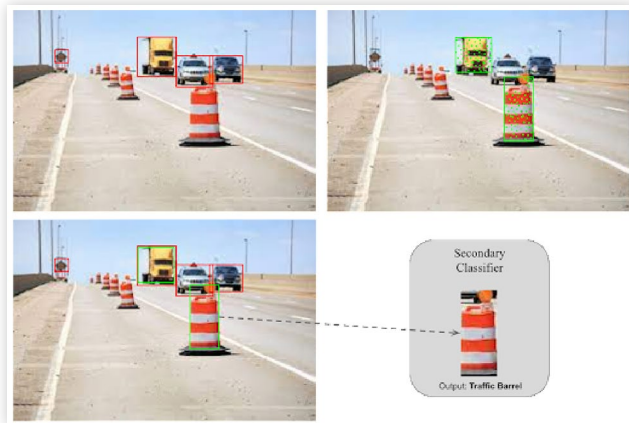
Approach. Firstly, there is a lack of full utilization of sensor data, since image data is ultimately not being used to make detections, only classifications. Furthermore, there is no redundancy in detections, since there is nothing to cross-reference with the LiDAR clusters.

The final proposed algorithm, the LiDAR-Image Approach, extends the above algorithm by running an image object detection pipeline in parallel with the LiDAR object detection pipeline. Detections made by the image pipeline are associated with detections made by the LiDAR pipeline in the sensor fusion step. Because the image pipeline is limited in what it can detect, there will be LiDAR detections that are not associated with image data. These extra detections are classified by following the same process as in the LiDAR-Based Approach: clusters are projected to 2D, its 2D bounding box is used to crop an image, and the cropped image is fed into a neural network classifier.

In the top left image of Fig. 8, sample image detections made by the object detection model are drawn on top of the input image. Note that these are limited to pedestrians, vehicles, traffic signs, and traffic lights. In the top right image of Fig. 8, some sample LiDAR detections are projected onto the input image, with their 2D projected bounding boxes drawn around them. In the bottom left image of Fig. 8, image and LiDAR bounding boxes are associated with each other. Based on the IoU metric, the truck bounding box has an association with LiDAR detection but the barrel does not. In the bottom right image of Fig. 8, LiDAR detections without an image detection association (in this example the traffic barrel) are classified using the secondary classifier.

The LiDAR-Image Approach is considered the most suitable approach as it has better sensor data utilization

FIGURE 8 Visual demonstration of the LiDAR-Image Approach.



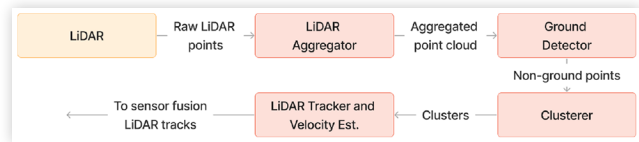
than the previously discussed approaches. This is because it performs object detection on both image and LiDAR data. Additionally, the detection of objects in both 2D and 3D provides redundancy in object detections, making the fused detections more reliable and consistent. Similar to the LiDAR-Based Approach, the heavy reliance on LiDAR clusters allows for reliable detection of road objects, regardless of the object's type. Moreover, the filtering process in the LiDAR-Based approach can be extended to incorporate image detections, as discussed later. However, a minor drawback of this approach is its complexity compared to other approaches.

All the proposed approaches were evaluated against the criteria used to design the sensor suite and an additional criterion for simplicity. The analysis indicated that the LiDAR-Image Approach is the most suitable for the requirements set forth. In comparing the LiDAR-based approach with the LiDAR-Image approach, we observed an average 15% increase in detection accuracy for pedestrians, cars, traffic signs, and traffic lights. Here, detection accuracy is defined as a correct bounding box ($\text{IoU} > 0.6$) and distance within ± 0.5 m. Furthermore, for all other objects, there was an average increase of 6% in detection accuracy, attributable to the classifier having fewer categories to discern among. The following section will provide a detailed description of both the image and LiDAR pipelines.

