aUToronto SAE AutoDrive Challenge II Concept Design Report Year 1

Justin Bauer, Amy Chen, Brian Chen, Yuan Shang Ding, Mustafa R. A. Khan, Sean Wu, Ziyi Xu, Yvonne Yang, Jiachen Zhou, Sandro Papais, Chude Qian, Jingxing Qian
University of Toronto, Apr 15, 2022

1 INTRODUCTION

After four successful years as part of the GM/SAE AutoDrive Challenge, the University of Toronto's self-driving car team has returned to participate in Round II of the challenge. More than 100 undergraduate and graduate students from diverse educational backgrounds contributed to the team's development efforts, building practical skills outside of the classroom in the process. Our solution is the product of 20 subteams, each focusing on a single component of the car, working together to meet and exceed competition requirements. We are hopeful that by the end of the four-year challenge, we can complete a fully autonomous Level 4 rideshare demonstration here in Toronto. For this year, we set our goal to source and build the perception system we will use for the next four years.

This year, the challenge mainly focuses on three scenarios: intersection with traffic lights, highway with road obstacles and speed signs, and dynamic object detection of cars, deers, and pedestrians. In the Traffic Light Challenge, teams are required to detect the traffic light state for each traffic light head, select the corresponding traffic light head to decide the vehicle's action based on the perception cart location, and report any objects in the intersection. In the Highway Challenge, teams are required to detect and report localization for obstacles on the roadway, speed signs on the side of the road, and the lane marking on the roadway. In the Dynamic Object Challenge, teams must report the position and velocity information for a car, a deer, and a pedestrian. Compared to all previous AutoDrive challenges that aUToronto has participated in, this year's challenge is more focused on the quantitative performance evaluation of our perception system, which has imposed a much more rigorous set of requirements on our system selection process.

In Year 1 of the Round II Challenge, teams were tasked with building a perception system that will serve as the foundation for the next three years of the competition. This meant we needed to build a mock-up of the perception system on a pushcart, instead of directly integrating it onto a vehicle. During the mechanical design phase, our team focused on building a hardware system that will both enable us to perform in the Year 1 challenge and be easily adaptable for the new 2022 Bolt EUV vehicle. Due to uncertainty in the computation system for future years' challenges, we have intentionally decided to opt for a conservative approach in our design by minimizing the use of software with platform dependency.

Figure 1: aUToronto's Perception Cart System.

This report will first describe the competition requirements and our sensor selection strategy based on these requirements. Next, we describe our perception cart design, including mechanical structure design, electrical system design, computation, and communication system. Afterward, we discuss our software architecture and the selected algorithms. Finally, we focus on describing our multi-sensor fusion system and introduce the newly overhauled multi-object tracking (MOT) system.

This report is organized as follows:

2 SENSOR SUITE SELECTION

Perception sensors are crucial for an autonomous vehicle to understand the world around it and to operate safely. Sensors are continually improving every year and unlocking improved perception and autonomy capabilities. Modern certified SAE Level 4 autonomous vehicles such as Waymo One, Baidu Robotaxi, or GM Cruise utilize four common sensors: Camera, LiDAR, Radar, and GNSS/INS localization module.

This section first analyzes the required perception coverage based on competition requirements. Next, we compare the available sensor options and justify our selection of sensors. Lastly, we provide a theoretical analysis of our sensor coverage and use it as a guide for our mechanical mounting design.

2.1 SENSOR COVERAGE REQUIREMENT

The coverage requirement analysis assumes an observation point on the top of a 2022 Bolt EUV vehicle, 1.8 meters above the ground. We also consider a mounting position similar to the 2022 Bolt EUV front bumper, which is 1.5 meters forward from the center of the 2022 Bolt EUV. In the sensor coverage analysis at MCity, we focus on analyzing the worst case scenario we can run into at different location of MCity.

Traffic Light Detection The traffic light detection task requires the ability to observe a traffic light up to 50 meters ahead of the ego system as illustrated in Figure 2. For the observing sensor, the field of view requirement in a worst-case scenario needs to cover the traffic light that is roughly 3 meters tall in front of the vehicle. The nominal coverage requires a horizontal field of view at 14° left and right and a +2° upward coverage.

Road Blockage Detection The task of roadblock detection is to enable the perception system to detect roadblocks up to 40 meters ahead of the ego system, with a horizontal spread of 60° as illustrated in Figure 3. As the barrels are short compared to the perception cart, vertical

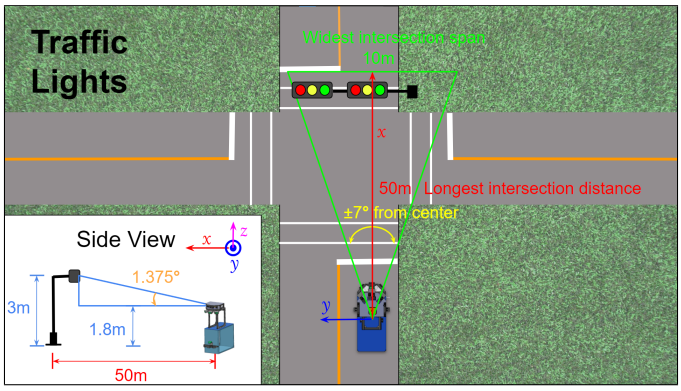

Figure 2: Traffic Light Worst Case Analysis

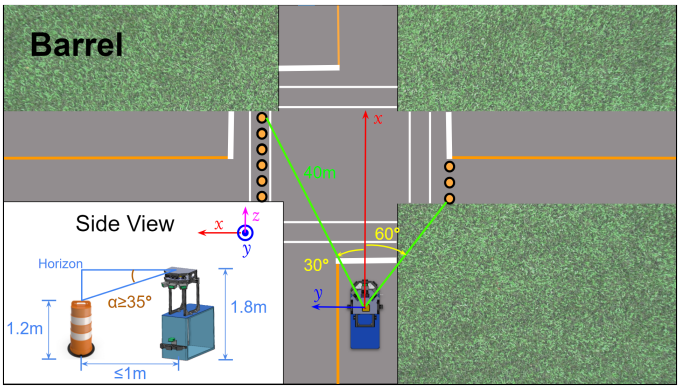

Figure 3: Road Blockage Worst Case Analysis

coverage at -40° from the top of our assumed observation point on top of the car is required.

Traffic Sign Detection The worst-case position of the traffic sign occurs in two conditions. The first condition is when the traffic sign is extremely close to the ego system, that will require a horizontal coverage of ±85° from left to right as illustrated in Figure 4. The second worst-case scenario is when the sign is extremely far ahead, namely 40 meters away, then the challenge is to recognize the signs' information correctly. A long-range camera lens will be needed with a minimal ±10° horizontal field of view to be able to see the far away sign.

Lane Marking Detection The lane marking detection task requires the robot to report the location of the stop lines and lane lines up to 10 meters ahead of the robot, in the field of view coverage. Additionally, the specifics of lane detection also require the robot to cover the lanes next to the ego system. Therefore, we deduce that the FOV coverage of the robot is ±60° horizontal and -30° vertically downwards.

Pedestrian, Deer, Vehicle Detection The worst-case scenario for the moving object occurs if an object is crossing right in front of the vehicle or if an object is crossing on the left-hand side of an intersection while our vehicle is pending a left-hand turn.

When an object, especially a small dynamic object such

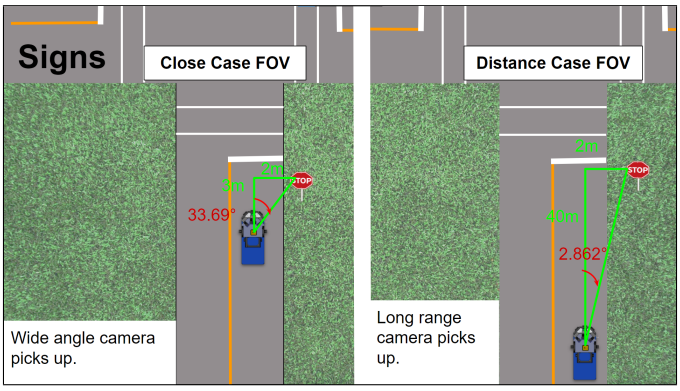

Figure 4: Speed Sign Detection Worst Case Analysis

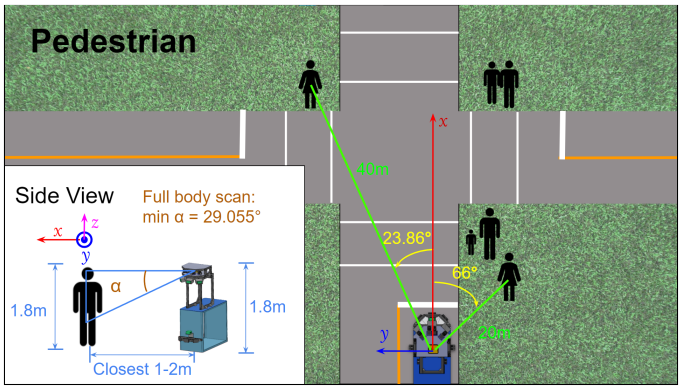

Figure 5: Dynamic Object Worst Case Analysis

as a child cross in front of our perception system, it might fall under the blind spot of the perception sensors on top of the vehicle. The horizontal field of view requirement is similar to the worst-case traffic sign scenario. The object can traverse from the right road shoulder to the left road shoulder. The deduced horizontal field of view is ±85°. The height of the child pedestrian dummy is similar to the barrel; therefore, we require a vertical field of view of -40° from the sensor stack to observe the child pedestrian dummy. To further minize the blind spot, we require the bumper sensor platform to be able to substitute in and provide coverage too. For the scenario where the pedestrian is crossing perpendicular to the ego system, the required distance is about 50 meters, with a horizontal field of view of -66° to -23°. Therefore, the final coverage requirement for the dynamic object handling is ±70° horizontal and -40° vertically, as illustrated in Figure 5.

Final Perception Coverage Requirement We can deduce the following perception coverage requirement from the aforementioned worst-case scenario analysis. Our system requires a horizontal perception coverage of ±80°, a vertical perception coverage of -40° (downwards) to +5° (upwards), and an object resolution in a camera image at 5 pixels per square inch. In the Year 1 challenge, the competition simplifies the requirement [1], and the maximum required observation coverage is 40 meters forward, with a horizontal spread of 60 degrees relative to the ego system. We configure our sensor system to maximize reusability to require minimal modification going into the Year 2 challenge.