Image and LiDAR Pipeline

The LiDAR pipeline (Fig. 9) consists of four successive processes: aggregation, ground segmentation, clustering, and tracking. Aggregation is performed to consolidate the point cloud data received from multiple LiDAR sensors. The aggregated point cloud is transformed into the car coordinate frame; the points further than a given distance ($> 40\text{m}$) are filtered out. Ground segmentation uses the RANSAC algorithm [15] to fit a plane to the point cloud, which allows it to separate the ground points from the non-ground points (or "object points"). Clustering takes the object points and identifies clusters of points using

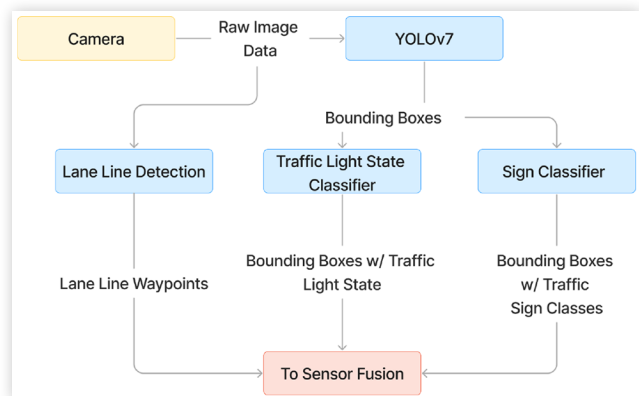
FIGURE 9 Node diagram and outputs of the LiDAR pipeline.



the DBScan algorithm [16]. The orientation (yaw angle) of clusters is found in this step by using L-shape fitting. The detected clusters are then passed to a tracking algorithm, which uses a combination of Kalman filters to assign persistent IDs and create tracks for clusters detected in the previous step. During this process, velocity for each cluster is also determined. The tracks (not the clusters) are published and represent the output of the LiDAR pipeline. The tracks can persist for multiple inactive frames before being pruned, meaning that even if LiDAR detections fluctuate the unique ID will persist.

The image pipeline (Fig. 10) consists of four key stages: object detection, traffic sign classification, traffic light state determination, and lane line detection. Object detection is performed using the YOLOv7 neural network, which detects and classifies objects in the images, producing bounding boxes with a corresponding class ID. Traffic sign classification and traffic light state determination occur in parallel with object detection and aim to further classify the traffic signs and traffic lights detected by the neural network. In traffic sign classification, a classifier neural network is used to determine the specific type of traffic sign from a set of required sign types, including stop, yield, speed limit, left turn only, right turn only, and railroad crossing. When applicable, the speed limit is determined using digit recognition. In traffic light state determination, a small object detection network is used to localize the lit bulbs of the traffic light, which are then classified by color (red, yellow, green) and pictogram (left turn, right turn, circle). The resulting updated bounding boxes, with additional classifications, are passed on to sensor fusion. In parallel, lane line detection is performed using color thresholding and a Hough transform to

FIGURE 10 Node diagram and outputs of the image pipeline. Note that traffic signs, traffic lights, pedestrians and vehicles are the only objects detected by the image pipeline.



identify lane lines and limit lines in the images, along with the type and color of the line. Lane line information is provided as 2D waypoints, which will be associated with 3D information in sensor fusion.

In Year 1, one of the major challenges was obtaining sufficient data to train an accurate object detection neural network that covered all the required classes in the competition. Specifically, it was difficult to obtain labeled data for certain classes such as deer, traffic barrels, railroad gates, and type III barricades in sufficient quantities and quality to achieve high accuracy. This class imbalance in the training data resulted in less accurate or less confident predictions for classes with less data. To address this issue, it was decided to focus the image pipeline on detecting the four classes for which there was abundant data: traffic signs, traffic lights, pedestrians, and vehicles. The LiDAR clusters would then be used in conjunction with sensor fusion to detect and classify other objects that were not in these four classes.