2.2 SENSOR MODALITY OPTIONS

As sensor technology develops, newer sensor modalities have been introduced in recent years to assist autonomy systems better. For example, the thermal imaging cameras [2] have gained popularity to perceive objects with a heat signature on poorly lit night roads. Spinning radar [3] has shown unique advantages in long-term localization during adversarial weather conditions. 4D LiDAR (FMCW LiDAR) [4] has shown simplicity in obtaining object localization information and velocity information simultaneously. Event camera [5] has shown promise for high dynamic range capability. To better evaluate different sensor modalities, we evaluate different sensor's capabilities with weighted factors, as illustrated in Table 1. The sensor selection criteria was broken into four sections: sensor performance functionality, sensor algorithm functionality, sensor cost consideration, and additional principal limitations.

Table 1: Sensor Modality Comparison

Criteria	Weight	Camera	LIDAR	Radar	4D LiDAR	Thermal Camera	Event Camera	Spinning Radar	Ultrasonic Sensor
Principal Limitation									
Competition Restriction	999	0	1	0	-1	0	0	0	0
Import Export Restriction	999	0	0	0	0	-1	0	0	0
Typically Used Standalone	10	0	0	0	0	0	-1	0	0
Sensor Performance									
Provide 3D Object Localization	10	5	10	5	10	5	5	5	1
Provide Object Velocity	10	0	5	10	10	0	0	0	0
Provide Class Information	10	10	5	2	5	5	1	1	1
Provide Detection Range	10	10	8	9	8	5	5	8	1
Sensor Noise	10	10	8	5	10	8	5	5	1
Adversarial Environment Bonus	10	0	0	10	10	10	10	10	0
Sensor Algorithms									
Available Public Dataset	5	10	5	0	0	2	2	0	0
Typical Classification Algorithm Runtime Cost	10	10	5	5	5	10	2	2	0
Typical Localization Algorithm Runtime Cost	10	5	10	5	10	5	5	5	5
Utilized for ego localization	5	8	10	2	10	0	0	10	0
Sensor Cost									
Sensor Availability	50	10	10	10	1	0	5	1	10
Sensor Cost	50	9	8	9	2	5	8	2	10
Totals		1540	2484	1470	-119	-259	980	560	1090
Rank		2	1	3	7	8	5	6	4

Sensor Performance Criteria A typical sensor suite for an autonomous vehicle should provide localization information, classification information, and object motion characteristics for any object of interest. Therefore, we established the following criteria to evaluate a sensor's capability: If a sensor can provide object localization in 3D space directly, if a sensor can provide object velocity information directly, and if a sensor can provide object classification utilizing machine learning with little latency. Other important performance metrics for a sensor are the detection range for a given sensor and the noisiness of the information produced. Therefore, we look at the maximum detection range of a given sensor and its signal-to-noise ratio in the 40-meter range. Lastly, some sensors are more robust to adversarial weather (e.g. snowing) and lighting (poorly lit night street roads) conditions. We consider these capabilities as a bonus when evaluating our sensors.

Sensor Algorithms Criteria Another important aspect to consider when selecting sensors is the algorithms needed to maximize the sensors' performance. Here we established four evaluation criteria. First, public datasets are important if we want to train a machine learning based solution for our sensors. Next, typical algorithm runtime cost is also very important as we cannot assume unlimited resources on our autonomous system. Therefore, we opt to evaluate common classification algorithms and state estimation runtime costs. Finally, some sensors can also be used for ego vehicle localization in addition to perceiving other road agents and signage on the road.

Sensor Cost and Principal Limitations Criteria As a student design team, when selecting sensors, we also need to consider the funding. With a limited budget and funding sources an emphasis was placed on finding low cost sensor solutions. Finally, principal limitations were considered based on the technology available for the team. These limited availability products were penalized with a weight of 999 to signify the lack of viability of these options. While some sensors have lots of advanced features included, they are classified as dual-use products where we will have to go through the additional procedures to obtain clearance for using them. In addition, with COVID-19 still hindering manufacturing capability globally, some sensor manufacturers are suffering from chip shortages, resulting in lead time delays. Lastly, the competition imposed additional restrictions on LiDAR-related products, for which we are only allowed to use Cepton X60 as our LiDAR sensor.

Based on all the evaluation criteria mentioned above and detailed evaluation listed in Table 1, we opt to establish our perception sensor suite composed of cameras, LiDARs, and Radars. For localization of ego vehicle, we opt to use a standardized GNSS/INS system.

Furthermore, we define each sensor's primary responsibility based on their native output information. The LiDARs in our system will be responsible for providing localization information, the cameras, with the help of neural networks, will be responsible for providing the classification information, and the radars will be responsible for capturing the velocity information. In addition to their primary responsibility, each sensor can also provide additional information to mitigate localization or detection errors during sensor outage.

2.3 SENSOR SELECTION AND PLACEMENT

GNSS/INS Sensor Selection aUToronto has opted to continue collaborating with Applanix, a local Toronto company producing GNSS/INS sensor, for the GM/SAE AutoDrive Challenge II. Therefore, for the GNSS/INS system, we will be using Applanix's top-of-the-line GNSS/INS system, LV-125, with a dual receiver setup for optimal centimeter level accuracy localization and heading estimation.

To simplify the downstream calculation for the GNSS/INS system, we opt to place the GNSS/INS reference x-axis (pointing forward) overlapping with the x-axis of the perception cart (pointing the front of the perception cart), with only a z-axis translation and x-axis translation. This decision simplifies the mathematical operation when calculating the vehicle's acceleration and velocity and avoids rotational correction every time we obtain a sensor reading.

LiDAR Sensor Selection Introduced new in the GM/SAE

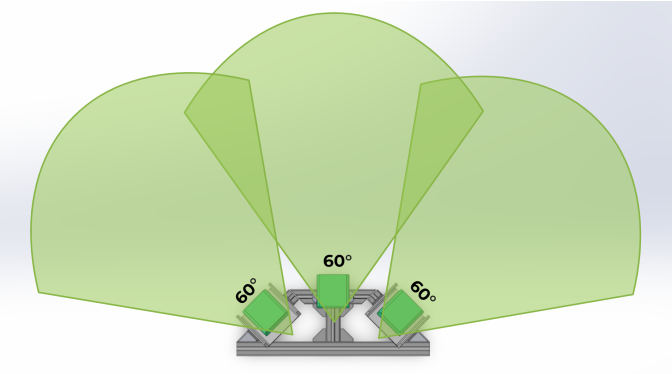

Figure 6: Proposed LiDAR placement top-down view

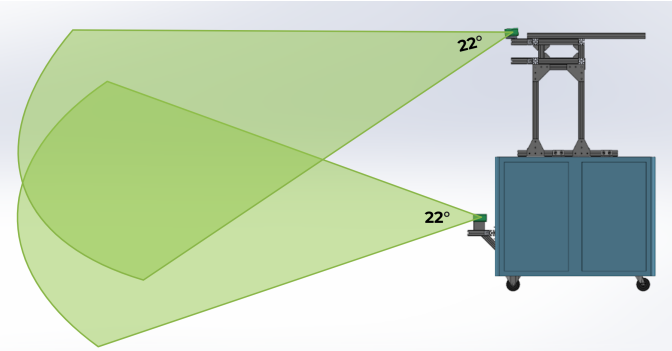

Figure 7: Proposed LiDAR placement side-view

AutoDrive Challenge II, Cepton is the exclusive LiDAR provider. For Year 1 competition, we are provided with the Cepton Vista P60 sensor option. Cepton Vista P60 provides a horizontal field of view span of 60 degrees, with a vertical field of view span of 22 degrees. A unique difference between Cepton LiDAR compared to other LiDAR products is that its beam pattern is uniformly distributed in the vertical space.

As solid state LiDARs typically provide a defined field of view in comparison to spinning LiDARs, we opt to establish our sensor suite with 4 Cepton Vista P60 sensors. Three of the four LiDARs are placed on top of the vehicle with a 10-degree pitch looking down to maximize coverage and density from 5 meters to 40 meters in front of the vehicle. The fourth LiDAR is placed on the bumper of the vehicle without any pitch offset to address concerns when objects are extremely close to the vehicle in the blind spot of the top three LiDARs and enhance the point cloud density straight ahead in front of the vehicle. The placement of the LiDARs is shown in Figure 6 and Figure 7. Note that, the top three LiDARs are placed with 45-degree yaw offset to maximize coverage while maintaining some overlaps to assist with alignment.

Camera Sensor Selection As part of aUToronto's effort to source suppliers locally, we have formed a collaboration and sponsorship relationship with LUCID Vision Labs, a Canadian machine vision camera manufacturer in Vancouver, BC. From past autonomous vehicle development experience, we value the importance of the IP67 camera solution, as they provide us with

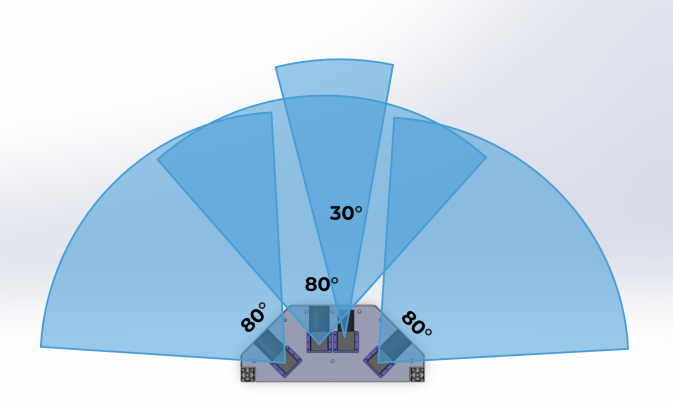

Figure 8: Proposed Camera Placement

additional camera protection and allow us to test our configuration in all weather conditions.

Within the IP67 camera lineup, we then decide on the CMOS Chipset we want to use. Currently, there are four different generations of Sony Pregius CMOS sensors supported. We focus more on CMOS's capability under varying lighting conditions when applying sensors on an autonomous vehicle. Compared to the other generation of CMOS sensors, 3rd generation CMOS from Sony Pregius provides the highest average dynamic range of 72dB, with the largest pixel size amongst all generations at 4.5 µm/pixel [6]. The third generation CMOS also has the best quantum efficiency allowing the best near inferred performance. A larger pixel size also adds the advantage of higher contrast [7] in the image and reduces the need for a high-resolution lens (which tends to be costly).

To narrow down the selection to a specific image sensor, we look at two important criteria: Sensor Resolution and Frame Rate. While a higher image resolution is a nice feature, sometimes too high resolution will cause unnecessary pre-processing overhead in the machine learning object detection pipeline. Therefore we set our search criteria for image resolution near 4k resolution, from 5 to 8 megapixels. A modern-day autonomous vehicle sensor system typically operates in the 10 to 30 Hz range, therefore, to allow versatility and future proof of our sensor selection, we use a sensor with a frame rate above 30 Hz. These criteria leave us with the final option of Atlas ATP071S with Sony IMX420 CMOS, which is capable of 7.1MP detection at a maximum frame rate of 70Hz.

Camera Lens Selection We can then calculate the field of view coverage for each lens configuration with the selected CMOS sensor. Table 2 compares the perception field of view with different lens options using the IMX420 CMOS (Type 1.1 Sensor) we selected above.