Sensor Date-Time Synchronization

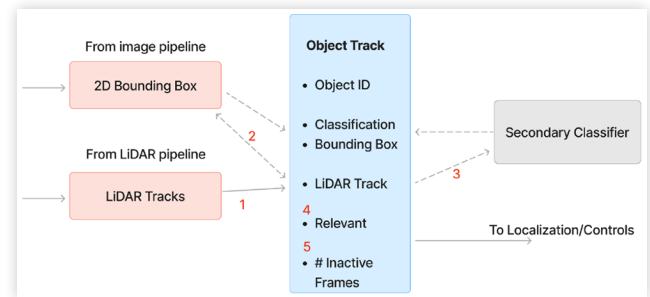
Effective synchronization of sensor data is mandatory for sensor fusion, ensuring that image and LiDAR pipeline detections correspond closely in time for accurate fusion. To achieve this, the *message_filters* ROS2 package [17] employs an approximate-time synchronizer, aligning incoming messages by comparing their timestamps through an algorithm (Fig. 11). The object tracks generated by the sensor fusion algorithm utilize the timestamp in the aggregated LiDAR point cloud since it most accurately represents the timestamp of the detection made by both the LiDAR and camera pipelines. This ensures that the sensor fusion algorithm fuses the detections made by both pipelines at roughly the same point in time.

LiDAR-Image Approach Overview

The LiDAR-Image Approach involves four consecutive steps: bounding box-cluster association, unknown object classification, object relevance determination, and track maintenance. The sensor fusion node also uses tracks to smooth out fluctuating object detections, where each cluster track published by the LiDAR tracker is given an associated object track with the same ID. An object track contains information about LiDAR detection (in the form of a cluster track) and image detection (in the form of a bounding box with classification).

In the first step of the sensor fusion process (labeled with a 1 in Fig. 12), each object track undergoes bounding box association, which involves associating the 2D

FIGURE 12 Diagram of how object tracks are handled in sensor fusion.



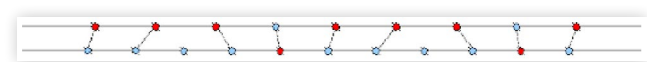
bounding boxes from the image pipeline with the clusters from the LiDAR pipeline. To do this, each cluster is projected to 2D and each projected 2D bounding box is stored. The cluster bounding boxes are then compared to the image pipeline bounding boxes using the Intersection Over Union (IoU) metric (labeled with a 2 in Fig. 12). A linear assignment (Hungarian Algorithm) [18] between the projected LiDAR bounding boxes and the 2D image bounding boxes based on the IoU score is used. If an image bounding box and a cluster bounding box “point” to each other, then the image bounding box is associated with the corresponding cluster. However, it is not always possible to register a cluster for objects far away (LiDAR points > 40 m away are removed in the point cloud aggregation step), so not all image bounding boxes may have a corresponding cluster.

Most clusters will not be associated with an image bounding box. One reason for this is that the image pipeline only detects traffic signs, traffic lights, pedestrians, and vehicles, so clusters for traffic barrels, barricades, railroad gates, and deer will not have an associated image bounding box. Another reason that most clusters will not have an associated image bounding box is that many clusters will be extraneous and not immediately relevant. For example, the clustering algorithm will return a cluster for each tree on the side of the road or the wall of a building across the street. While these clusters are important if they impede the path of the vehicle, they can be ignored if not in the road.

Determining which of the unassociated clusters to care about is handled by the next step in the sensor fusion algorithm: unknown object classification (labeled with a 3 in Fig. 12). In this step, clusters with no associated image bounding box are classified based on the image data that they correspond to. The image is cropped based on the projected cluster bounding box and fed to a neural network classifier. This neural network classifier is trained on data from the classes not covered by the object detection model in the image pipeline (ideally, all traffic lights, traffic signs, pedestrians, and vehicles have been detected and associated properly during the bounding box association step).