With redundancy requirements in mind, we propose a combination of wide-angle and long-range cameras to compose our sensor suite. Our final selected sensor suite comprises four cameras: three wide-angle lenses

Table 2: IMX420 with lens FOV analysis

Focal Length	HFOV	VFOV
8mm	78.19	73.74
12mm	56.89	53.13
16mm	44.22	41.11
25mm	29.15	26.99
35mm	21.04	19.46
50mm	14.81	13.69

at 8mm f/1.8 and one long-range lens at 16mm f/1.8. The three wide-angle lenses have a 45-degree yaw offset similar to our LiDAR setup. This configuration allows us to perform sensor fusion easier with LiDAR. Additionally, with nearly 90 degrees of horizontal coverage with the lens configuration, the cameras pointing right and left also provide sufficient overlapping region to guarantee redundancy straight ahead of our ego vehicle, mitigating risk caused by the degraded center camera. The long-range camera's 45-degree horizontal field of view allows us to perceive objects far ahead of us on the roadway. The long-range camera zooms in on the scene with the same resolution as the wide-angle lens configuration, allowing the neural network model to detect objects far ahead more confidently with more pixels per square foot. The camera placements and field-of-views are visualized in Figure 8.

Radar Selection To build a redundant safe autonomous vehicle perception system, we introduce radar for better object motion and trajectory observation and use it as a redundant sensor in sub-optimal weather conditions such as snowy days. With global chipset shortage and limited funding, we resort to using what is leftover from our other research project and chose Continental ARS430 as our primary radar sensor. With more funding in the future years, we intend to switch over to a more advanced platform such as the ARS540 lineup, which not only provides the radar target, but also ranges doppler information that can be used to infer object type and its 3D position. Additionally ARS540 have a much cleaner signal when the sensor itself is in motion compared to the ARS430, a feature we don't necessarily need in Year 1 Challenge, but a nice to have in future years.

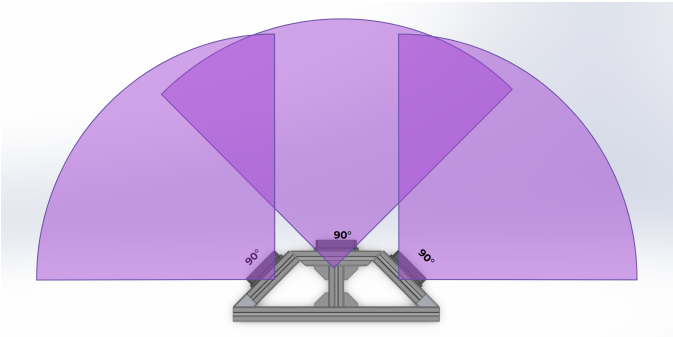

Figure 9: Proposed Radar Placement

Each ARS430 sensor provides 90-degree coverage horizontally and a maximum perception range of 60 meters ahead when operated in the NRS (near-field) mode. However, while the ARS430 also provides a far-field operation mode, the accuracy degrades drastically. Therefore, we opt only to use ARS430 in the NRS mode. In addition, while configuring the placement of the radar, we need to remember that radial velocity can be zero when the target motion is perpendicular to the radar's beam (zero isodop). Therefore, we use a triple radar configuration to maximize detection accuracy and simplify sensor fusion, each with 45-degree yaw offset similar to how we set up our LiDARs. With the yaw offset, we can detect an object's motion robustly even if the object's motion trajectory is perpendicular to the radar beam of the sensor. The radar placements and field-of-views are visualized in Figure 9.

3 PERCEPTION CART DESIGN

This section will describe our approach to designing the perception cart. We will approach the design from the mechanical perspective, compute and signal perspective, and electrical perspective.

3.1 MECHANICAL STRUCTURE DESIGN

3.1.1 Mechanical Design Principal The mechanical platform for AutoDrive Challenge II focuses on delivering a well-developed testing platform with characteristics such ease of assembly, feature expandability, precise sensor placement and weather protection in mind. The team's mechanical design partly refers to modern level 4 autonomous vehicle company's design language. While some features, such as industrial level smooth enclosure, are hard to achieve with a limited budget, the team's try to get our design as close as a deliverable product for the 2022 Bolt EUV. Additionally, while testing the hardware on a pushcart for year 1, our mechanical components are designed with reusability in mind. Therefore, we are hopeful that our design will be easily adapted to the new 2022 Bolt EUV vehicle. At the end of the section, we will show a proof of concept of what our sensor system will look like on the 2022 Bolt EUV.

3.1.2 Top Sensor Platform Design The top sensor platform is placed on top of the perception cart to match the height of the actual roof rack of the Bolt EUV. The primary function of the top sensor platform is to provide a stable design for the placement of all the primary sensors at their designated location and orientation, including four cameras, three LiDARs, two GPS antennas, one GNSS/INS receiver, and a few peripheral devices. The structure of both the top sensor platform and bumper sensor platform are made from 45mm x 45mm lightweight aluminum extrusions connected via

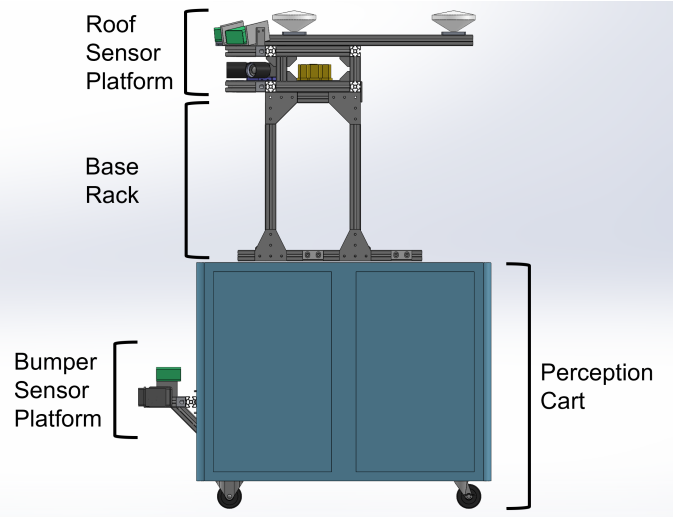

Figure 10: Overview of the two Perception Cart Sensor Platforms

various metal brackets and connector plates. The utilization of standard t-slot extrusion provides good modularity and customization characteristic that enable the team to fabricate the design at a lower cost with less manufacturing time.

As indicated in Figure 10, the entire top sensor structure can be separated into two sections, the base rack, and the roof sensor platform. The base rack is a simple rectangular framing structure designed to mount directly on the metal top surface of the perception cart. It provides additional height for roof sensor platform that simulates the rough vehicle height of the 2022 Bolt EUV. The roof sensor platform is mounted on top of the base rack with 3 connector plates, this modular design allows the quick removal of sensor platform for transportation and modification.

The roof sensor platform has sensors placed in 2 different levels, the main and upper level. The main level is designed to be more enclosed from different weather events such as rain or snow, thus protecting the more sensitive camera sensors and IMU device from external damage. Sensors at this level are mounted on a stable 1/8" thick sheet-metal surface and acrylic panels can be fitted to walls to isolate the sensor from extreme weather. Due to the characteristic of different sensors, the LiDAR sensors and GNSS/INS sensors are located at the open upper level. This arrangement gives the LiDAR sensors unblocked field of view of front and allows the platform to accommodate the length required for the dual receiver setup of the GNSS/INS sensors. Specifically, the two GNSS/INS antennas are placed 150 cm apart on a single extrusion beam that spans pass the back side of the sensor rack. The additional length of the antenna beam requires additional corner brackets to ensure the structural integrity.

Viewing the roof sensor platform from the top as illustrated in Figure 8 shows its polygon shape, where the front of the

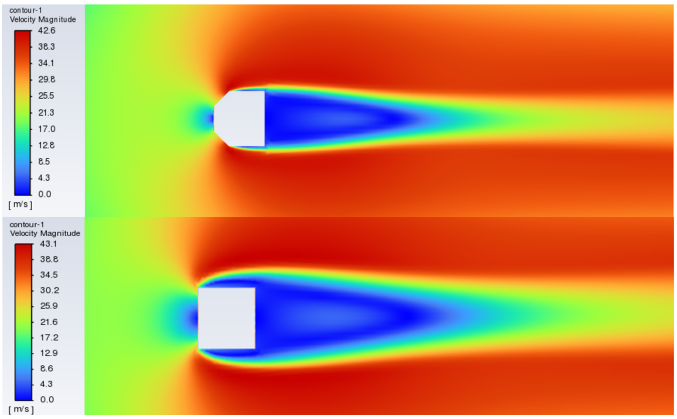

Figure 11: Velocity Magnitude Contour Profile comparison for the two design option of roof top sensor rack

rack has a flat side with two adjacent sides each a 45° to the front plane. This design provides aerodynamic benefit compare to a plain square shape design and allows the accommodation of the sensor's selected 45-degree yaw offset mentioned in the previous section. Furthermore, in the team's CFD study, the coefficient of drag for the selected sensor rack shape is 60% lower than the CD of a similar square shape design as illustrated in Figure 11, thus further justifying the chosen design.

3.1.3 Bumper Sensor Platform Design The bumper sensor platform is placed in the front of the perception cart to simulate a rough location and height of the front bumper on the actual Bolt EUV. The bumper rack is designed in an isosceles trapezoid shape with each leg of the shape at 45° to the base. This design enables the team to place 3 radar units in the bumper sensor rack with each of the 2 side radar units at 45° yaw offset for maximum coverage. Furthermore, the use of t-slot aluminum extrusion in the bumper platform enables the team to add an additional front-facing Cepton LiDAR on the bumper rack using standard rectangle tubes. The thin sheet-metal panel on the perception cart is structurally unstable for mounting the bumper rack, therefore the team added a lower diagonal support and a 6mm thick aluminum plate behind the perception cart panel to improve the design's structural stability.

3.1.4 Year 2 Proof of Concept The use of standard t-slot framing, removable brackets, and modular design structure paved an easy path for future expansion, modification, and transition of the exiting sensor platform to the 2022 Bolt EUV vehicle. To validate the design's transferability onto year 2's vehicle, the team overlaid the current mechanical design onto the Bolt EUV CAD model to illustrate the rough location and placement of the roof and bumper sensor platform. Figure 12 shows the render of our sensor system on the new Bolt EUV vehicle.

Figure 12: Rendering of sensor on a 2022 Bolt EUV

3.2 COMPUTE SIGNAL SYSTEM DESIGN

3.2.1 Compute System Design Due to the supply chain issue caused by COVID-19, the available choice for a compute platform is limited. When sourcing for the platform, we opt for a system with similar capability as we will get in the future with an Intel Xeon Server and an Intel Xe-HP Graphics Card. This year we opt for an Intel 10980xe CPU, with 18 CPU cores and 36 CPU threads. This 14nm Cascade Lake chipset has been proven reliable in daily operation use, and a large amount of threads allows the team to re-use the computer afterward as a virtualization server. In addition, we paired the CPU with a top-of-the-line Nvidia RTX 3090 GPU to facilitate algorithm testing and benchmarking. We intend to use this graphics card later as part of our multi GPU training cluster to minimize financial overhead. Additionally, based on early-stage testing released by the press, the Intel Xe-HP Graphics Card is expected to have a similar performance and computational capability at a near 40TFlops as an RTX 3090 graphics card. Finally, we paired the system with 128GB of RAM to fully utilize the CPU and GPU. The compute system alone sums up about 1300 wattage of peak power draw.

3.2.2 Network System Design One of the important lessons we learned in the last round of the AutoDrive Challenge I is the significance of a reliable network system on the vehicle. Previously our sensor system was driven through a single Netgear Gigabit Switch with a default configuration. However, with introducing a higher resolution imaging sensor and multiple LiDAR and Radar sensors, the existing networking capability is not sufficient to deliver information reliably and timely. Additionally, with the need for PTP time synchronization, we reconsidered the networking architecture for our sensor suite. The LUCID Vision Labs Camera utilizes 5GbE networking to maximize delivery bandwidth and guarantee low latency compared to the previous 1GbE camera we used in AutoDrive Challenge I. Therefore, a 40GbE capable switch from Ubiquiti is selected to minimize bandwidth limitation for the sensor suite and future proof for any high data bandwidth sensor. A designated Quad 10GbE

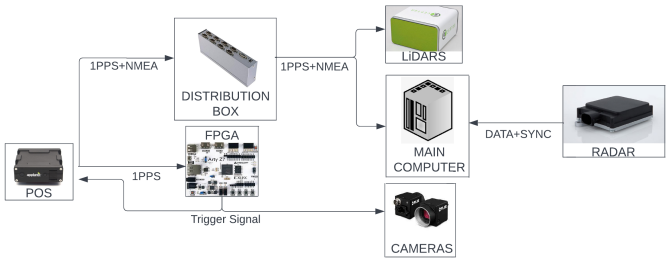

Figure 13: Time Synchronization Architecture

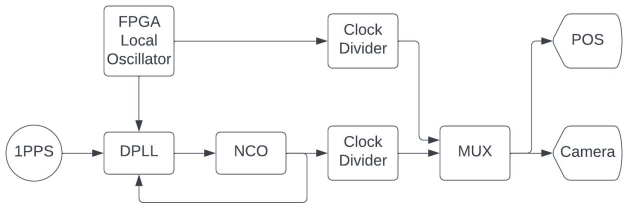

Figure 14: FPGA Trigger System Architecture

SFP+ networking card is selected on the computer in the perception cart to create the 40GbE Linkage to the networking switch using link aggregation.

Additionally, we noticed a conflict between the multicasting address of Cepton LiDARs and the LUCID Vision Labs Camera. We opt to isolate all Cepton LiDARs and provide them with a designated switch to address that issue. As Cepton LiDARs use 100M networking, we utilize a 1GbE network port on the computer to communicate to all Cepton devices.

3.2.3 Time Synchronization System Design With a total of 11 sensors to perceive a scene, the need for sensor synchronization is unprecedented as sensor fusion is highly reliant on an accurate timestamp for each sensor observation. The time synchronization problem for this project is addressed with multiple techniques to reach a correct interpretation and translation of timestamps as well as a simultaneous triggering for cameras.

We define the timestamp from our Applanix GNSS/INS system as the master clock to guarantee time accuracy. The timestamp from the Applanix GNSS/INS system utilizes GPS time from the satellite system. The Applanix module also has a clock locally that outputs the NMEA+PPS signal. The NMEA signal is an RS232-based serial signal following the standardized NMEA 0183 format. The 1Hz PPS is a 5V TTL pulse with less than one-second width and a sharply rising or abruptly falling edge that accurately repeats every second. Combing the 1Hz PPS signal with the GPRMC NMEA message allows us to achieve sub-microsecond accuracy. Figure 13 illustrates the time synchronization architecture.

Cameras: Synchronization Board The LUCID vision labs cameras support an external triggering configuration that uses a square signal with a specific duty cycle to control data acquisition and exposure time. Using the same trigger signal, all the cameras would trigger the start of the exposure and capture images simultaneously. To accurately timestamp the exact moment trigger started, we synchronize the 20Hz triggering signal to the 1Hz PPS signal, and we also back feed the trigger signal into the Applanix module. The Applanix module will also record the exact instance of the triggering event for further ease of time stamping images.

We used an FPGA Xilinx board to create a variable frequency square signal to work as a trigger signal for the cameras. The implementation uses phase-lock techniques to synchronize a numerically controlled oscillator (NCO) with 1PPS in phase and frequency. In addition, a discrete-time Phase-Locked Loop (DPLL) was implemented to achieve the signal alignment and set the NCO to a stable 10 MHz signal that could be used with clock dividers to create the required trigger signal. See Figure 14 for the architecture overview.

However, sometimes the 1Hz PPS signal might not be available due to sensor degradation (i.e., driving in a long tunnel) or indoor testing of the system; we utilize an additional multiplexer (MUX) to trigger the cameras using the internal clock on the FPGA.

LiDARs and PC: NMEA+PPS The Vista-P60 Cepton LiDAR and the Computer needs an external 1PPS signal and an NMEA message for time synchronization. While five end clients need the NMEA+PPS signal, the Applanix module only has 1 NMEA PPS output. We implemented an RS232 buffer chipset to duplicate the RS232 signal to all the sensor that needs the NMEA string to address this issue. We utilize a line buffer on the 5V TTL signal for the PPS signal. This line buffer prevents impedance issues introduced by attaching multiple clients to a single 5V TTL signal line.

A designated node that publishes the latest NMEA+PPS time reference into the ROS Graph is created to synchronize ROS Clock for cross node time tracking and ROS frame transformation calculation.

Radar: PTP On our host computer, GPSD is enabled to synchronize the clock on the PC to the standardized GPS time. The host computer also serves as a PTP time source for any device that utilizes PTP time synchronization. Radar utilizes the PC as its reference time source to stamp its observation.

3.3 ELECTRICAL SYSTEM DESIGN

To design the electrical system, we need to evaluate the load created by devices on the perception cart. Table 3 illustrates the power consumption for each sensor or computing equipment. These devices put our peak

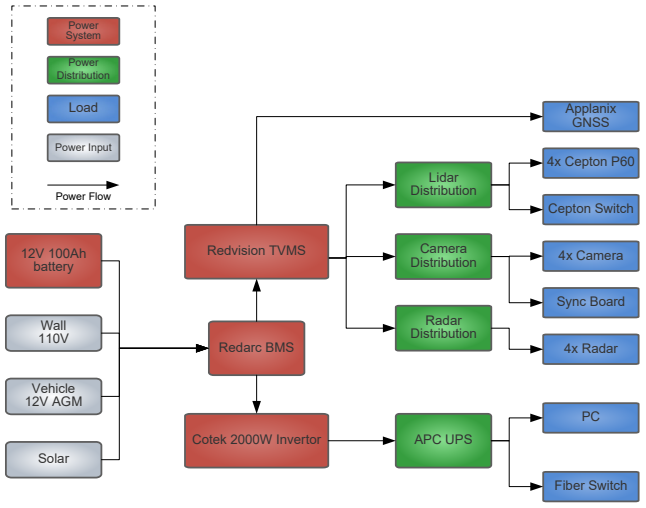

Figure 15: aUToronto Electrical System Diagram

operation power at 2000 watts. With the competition requirement that the perception cart needs to self-sustain for at least 20 minutes in the worst-case scenario, we will need a battery size of 100 amp hours to guarantee our perception can run de-tethered from line power.

As illustrated in Table 3, in our system, there are four types of voltage required by devices: 110V line power for the computer and network switch, automotive 12V (nominal 13.6V) for our sensors, regulated 12V for the sync board (nominal 12v), and 5V for our signal duplicator for NMEA PPS signal. For the compute server, we introduced a 2000 Watt DC inverter directly attached to the battery with six gauge cable to accommodate potential high current load. Before we power the computer, an additional commercial UPS is added to protect the computer further and provide urgent backup if the 100AH battery is fully drained, but we need to save some documents on the machine. Figure 15 illustrates the perception cart's power delivery system architecture.

In AutoDrive Challenge I, the electrical system was an aspect we did not pay too much attention to. Unfortunately, this oversight resulted in a couple of close calls during the testing of our autonomous vehicle. To prevent the same mistake from causing safety concerns, aUToronto now follows the industry standard

Table 3: Electrical Component Load Analysis

Load Device	Voltage Requirement	Power Requirement
PC	110V	1600W
GPS	12V Automotive	20W
Radar	12V Automotive	7W*4
Camera	12V Regulated	2.4W*4
LiDAR	12V Automotive	9W*4
Monitor	110V	200W
Misc	12V	20W
Total		~1800W

ipcwhma-a-620 [8] when creating wiring harnesses and designing the electrical system. Each sensor type shares a distribution block for the sensors, and each sensor is individually fused as illustrated in Figure 16.

We are fortunate to form a strategic collaboration with REDARC Electronics for AutoDrive Challenge II, who specialize in developing Overlanding vehicles' electronic control systems and in-vehicle dual battery chargers. Utilizing REDARC's Manager30 battery management system, we can plug in the perception cart when there is 110V power and utilize the battery if there is no power available for the perception cart. Manager30 also allows us to extend our run time by integrating solar panels into the system. In addition, the RedVision smart controller is a nice smart digitized solution for controlling the on and off of sensors and providing a summary of battery load and estimated runtime left. During the Ontario COVID-19 lockdown, we were able to control the RedVision smart controller using GPIO to turn on and off the perception cart remotely, allowing development to happen even though we were not physically with the perception cart.

4 SOFTWARE ARCHITECTURE

This section will review existing autonomous vehicle system architecture and object detection algorithms and select the best-fit algorithm to achieve the given task in the SAE AutoDrive Challenge II.