This *unknown object* classifier has also been augmented to determine if an image is “in-distribution” or “out-of-distribution”. In-distribution data is data that resembles training data; in this case, traffic barrels, type

FIGURE 11 Synchronization performed between two topics with different frequencies. Red dots indicate the pivot message, and the line links messages in a set that are sent to the callback by the synchronizer.



III barricades, deer, and railroad gates. Out-of-distribution data is data that does not resemble training data (e.g. trees, buildings, bushes, etc.). This allows it to differentiate between objects that matter and objects that do not matter. The result of this process is that each cluster has a label for its class, and objects that are not in the set of competition objects are labeled as “unknown”.

Next, all unknown objects are filtered. The filtering step sets the “relevant” field of the object track to false for clusters that were not determined to be a competition object, but only if they are outside the road and not moving at a significant velocity (labeled with a 4 in Fig. 12). This is done in an effort to reduce the complexity of the scene passed to Localization and Controls. Finally, track maintenance is performed. Object tracks that no longer have a LiDAR track association are removed. This is because LiDAR tracks themselves are persistent (due to the design of the LiDAR tracking algorithm). Thus, if the LiDAR track with a given ID has been deleted, then it can be certain that the object is no longer in the scene.

Object tracks that had a previous 2D bounding box association (e.g., if the classification is one of the things detected by the image pipeline, but does not have an association in the current frame) are persisted, but the “inactive frames” field is incremented (labeled with a 5 in Fig. 12). This field is reset to zero when a bounding box association is made again. Past a certain threshold, the object track is deleted. This persistence is done in an effort to tackle the fluctuations that are often observed in image pipeline detections.

Physical Analog Usage - Perception Cart

The perception cart developed in the first year of the competition was critical in developing the perception stack. Using the perception cart as a physical analog to the vehicle facilitated ease of use while testing various sensor setups and perception algorithms. Data collection was simplified by moving the cart around, and the performance of the perception stack was evaluated using the data gathered remotely on the server. This approach enabled efficient development, as the actual vehicle was in general simultaneously needed by multiple sub-teams. For example, the use of the perception cart allowed for the testing and development of the perception stack while the controls sub-team tuned their controller parameters using the vehicle. This was greatly beneficial due to the short timeline prior to the competition.

HD Map and Perception Sensor Fusion

The fusion between the HD Map and perception data is a key component of our overall system. While both data sources are important, each has its own unique strengths that make them useful in different ways. HD Maps provide highly accurate and reliable information about lane lines

and drivable waypoints. This information is used outright in almost all circumstances, except when GPS data is absent or unreliable, at which point perception lane line data is used as a backup. This ensures that the autonomous vehicle is always able to navigate safely and reliably.

By fusing the information from both the HD Map and perception data, the autonomous vehicle is able to make highly informed decisions about its environment. The HD Map provides reliable information about the layout of the road, while perception data provides real-time information about the environment, including the location and movement of surrounding objects. By combining these two data sources, the system is able to generate a highly accurate representation of the environment in real-time, which is used to make decisions about how the vehicle should navigate its environment.

Route Planning

Control Stack Structure

Year 2's objective was to develop a comprehensive route planning system that allows closed-loop route planning from Point A to Point B. The route planning system needed to include a high-level planner that provides the shortest path between Point A and Point B, similar to Google Maps. This high-level planner would be responsible for generating a global path that the vehicle would follow to reach its destination. The mid-level planner further interpolates the path output from the high-level planner. It takes actions based on input from perception and localization, such as obstacle avoidance, reactions to traffic lights/signs, lane changes, turns, etc. Finally, the low-level planner creates a trajectory given the current state of the vehicle and the target state of the vehicle. The state would include the position, velocity, and heading of the vehicle. The low-level planner is responsible for generating a smooth and safe trajectory by considering constraints such as acceleration, jerk, and torque limits. These three planners work together to navigate situations presented in the Intersection Challenge, Highway Challenge, and scenarios for the Simulation Challenge.