4.1 ARCHITECTURE OVERVIEW

The high-level software architecture is based on a modular and robust design inspired by other publicly available autonomous driving architectures. There are two major open-source autonomous vehicle software reference designs: Autoware.Auto and Baidu Apollo. Our

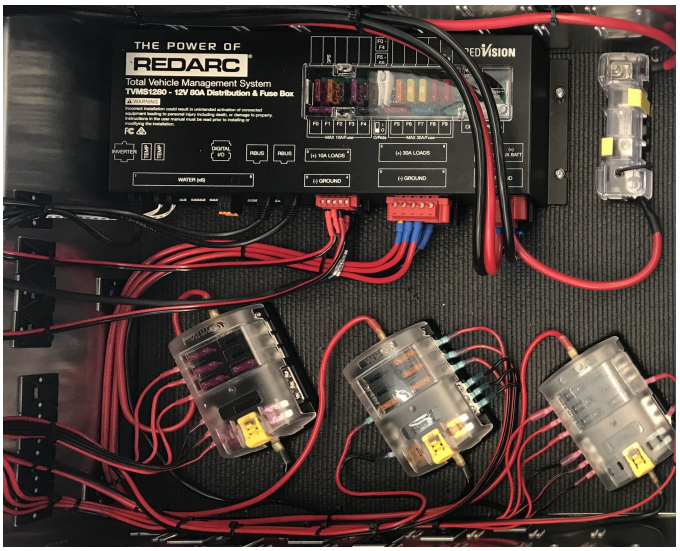

Figure 16: aUToronto System Wiring following industry standard practice, all power line individually fused

architecture follows a similar architecture decomposition as these architectures and shares many similar software algorithm choices.

Autoware.auto [9] claims to be an open-source autonomous vehicle system that provides a complete set of tooling, including perception, localization, planning, and control. The Autoware object detection pipeline uses both 2D cameras and 3D LiDAR point clouds. Unfortunately, there is no description regarding other sensor modalities, i.e., Radar for object detection.

Baidu's Apollo 6.0 [10] platform claims that it is built for the SAE Level 4 passenger vehicle, and its system has been operating on public urban roads. Unlike Autoware's ROS approach, Apollo Open Platform implements a proprietary software platform for communication between nodes. Apollo's capability has been demonstrated and verified on public roads in China and achieved many appraisals. Apollo's Platform support multiple sensor modalities, including cameras, LiDAR, and Radar. Apollo relies heavily on LiDAR for object detection and tracking and the RTK GPS sensor as its primary localization method. When GPS is not available, multiple sensor fusion was employed for better localization.

Figure 17 illustrates aUToronto AutoDrive Challenge II's system architecture design. Similar to Autoware.auto's design and Apollo's design, our system architecture can be decomposed into several software modules. The driver layer, illustrated as green and light green, handles interaction with the driver and outputs sensor observation for object detection. Our upstream processing node includes 2D Object Detection, 3D Object detection, Color Segmentation, and Lane Detection. The upstream nodes provide per sensor observations, which are fed into the sensor fusion stage of the pipeline. The sensor fusion provides four groups of observation for object tracker to track objects' position, velocity, and classification. Finally, this tracked information is fed into a behavior planner, generating high-level summarized information for the CAN driver to send to the competition scoring system. For lane detection and traffic light detection, as we already have ground-truth information from the site map, the behavior planner is responsible for directly handling the conversion of the information.

4.2 COMMUNICATION BACKEND

For a complex robotics system, communication is usually done via various remote procedure calls (RPC). Since 2011, Robot Operating System (ROS) has gained popularity amongst robotics developers as its standardization of communication interface. ROS enables nodes to communicate with each other through XMLRPC without worrying about implementing the communication layer on the robot. However, since then, technology has evolved majorly, and various new technology and larger sensor payloads put a higher constraint on the single host ROS communication system. Our research group noted significant performance degradation with tasks such as transporting large images or a large nondeterministic varying size array. Additionally, safety cannot be certified with the vanilla ROS1 system due to a potential single point failure with ROS Master. While some mitigation and workaround have been created, it is not as popular as its successor ROS2.

ROS2 has gained more and more popularity in recent years as it switched its communication backend to a data distribution service (DDS). DDS uses the "Interface Description Language (IDL)" as defined by the Object Management Group (OMG) for message definition and serialization. DDS has a much higher safety standard and is typically used in industrial safety-critical environments such as battleships, space systems, and flight systems. In 2021, Apex.ai obtained ISO26262 certification up to ASIL-D. In the published analysis of ROS2 and ROS1 comparison, it is noted that ROS2 has a better advantage in handling messages with larger payload sizes, and the versatility of QoS Policies allows a tunable configuration of communication priority within the system.

We opt to use ROS2 as our communication backend for our Round II system. The introduction of ROS2 allows us to address both the latency concern we observed in a typical ROS1 system and the safety concern caused by a single point of failure in the ROS 1 system. With ROS2 as our preferred middleware for our nodes, we can leverage the built-in function of node lifecycle and QoS to monitor the system's health and if a node has crashed.

4.3 2D OBJECT DETECTION

In the 2D Object Detection domain, YOLOv5 [11] and its variants [12] have achieved promising results. When looking at the existing SOTA 2D object detection algorithms, an important consideration is its implementation support for OpenVINO as our future compute platform will be Intel-based. With that constraint in mind, we compared the following family of algorithms: YOLOv3 [13], YOLOv5 [11], NanoDet [14], Haar+SVM.

When comparing potential model architectures, we opted to only benchmark a single task. This is because it would act as a standardized criteria to judge the merits of these models, allowing us to quickly iterate on architecture designs. We assumed that the benchmark would be a reasonable metric by which to judge the relative performance of the different models. Each trial was trained for 400 epochs with the Adam optimizer on a single NVIDIA RTX 3090 GPU. The learning rate was tuned using the PyTorch Lightning learning rate finder. We ran inference over ten images to benchmark the inference speed and took the average FPS on the GPU. To evaluate the architecture's performance, we analyzed

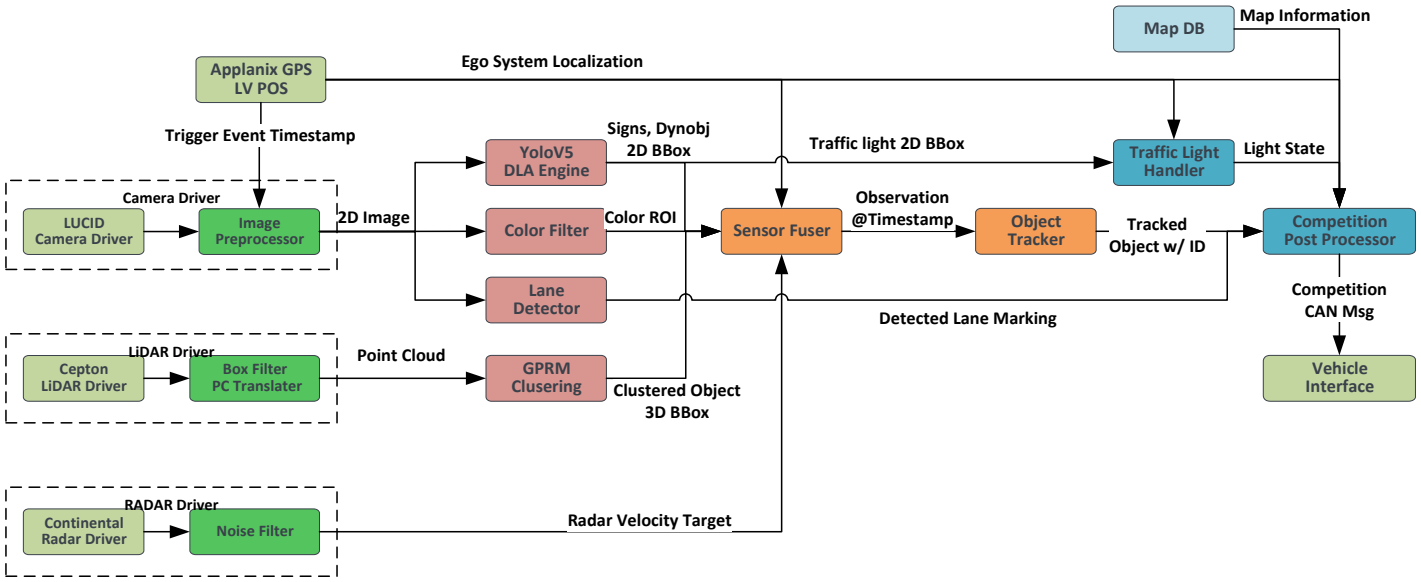

Figure 17: aUToronto System Architecture Diagram

Table 4: 2D Object Detection Inference performance

Architecture	Resolution	mAP	FPS
Nanodet-e0	512x288	0.2	239.5
Nanodet-e2	640x384	0.3	180.22
Nanodet-e4	1024x576	0.34	92
YOLOv3	640x640	0.38	81
YOLOv5S	640x640	0.32	120
YOLOv5M	640x640	0.36	86.06
YOLOv5L	640x640	0.39	90
YOLOv5L	1280x1280	0.527	35.1
YOLOv5L	1024x1024	0.472	41.66

metrics provided by the pycocotools API [15]. We took mAP as our primary metric. The COCO API [15] is calculated by taking the average AP of all classes for an IOU ranging between 0.5 and 0.95 at a step size of 0.05. This metric was chosen over the AP50 metric used in the PASCAL VOC challenge since it is more rigorous in evaluating IOU correctness. Table 4 captures the inference speed and accuracy across different model configurations.

We have to balance the model accuracy and inference speed to select the algorithm to use. In this year's competition, since we are mainly stationary with a slow-moving object, the emphasis is placed on the accuracy of the model rather than on the model's run time. Due to this emphasis, we considered our model's performance on large, standardized datasets such as ImageNet and a custom dataset we curated for the AutoDrive challenge. Furthermore, model size is another important consideration for autonomous vehicle applications involving embedded systems. However, we assume we have sufficient space. This means model size is weighed less compared to other factors that determine which architecture to adopt. Table 5 shows the weighted criteria used to select a model.

YOLOv5 stands out for its optimal model accuracy when tested on a complex dataset and its rapid inference or run time performance. The added benefit of choosing the YOLOv5 family of models is that they all share the same inference engine. This means we can fine-tune the model, its size, and its parameters to achieve optimal performance and rapid inference speed while avoiding repeated labour. For the competition, we opted for a small model size with a high image resolution. This assisted with traffic sign detection as these signs have similar shapes and so the model requires a higher resolution to distinguish between details such as digits. Similarly, for traffic light detection, we opted for a medium-sized model with a high image resolution to account for differences in traffic light appearances. Lastly, for the dynamic objects such as pedestrians and cars, we opted to use a large size model with low image resolution, as the distance within which detections are required is 40 meters.

Environment cues were used to detect barrels since there is no readily available dataset for barrels. Barrels and construction-related signage are typically painted with a bright orange color. A color filter is implemented to identify

Table 5: 2D Object Detection Algorithm Comparison

Criteria	Weight	YOLOv3	HAAR+SVM	YOLOv5	NanoDet
Principal Limitation					
Reported model performance	3	4	1	5	3
Reported model runtime	1	3	5	4	4
Tested model performance	3	4	1	4	3
Tested model runtime	2	3	5	3	5
Model size	1	3	5	3	4
Totals		36	26	40	36
Rank		2	4	1	2

this specific orange color and the information is used alongside the 3D object detection pipeline in the fusion stage to achieve detection.

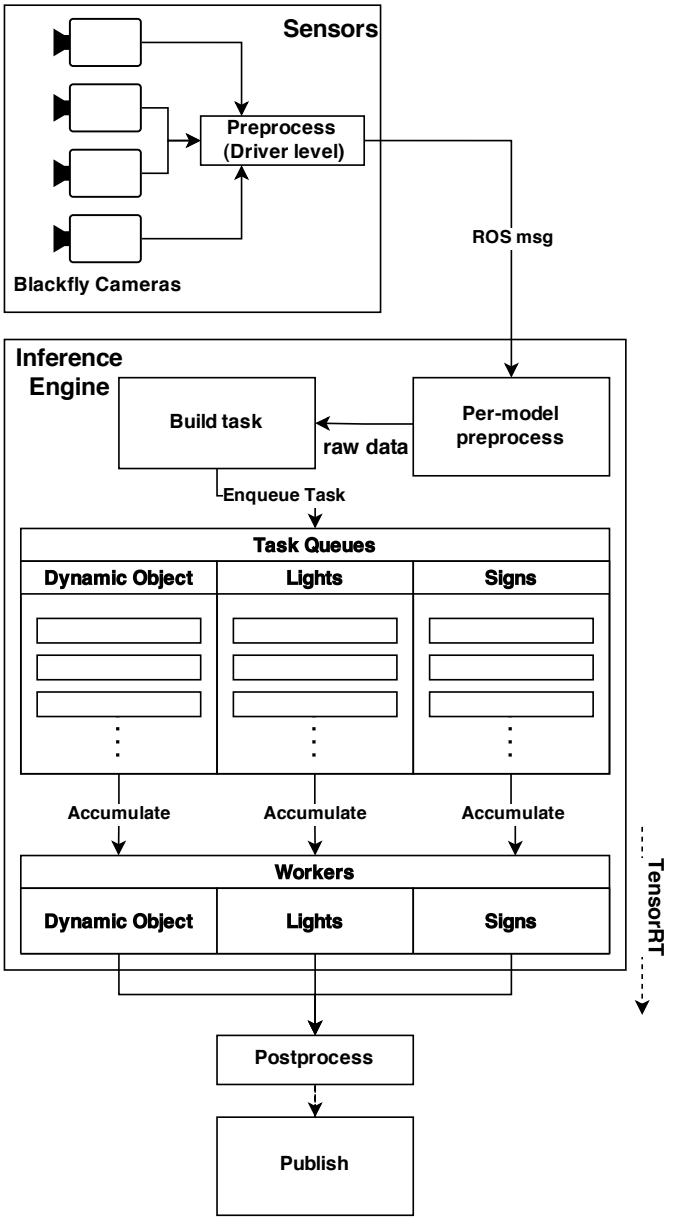

Figure 18: 2D Object Detection Inference Pipeline

Figure 18 illustrates our YOLOv5 inference engine integration using the worker-producer model. Pre-processing (resizing, cropping, image adjustments, etc.) was found to be surprisingly computationally expensive and we found that performing it on a hardware level as much as possible helped minimize latency. Inference is accelerated using Nvidia TensorRT, which enables us to optimize the inference engine built from our trained models to our specific hardware to maximize inference speed. For Year 2 and onwards, we expect this will be substituted with Intel OpenVINO hardware acceleration SDK. Due to variable system load and our camera's time synchronization we found it beneficial to leverage dynamic batching capabilities. This was integrated through a queue-and-accumulate model which makes the best effort to feed images in at an optimum batch size. Inference output (bounding boxes and classes for detected objects) are then put through post-processing non-maximum suppression (NMS) and re-scaling to produce usable detections for downstream subsystems. The 2D traffic signage and object detection pipelines all run at 15Hz end-to-end, satisfying the detection latency requirements set in the competition rules.

4.4 LANE DETECTION

There are three lane detection approaches: a computer vision-based approach, a probabilistic approach, and a deep learning-based approach. The lane detection task can also be tackled by considering the intensity of road surface points identified by LiDAR. However, we did not take this method into consideration because we noticed that the reflectivity of LiDAR point clouds was not distinct enough to consistently identify lane lines.

In the computer vision-based approach, the edge map is first generated and then a Hough transform is performed to extract lane segments. As illustrated in [16], the authors convert the image to grayscale, then use the Otsu thresholding method to create a binarized image. Next, the edge map is generated using the Canny edge detector, followed by the multi-scale Hough transform (MSHT), a variant of the standard Hough transform (SHT), where a down-sampling factor is introduced to parameterize lines. As illustrated in [17], the task is divided into a near-field view and a far-field view to handle curved lanes. A straight-line model represents the lane marking for the near-field of view while a higher-order curve represents the far-field of view. Higher-order curves are solved for using humanly imposed constraints and using least-squares curve fitting.

In the probabilistic approach such as [18], a neural network classifier is used to detect lane marking pixels using a bird's eye view image and cubic splines are fit to the detected pixels. RANSAC [19] is used to generate

Table 6: Lane Detection Algorithm Comparison

Criteria	Weight	Traditional CV Method	Probabilistic Method	Machine Learning Based
Algorithm Performance				
Detection Accuracy	10	8	9	9
Operation speed	10	10	8	2
Robustness after environment change	10	2	5	10
Implementation Specific				
Requires dataset	5	10	5	0
Requires GPU	20	10	5	0
Ease of implementation	10	5	10	10
Easily Tunable	5	10	5	8
Totals		550	470	350
Rank		1	2	3

one set of hypotheses of lane markings while another set of hypotheses is generated using a particle filter tracking algorithm. Probabilistic grouping sorts these hypotheses into left and right lane boundaries.

Deep learning methods can bring robustness that cannot be easily achieved using traditional methods. For example, the authors in [20] introduce the Spatial CNN (SCNN), where directional slice-by-slice convolutions are introduced to propagate information across rows and columns within a layer. This architecture was found to improve the detection of thin, long, and often occluded targets such as lane markings and traffic poles.

Candidate lane marking detection algorithms are evaluated on the following criteria: Robustness against changes in environment, operation speed, and detection accuracy as illustrated in Table 6. Furthermore, we considered the potential difficulty in implementing these candidate algorithms. While we are not constrained by computing in Year 1 of the competition, opting for a solution that can work reliably even if our compute power is stretched thin was a more future-proof decision. Therefore, for lane-detection, GPU-based machine learning methods were not preferred. Instead, in our Year 1 system architecture we opted to use a traditional, computer vision-based method. This is because we expect the lane markings at MCity will be relatively straight in the vast majority of locations and we hope our system can be re-used in the future.

A traditional computer vision-based approach shown in Figure 19 was used for lane detection as it works efficiently. This method has three major components: a perspective transform to convert the image to a bird's eye view, edge pair detection, and line marking candidate extraction. First, the camera input is converted to grayscale and transformed into a bird's eye view image using the homography we obtained during the calibration stage for the camera. More details about sensor extrinsics calibration are included in Section 5. Then, additional pre-processing of the image, including histogram equalization and a Sobel filter is applied to the image. Histogram equalization allows for more robust lane detection under varying lighting conditions and a Sobel filter extracts edges by approximating gradients and is used to smooth and suppress noise in the image.

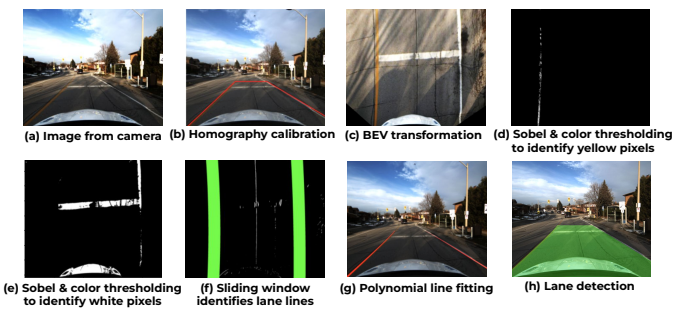

Figure 19: 2DOD Lane Detection Overview

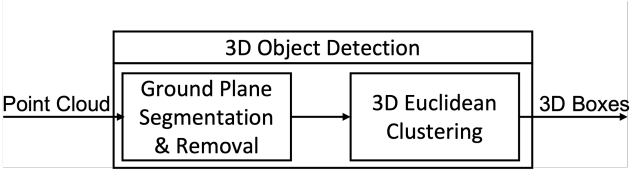

Figure 20: 3DOD System Overview

Next, color thresholding allows us to extract yellow and white colors in the driving scene, which are the typical, high saturation colors of lane lines on the road. Binary thresholding is then used to eliminate dull road colors. Finally, a sliding window scanning the image aggregates potential lane line pixels and allows us to distinguish between solid and dashed lane lines in the driving scene. Window sizes are selected based on prior knowledge of the width of standardized lane lines. We check the color continuity on the predicted lane marking to distinguish between solid and dashed lane markings.

For stop line detection, pairs of positive and negative edge pixels are found by checking the distance between pixels with positive and negative gradients. A probabilistic Hough transform is used for stop line candidate extraction to extract line segments. Line segments that are sufficiently close together are merged, as there may be occlusions covering the stop line sections.

Finally, the lane line and stop line segments are validated against our map which helps reject false-positive detections. The lane detection system also achieves a run-time performance of 15Hz.

4.5 3D OBJECT DETECTION

The 3D object detection module takes a real-time ROS2 message published by each Cepton LiDAR as an input and converts it into an Open3D [21] Point Cloud format. The core detection stack, as shown in Figure 20, consists of two major algorithm components: ground plane segmentation and 3D Euclidean clustering. The output is a list of K 3D bounding boxes, each with its corresponding object type and classification confidence, as denoted by $y_k = [x, y, z, l, w, h, \theta, \rho, type, confidence]^T$, where (x, y, z) are 3D object centroid location, (l, w, h) are length, width, and height representing 3D object dimensions, θ stands for object's yaw angle with respect to the ego perception cart, and ρ is the average intensity of all the LiDAR points enclosed in each bounding box.