Development

For the high-level planner, the team debated between using Dijkstra's Algorithm [19] and the A* (A-star) algorithm [20]. The latter was chosen because it is more efficient than Dijkstra's algorithm, given the edge weights (distances between intersections) available through the HD map of M-city. The high-level planner was integrated into the ROS stack as a service node, allowing the control stack to call upon it only when performing initial routing or rerouting.

For the mid-level planner, a similar approach to the one used by the University of Toronto [7] in SAE AutoDrive Challenge I was adopted. Initially, Python and a simple

Matplotlib simulation that took lane data from manually written CSV files (as shown in Fig. 9) were used for its development.

The mid-level planner is formulated as a nonlinear optimization problem to identify the best path when needing to make a lane change, avoid an obstacle, react to traffic lights/signs, and handle other situations. The optimization problem is based on three criteria: obstacles, curvature, and deviation. It is expressed as follows, with N points on the path:

$$E = \sum_{i=1}^N E_{i,cur} + E_{i,dev} + E_{i,obs}. \quad (1)$$

With this nonlinear optimization problem, the aim is to identify the path $P = [p_1 \dots p_N]$ that minimizes the cost E . Each pose p_i in the path P is defined by x , y , and heading Θ values. The curvature term seeks to limit any jerky or sharp curves and steering by weighting heading change:

$$E_{i,cur} = w_{cur} \left(\frac{\Delta\Theta_i}{|\Delta x_i|} \right). \quad (2)$$

The second term in the optimization problem is the deviation term, which tries to ensure that the generated path stays close to the center of the lane:

$$E_{i,dev} = w_{dev} |x_i - c_i|, \quad (3)$$

where c_i is the closest point on the lane's centerline to the pose x_i .

The final term in the optimization problem is the obstacle term, which aims to keep poses in the path away from any detected obstacles. It also takes into account the distance of the car to the obstacle, and only considers obstacles within some distance threshold d , so as to not give undue importance to far-away objects:

$$E_{i,obs} = 0, \quad \text{for } |x_i - o_i| \geq d$$

$$E_{i,obs} = w_{obs} \left(\frac{d - |x_i - o_i|}{d} \right), \quad \text{for } |x_i - o_i| < d. \quad (4)$$

Each of these terms includes tunable weight parameters, denoted as w_{cur} , w_{dev} , and w_{obs} , which can be adjusted to generate a smooth, obstacle-free path. These weights are manually adjusted after running the mid-level planner and observing the resulting behavior.

To formulate and solve the optimization problem in the code, the CVXPY [21] and OSQP [22] solvers were initially used. However, it was discovered that the optimization problem was non-convex and not compatible with these solvers. Alternative optimization algorithms, such as L-BFGS, were explored, but ultimately, a gradient descent algorithm was chosen to solve the problem taking inspiration from [6]. This gradient descent

algorithm utilized an initial guess, denoted as P_0 , representing the centerlines of each lane on the path, and an approximated gradient

$$\frac{\partial E(P^{(t)})}{\partial x_i^{(t)}} \approx \frac{E(P^{(t)} + \Delta P_i) - E(P^{(t)} - \Delta P_i)}{2\Delta x_i^{(t)}}. \quad (5)$$

With this approach, a list of waypoints is generated, enabling the vehicle to respond to changes in the environment while maintaining a smooth path.