Ground Plane Segmentation The purpose of ground plane segmentation is to remove LiDAR points that are projected on the ground while preserving objects of interest in the foreground. The benefits of doing this are two-fold. First, by removing ground points, the subsequent clustering-based detection submodule would have fewer points to process, cutting down computational time. Secondly and more importantly, removing these

Table 7: Ground Plane Segmentation Algorithm Comparison

Criteria	Weight	RANSAC	Himmelsbach's Method	GndNet
Limitation				
CUDA GPU Dependency	10	0	0	-1
In Need of 3D Dataset Annotations for Training	10	0	0	-1
Algorithm Implementation				
Ease of Implementation	2	6	3	-2
Tunability	1	2	6	0
Algorithm Performance				
Sensor Dependency	1	0	0	-2
Ground Plane Segmentation Performance	2	6	10	10
Ability to Handle Uneven Ground Segmentation	2	1	5	8
CPU Runtime	2	6	5	-1
Totals		40	52	8
Rank		2	1	3

Table 8: 3D Object Detection Algorithm Comparison

Criteria	Weight	Clustering	CenterPoint	PointPillars
Limitation				
CUDA GPU Dependency	10	0	-1	-1
In Need of 3D Dataset Annotations for Training	10	0	-1	-1
Algorithm Implementation				
Ease of Implementation	2	6	-3	-2
Tunability	1	2	1	0
Algorithm Performance				
Sensor Dependency	5	0	-2	-2
Object Classification Performance	4	6	10	10
Object Localization Performance	5	10	8	6
CPU Runtime	5	6	-1	-2
Totals		118	40	26
Rank		1	2	3

background points would reduce substantial noises to the clustering algorithm as the clusters are determined by the neighboring points based on 3D Euclidean distances. Having a cleaner foreground 3D representation is thus crucial to a faster and more robust object detection result.

Three candidate algorithms are considered as shown in Table 7. The baseline is a built-in segmentation function in Open3D [21], called RANSAC [19]. The proposed method follows the work [22]. In addition, GndNet [23], a machine learned (ML) based method is also investigated and compared against.

Table 7 indicates the preferable choice is using the algorithm proposed by Himmelsbach *et al.* [22], for its ability in handling slight unevenness in the ground plane while preserving a fast runtime ($< 10ms$). As a non-ML based method, this approach also does not require a human labeled dataset to be trained on, nor does it need to be run on a CUDA supported GPU.

3D Euclidean Clustering 3D object detection is often done either through a supervised ML-based approach or a traditional method based on a non-learning clustering algorithm. As can be seen from Table 8, the choice between these two boils down to hardware limitations, in terms of both LiDAR sensor modalities and computation resources.

Robust ML-based methods require training using a supervised learning way on extensive datasets with human-labeled annotations. Despite the fact that there are large public object detection and autonomous driving datasets available [24]–[27], they were all collected using spinning LiDARs. However, in this year's competition and the years following, the team is going to adopt Cepton's MMT®-based LiDARs which exhibit a unique point cloud beam pattern different from those produced by spinning LiDARs. Therefore, directly applying existing ML-based methods pre-trained on publicly available datasets faces an impediment to transfer learning. Dependent on sensor modality, ML-based methods are only attractive if they can be trained on a well-labeled 3D object detection dataset collected using a Cepton LiDAR. Though it technically can be achieved, fitting a labeling task into the perception's development timeline in the first year is not practical, let alone cost-effective.

Moreover, to be deployed on a real-time system, ML-based methods should be facilitated by proper deep learning acceleration hardware. Given the fact that CUDA-supported GPUs will no longer be allowed to use in the following years, the team examined the existing ML-based methods (e.g. CenterPoint [28] and PointPillars [29] as shown in the Pugh chart) and concluded that these methods, despite the state-of-the-art performance running on GPUs, show very poor runtime performance on CPUs due to expensive 3D convolution operations.

This led to the design choice of running clustering algorithm across point clouds from the Cepton LiDARs. A KdTree clustering approach was employed which was able to obtain near real-time performance on CPUs. The 3D object detection is achieved by identifying objects' point cloud representation in clusters based on the 3D Euclidean distances between a point and its neighboring points within a certain search radius. It uses prior object shapes and intensities of the LiDAR points as cues to infer object classes (e.g. car, pedestrian, traffic barrel, etc.). The output from the clustering algorithm is a list of bounding boxes that tightly enclose the points in each cluster. Based on the bounding box shape, each bounding box is represented by $y_k = [x, y, z, l, w, h, \theta, \rho, type, confidence]^T$ as aforementioned.

In summary, as described in Algorithm 1, our 3D object detection module consists of 2 major components: ground plane removal and 3D Euclidean Clustering.

1) The ground plane removal component consists of 3 steps based on the algorithm proposed by Himmelsbach *et al.* [22]: data partition, line fitting and ground point extraction. First, given the input point cloud, we represent the XY plane as a circle and partition the circle into segments. Each 3D point is mapped to a corresponding segment based on its x and y coordinates. Within each segment, the points are organized into bins based on their range with respect to the Cepton LiDAR. Second, we choose a point with the lowest z-coordinate in each bin and use the fitting method of *total-least-squares* (TLS)

Algorithm 1 3D Object Detection

Input: A LiDAR point cloud $p \in \mathbb{R}^{4\times n}$ with n points
Output: A list of K 3D bounding boxes $y : y_k = [x, y, z, l, w, h, \theta, \rho, type, confidence]^T$

$p \leftarrow ROStoOpen3d(p)$
$g \leftarrow GroundPlane(p)$
$\{p\} \leftarrow \{p\} - \{g\}$ // remove ground plane

for $i \leftarrow 1$ to n **do**
    clusters $\leftarrow EuclideanClustering(p_i)$
end for

for $k \leftarrow 1$ to $len(clusters)$ **do**
    $b_k = Get3DBoundingBox(cluster_k)$
    $type, conf. \leftarrow ClusterShapeInference(b_k)$
    $y_k = [b_k, type, conf.]^T$
end for

return y

with a root mean square error T_{RMSE} to extract lines. To model sloped terrain and to eliminate non-ground objects, only lines with a slope that is within an acceptable range are kept to represent the ground plane. Third, we classify a point to be a ground point if its distance to the closest line is within a specified $tolerance$. The ground points are removed for the subsequent 3D Euclidean Clustering.

2) The Euclidean Clustering component consists of 4 steps: data organization, 3D Euclidean clustering, bounding box filtering and object type inference. First, we use the Open3D library [21] to convert the 3D point cloud to a KdTree to achieve a total runtime complexity of $O(n\log(n))$ instead of $O(n^2)$ if the algorithm is run with unorganized 3D point cloud data. Second, we group neighbouring points into clusters based on their 3D Euclidean distances. For each cluster, we compute its corresponding bounding box $b_k = [x, y, z, l, w, h, \theta, \rho]^T$. Third, To optimize computational time for the object tracking stage, we filter out 3D bounding boxes that do not represent objects of interest based on their size and the z-coordinate (height) of the centroids. For example, in our context, a bounding box with a height of 4 meters would likely represent scenery objects and can be safely removed. Lastly, we use priors and average intensity of LiDAR points to assign each bounding box one of the following types: barrel, barricade, pedestrian, deer, car and traffic sign. We also compute the confidence for the type classification based on the priors and the size of the bounding box for the detected object. The final output of the 3D object detection module is a ROS2 message consisting of an array of bounding boxes, each specified with its centroid location, size, yaw angle, average intensity, object type and confidence.

The results are demonstrated in Figure 21. The bottom picture depicts the testing setup with all types of objects of interest. The top left figure shows the original LiDAR point cloud of the scene in a bird's eye view (BEV) (Note: for visualization purposes, the point clouds are truncated at 45 meters away and 20 meters to each side. Therefore, the additional traffic signs and traffic lights at the back are not displayed in the BEV.) As we can see, since a large portion of the point cloud is projected on the ground, it would take strenuous efforts to locate and detect the foreground objects even by visual inspection. From the clustering algorithm's perspective, the original point cloud input would bring too much noise to the clusters and thus hinder the performance. However, after applying our ground plane removal, we can see from the top right figure that almost all the ground plane points in the central region are segmented out, leaving 5 well-isolated point clusters with clear margins. Moreover, removing ground plane points that are not necessary for detection speeds up the clustering algorithm significantly, resulting in an overall runtime of less than $20ms$ of the 3DOD pipeline.

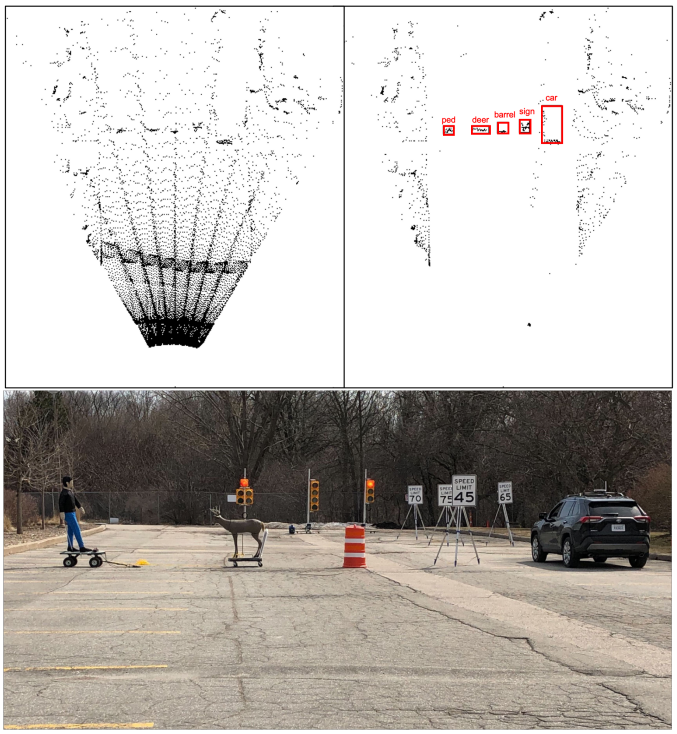

Figure 21: 3DOD Results Illustration. *Top Left*: original LiDAR point cloud; *Top Right*: detection operated on ground plane removed point cloud; *Bottom*: actual scene setup.

4.6 SENSOR FUSION AND OBJECT TRACKING

For details on the software architecture relating to sensor fusion and object tracking, refer to Section 5.

4.7 TRAFFIC LIGHT HANDLING PIPELINE

The traffic light pipeline is structured as a single ROS2

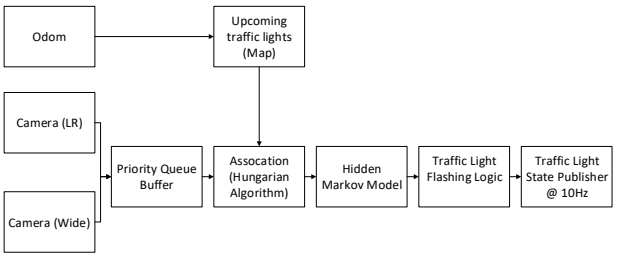

Figure 22: Traffic Light Pipeline

node as shown in Figure 22. Its primary purpose is to associate 2D traffic light bounding boxes from the perception stack with known traffic lights in the HD semantic map. Once traffic light detections are associated, the traffic light state is processed and sent downstream.

The traffic light pipeline receives GNSS data and uses data for two purposes. Firstly, it updates the current ego vehicle position. Secondly, the pipeline uses the precise ego vehicle position to query the HD semantic map and determine the upcoming traffic lights within the following two intersections. It also retrieves each traffic light's precise location, dimensions, type, and ID from the mapping data and projects them onto the 2D camera frames. Each identified traffic light is then fed into the association stage of the traffic light detection pipeline to be associated with their corresponding 2D detection bounding boxes.