The low-level planner focuses on trajectory generation for the vehicle. We developed a planning approach called FlatVCP [23], which was proven to be highly effective. This approach involves solving an optimal control problem that is relaxed into three second-order cone programs (SOCPs) that are sequentially solved using YALMIP with MOSEK. These SOCPs find the safe path, trajectory duration, and speed profile, respectively. By solving these SOCPs, a sub-optimal but feasible trajectory that takes into account the current state and target state of the vehicle is generated. FlatVCP, as shown in Fig. 13, has demonstrated its effectiveness as a planning approach that enables the generation of safe and feasible trajectories. A simple PID controller was used to interpret the trajectories and output the desired car inputs, including linear acceleration, heading rate, and steering angle.

To close the simulation-to-reality gap, differences in behavior between simulation and reality needed to be identified. To achieve this, a mock mid-level planner that outputs hard-coded waypoints was used. This allowed the low-level planner and controller to be tested independently in reality, and the results could be compared with those in simulation. Graphs generated in MATLAB (Fig. 14) were used to visualize values in simulation, such as speed and position, while Foxglove (Fig. 15) was used to visualize values such as torque in reality.

This approach allowed for testing the low-level planner's ability to follow waypoints and the Control Stack's ability to maintain the vehicle's stability and safety in both simulation and reality. Once the low-level planner and the controller were tuned and provided performance similar to reality in simulation, the team continued to test and

FIGURE 13 Representation of the FlatVCP Problem

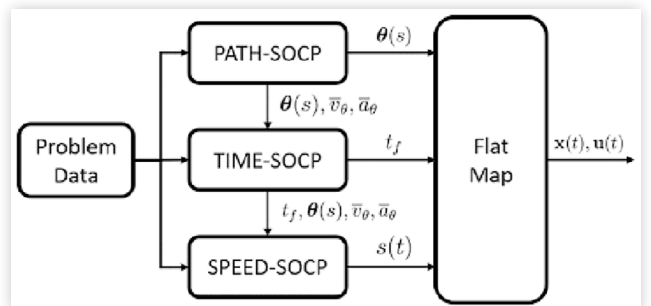
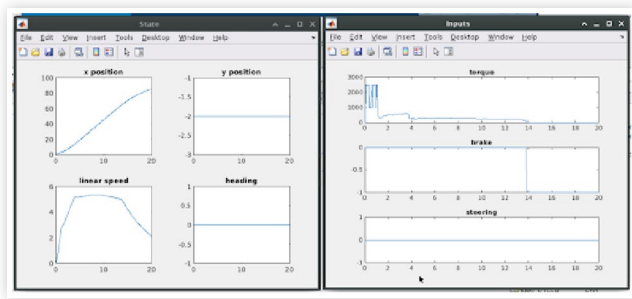
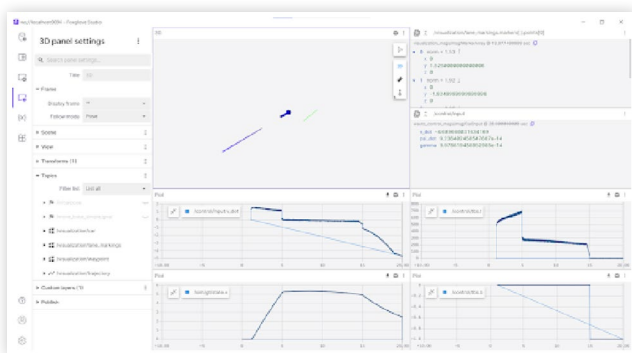
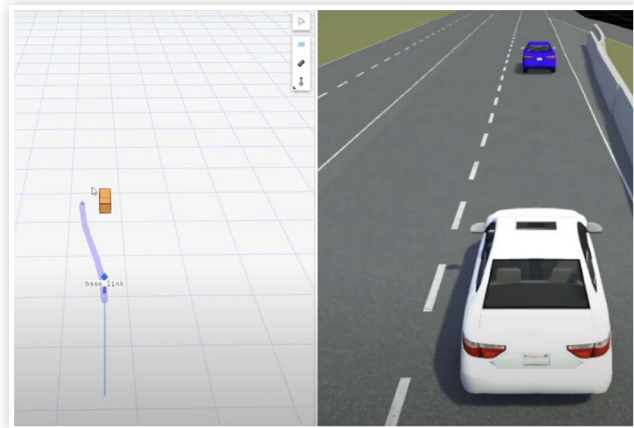


FIGURE 14 MATLAB graphs of low-level planner**FIGURE 15** Foxglove interface while testing in reality

validate the rest of the control stack in simulation, testing with various scenarios. To create different scenarios, the MathWorks tool RoadRunner was used [24]. This tool allowed for easy scenario variations and incorporated dynamic actors. MathWorks provided a simulation environment that resembled M-City and a mock HD map of the simulation environment. Furthermore, a ROS publisher was developed in MathWorks to mock the information that perception would provide to the control stack from the environment (i.e., cars/objects). This allowed for running the control stack similarly to how it operates in reality. Testing in simulation enabled multiple team members to work on testing the behavior of the mid-level planner in different scenarios, allowing the team to quickly identify and fix bugs in the algorithm.

Once the algorithms were tested and validated in simulation, the team moved back to testing in reality. The process for testing in reality involved testing the control stack level by level, starting from the lowest part. First, the low-level planner was tested, then the high-level planner was added, and finally, the high-level planner was tested with the rest of the control stack. Up to this point, perception data was mocked by simulating the ROS messages that perception would publish.

Once the control stack worked independently of perception, integration testing was performed. This allowed the team to further identify issues stemming from inconsistencies with perception data and fine-tune the relationship between the two nodes.

FIGURE 16 Simulation scenario of passing a moving vehicle in RoadRunner with Foxglove visualization on the left

Discussion

During the competition, our vehicle performed largely as expected but required some last-minute tuning to adapt to the environment. Specifically, several logic and gain adjustments in the control stack and modifications to certain aspects of the Controller Area Network (CAN) infrastructure were required.

The team demonstrated strong performance in the static challenges, securing a second-place overall ranking. In the dynamic challenge segment of the competition, our team excelled in the intersection challenge. This success can be attributed to the robust performance of both the low-level planner and the controller. They effectively guided the vehicle to accurately follow the planned path, resulting in the successful completion of the challenge.

However, a significant challenge arose in the competition concerning the perception stack. While the perception stack effectively detected obstacles, it encountered issues when relaying messages over the CAN bus. The CAN node was configured to operate with a single thread, and this limitation led to bus overloading. Unfortunately, the issue was not identified during earlier testing due to a lack of stress-testing in that specific area of the system. The competition's judging process heavily relied on the CAN bus for event scoring, and our inability to transmit perception information accurately through the bus impacted our overall performance. We attempted to reduce the number of messages being sent, but this strategy occasionally resulted in correct detections not being communicated, causing a loss of points. Another challenge that surfaced due to time constraints was the absence of thorough integration testing. This omission had a detrimental impact on our performance during the competition. While we had previously tested simple scenarios, the richer perception data at the competition overwhelmed the control stack, resulting in erratic vehicle behavior during the highway challenge.

Conclusion

A predominantly undergraduate team with limited funding, we approached the SAE AutoDrive Challenge II Competition with a startup-like mindset. Innovative thinking, agility, resourcefulness, and a commitment to continuous learning and improvement played pivotal roles in our performance and achievements in the competition. The most important aspect contributing to our success, in addition to team work, was an emphasis on the use of physical analogs and simulation for algorithm development and testing. By embracing early testing via analogs and simulation, the team managed to quickly create a minimum viable product that outperformed most of the competition. Looking ahead to Year 3, we will expand our current autonomy stack. We aim to tackle more advanced tasks, such as Level 4 autonomous driving, incorporating high-level path planning, dynamic rerouting in response to stationary and moving obstacles, and improved parking capabilities. Furthermore, we are planning to upgrade our mapping system to reduce its reliance on HD Map data. We will continue the same approach, one in which early testing via simulation will hopefully help us improve our autonomous vehicle.

References

1. SAE AutoDrive Challenge II. <https://www.sae.org/attend/student-events/autodrive-challenge-series2>. Accessed: 2023-10-21.
2. SAE Levels of Driving Automation. <https://www.sae.org/blog/sae-j3016-update>. Accessed: 2023-10-22.
3. M-City. <https://mcity.umich.edu/>. Accessed: 2023-10-21.
4. Wisconsin Autonomous. <https://wa.wisc.edu/>. Accessed: 2023-10-21.
5. Choi, H.S., Crump, C., Duriez, C., Elmquist, A. et al., "On the Use of Simulation in Robotics: Opportunities, Challenges, and Suggestions for Moving Forward," *Proceedings of the National Academy of Sciences* 118, no. 1 (2021).
6. Burnett, K., Qian, J., Xintong, D., Liu, L. et al., "Zeus: A System Description of the Two-Time Winner of the Collegiate Sae Autodrive Competition," *Journal of Field Robotics* 38, no. 1 (2021): 139-166.
7. Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D. et al., "Stanley: The Robot that Won the Darpa Grand Challenge," *Journal of Field Robotics* 23 (2006): 661-692.
8. Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H. et al., "Junior: The Stanford Entry in the Urban Challenge," *Journal of Field Robotics* 25, no. 9 (2008): 569-597.
9. Ziegler, J., Bender, P., Schreiber, M., Lategahn, H. et al., "Making Bertha Drive—An Autonomous Journey on a Historic Route," *IEEE Intelligent Transportation Systems Magazine* 6, no. 2 (2014): 8-20.
10. Gaber, H.A. et al., "The Role of Simulation in Development and Testing of Autonomous Vehicles," *ResearchGate* (2020).
11. AbdelHamed, A., Tewolde, G., and Kwo, J.. "Simulation Framework for Development and Testing of Autonomous Vehicles," in *2020 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, 1-6, 2020.
12. PyPi: Autonomy-Toolkit. <https://pypi.org/project/autonomy-toolkit/>. Accessed: 2023-10-21.
13. Elmquist, A., Young, A., Mahajan, I., Fahey, K., et al. "A Software Toolkit and Hardware Platform for Investigating and Comparing Robot Autonomy Algorithms in Simulation and Reality," *arXiv preprint arXiv:2206.06537*, 2022.
14. Wang, P., "Research on Comparison of Lidar and Camera in Autonomous Driving," *Journal of Physics: Conference Series* 2093, no. 1 (2022): 012032.
15. Fischler, M.A. and Bolles, R.C., "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM* 24, no. 6 (1981): 381-395.
16. Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 226-231, 1996.
17. Message Filters. http://wiki.ros.org/message_filters/ApproximateTime. Accessed: 2023-10-21.
18. Kuhn, H.W., "The Hungarian Method for the Assignment Problem," *Naval Research Logistics Quarterly* 2, no. 1-2 (1955): 83-97.
19. Dijkstra, E.W., "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik* 1, no. 1 (1959): 269-271.
20. Hart, P.E., Nilsson, N.J., and Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics* 4, no. 2 (1968): 100-107.
21. CVXPY: Convex Optimization Library. <https://www.cvxpy.org/>. Accessed: 2023-10-21.
22. OSQP (Operator Splitting Quadratic Program): Open-source numerical optimization solver. <https://www.cvxpy.org/>. Accessed: 2023-10-21.
23. Freire, V. and Xiangru, X., "Optimal Control for Kinematic Bicycle Model with Continuous-Time Safety Guarantees: A Sequential Second-Order Cone Programming Approach," *IEEE Robotics and Automation Letters* 7, no. 4 (2022): 11681-11688.
24. Mathworks RoadRunner. <https://www.mathworks.com/products/roadrunner.html>. Accessed: 2023-10-21.