The perception cart's center-wide and center-long-range cameras feed separate 2D bounding box detections to the traffic light pipeline. The detections from the cameras are received at 15 Hz and are stored in a priority queue ordered by received timestamps to fix any out-of-sync issues between the two cameras. After the priority queue reaches a specified size, the detection with the most recent timestamp is popped and processed whenever new detections are stored in the queue. This ensures that the priority queue does not continue growing or shrinking. Now we reach the association stage between M perceived traffic lights and N mapped lights previously calculated. An $M \times N$ cost matrix is set up with the Euclidean distance between centroids for the closest match, Intersection Over Union (IOU) for the largest overlapping area, and a comparison of the detected and mapped bounding box sizes to ensure, say, a 3-element light does not associate to a 4-element. This cost matrix is used by a Hungarian algorithm to associate the 2D bounding box detections with the traffic lights retrieved from the HD semantic map. After association, each 2D detection's associated light state is fed into a Hidden Markov Model (HMM) with its previous traffic light state confidences. The HMM filters the traffic light state and addresses the scenario where the upstream perception pipeline may provide incorrect traffic light state labels for a few camera frames. The traffic light pipeline appends the filtered traffic light state to the saved history of light states to determine whether the light was flashing. If the number of traffic light state transitions exceeds a specified threshold within a sliding window, the traffic light is assigned a flashing state. The final stage of the traffic light pipeline publishes the associated traffic light and its processed light state.

4.8 BEHAVIOR PLANNING PIPELINE

This year, the behavior planner module is responsible for three main tasks: 1) tracking and calculating relative orientations of objects, 2) determining the relative lane of detected objects, and 3) handling speed signs to output the next observed speed limit.

Task 1 is relatively straightforward and consists simply of a tracker which keeps track of the velocity of each object to determine its dynamic property (whether it is moving in the same direction as the ego, opposite direction, has moved but is currently stopped, etc.). The relative direction of travel to true north is approximated by the UTM grid north, which is accurate to 1-2 degrees. The behavior planner also filters objects within 60 degrees of the vehicle's line of sight and less than 40 meters away from it.

The relative lane of objects is determined using the known positions of the ego vehicle and detected obstacles to query the HD semantic map for their associated roads. The behavior planner uses the provided road IDs from the map queries to systematically test the relative lane cases (host lane, left lane, right lane, left lane shoulder, right lane shoulder, off-road, unknown, overhead). The HD semantic map consists of a graph structure where each road can have associated sibling, successor, and predecessor roads. If the object has no associated road, it is on the left shoulder, right shoulder, or off-road. We perform a depth-limited breadth-first traversal (BFS) from the ego vehicle's road to handle the host, left, and right lane cases. If we discover the obstacle's road during BFS, the obstacle and ego vehicle are either on the same road (host lane) or sibling roads (left or right lane). We determine the depth limit from testing to ensure we do not search the entire road map graph structure. If BFS fails to find the obstacle's road, we test for left bound or right bound cross-traffic cases by comparing the obstacle and ego vehicle heading. If the object's relative lane is none of the previous cases, we declare it to be the opposing case by the process of elimination.

The behavior planner receives upstream traffic sign detections and processes them to identify the next observed speed limit. Each sign detection includes the type, position, and value (for speed signs). The next observed speed limit is found by calculating the Euclidean distance of each sign and picking the closest one. The behavior planner checks each sign position to see which signs are within the competition requirements

of 40 meters and 60 degrees of the ego vehicle. For each sign within range, the behavior planner sends the calculated relative position, type, and value upstream to the CAN bus.

5 SENSOR FUSION STRATEGY

In this section, we describe our strategy for fusing the information provided by the 11 sensors onboard the perception cart. We first discuss our strategies for intrinsic and extrinsic calibration to achieve consistent measurements between the sensors. Next we discuss our sensor fusion pipeline and object tracking pipeline that combine measurements from the sensors to output object-level geometric and dynamic information.

5.1 SENSOR CALIBRATION

Sensor Calibration is the process of determining the intrinsics (internal sensor parameter) and extrinsics (sensor mounting position and orientation (pose) with respect to another sensor) of the onboard sensors. Good sensor calibration is crucial for us to correctly fuse information from different sensors as the calibration allows us to convert observations from one sensor reference frame to another.

5.1.1 Extrinsics Tree The extrinsics tree describes the relationship between coordinate frames in a tree structure buffered in time. A properly setup extrinsics tree should have at least one edge connecting to each of the sensor reference frame node, where the edge is an $\mathbf{SE}(3)$ matrix describing the frame transformation. Figure 23 illustrates the coordinate frame transform relationship of the onboard sensors and how we intend to obtain the calibration for each sensor pair.

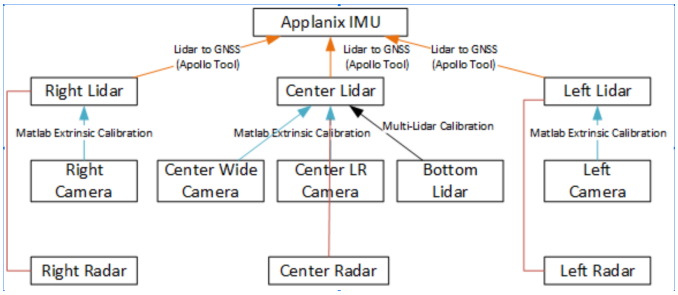

Figure 23: aUToronto Sensor System Extrinsics Tree

We define the following notation: Let the edge ${}^{F_1}\mathbf{T}^{F_2} \in \mathbf{SE}(3)$ be the pose of frame F_2 relative to F_1 which can be equivalently interpreted as a coordinate transformation from frame F_2 to F_1. If we have a point ${}^{CL}\bar{\mathbf{P}}^{A}_{CL}$ in the center top LiDAR frame and we would like to convert it into world frame ${}^{UTM}\bar{\mathbf{P}}^{A}_{UTM}$, we will do:

$$ {}^{UTM}\bar{\mathbf{P}}^{A}_{UTM} = {}^{UTM}\mathbf{T}^{INS} \cdot {}^{INS}\mathbf{T}^{CL} \cdot {}^{LR}\bar{\mathbf{P}}^{A}_{LR} $$

of which ${}^{INS}\mathbf{T}^{CL}$ is the edge from center top LiDAR frame to GNSS/INS reference frame, and ${}^{UTM}\mathbf{T}^{INS}$ is the odometry information that describes the vehicle position in UTM coordinates.

The following sections will describe how we calibrate the different sensors.

5.1.2 Camera Intrinsics Calibration We perform camera intrinsics calibration to address lens distortions. There are two common types of distortion: radial distortion and tangential distortion. Radial distortion causes straight lines to be curved on the image and progressively gets worse towards the side of the image. Mathematically, this phenomenon can be described with the following equations:

$$ x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) $$
$$ y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) $$

where r is the radial distance of the pixel from the image center.

Tangential distortion causes tilts in the image plane, which can be described with the following equations:

$$ x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)] $$
$$ y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]. $$

Hence, we need to solve for 5 parameters to model the radial and the tangential distortions: $D = [k_1 \quad k_2 \quad k_3 \quad p_1 \quad p_2]$. Additionally, based on camera lens selection, we can obtain a camera intrinsics matrix described with focal length f_x, f_y and camera optical centers c_x, c_y. These creates a camera projection matrix defined as follow:

$$ K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} $$

Utilizing a checkerboard test pattern (which contains known corner position), we solve these unknown values and derive the camera matrix as well as the distortion factors using MathWork's Camera Calibration App. With these variables, we can rectify the images and calculate pixel-wise correlations of a 3D object on the 2D image plane.

5.1.3 Camera Homography Calibration While we cannot transform a point from the 2D image plane to a 3D world, we could associate the pixels on the 2D image plane to a 2D plane (such as road surface) in the 3D world via a homography matrix. To estimate this homography matrix, we place multiple landmarks on a section of a flat ground surface and measure the location of each landmark with respect to a reference point (physical center) on the perception cart. Next, an image of the landmarks is taken using each camera, and the pixel locations of each landmark are identified. Finally, the homography matrix is estimated by simply minimizing the reprojection loss (i.e., transforming the pixel locations of each landmark and comparing them to the measured locations), which is achieve with OpenCV. These estimated homography matrices are then used in the stop line and lane detection algorithms.

5.1.4 Camera-LiDAR Calibration LiDAR-camera calibration was performed using MathWork's LiDAR Camera Calibrator App to determine the transformation matrix between the a LiDAR-camera pair that can be used for more robust sensor fusion. Based on the design of the perception cart, LiDAR-camera calibration was performed on the following LiDAR-camera pairs:

- Top center LiDAR with center wide-range camera
- Top center LiDAR with center long-range camera
- Top right LiDAR with right camera
- Top left LiDAR with left camera

Data from both the LiDAR and camera were captured simultaneously using a hardware synchronizer. To avoid inaccurate checkerboard detection and motion blur effects, the checkerboard was held far from the body and moved slowly [30]. Other items in the vicinity were also removed to ensure they were not clustered with the checkerboard [30]. The collected LiDAR-camera pairs

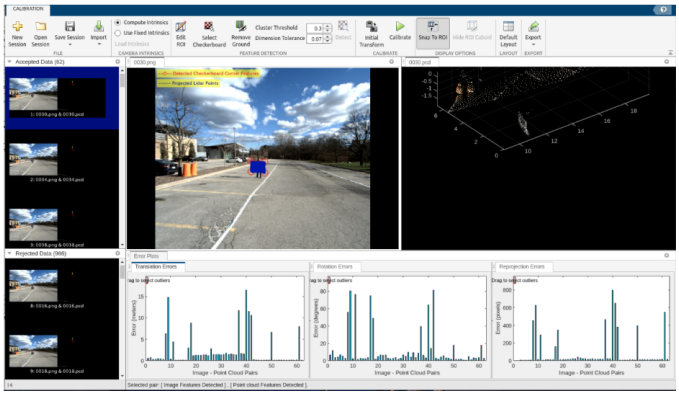

Figure 24: Calibration results from the LiDAR-Camera Calibrator App. The LiDAR projection on the image for one of the good LiDAR-camera pairs is shown. The LiDAR-camera pairs with high errors were later removed.

were loaded in the LiDAR-Camera Calibrator App, which automatically calculates the rigid transformation matrix between the LiDAR and camera. The app first estimates the checkerboard corners and board dimensions from the camera data using the camera intrinsic information and the measured size of the squares and padding . The app then attempts to detect the checkerboard plane from the LiDAR data using the board dimensions. The transformation matrix between the two sensors is then calculated from these detections based on the algorithm proposed by Zhou et al. in "Automatic Extrinsic Calibration of a Camera and a 3D LiDAR Using Line and Plane Correspondences" [31].

For better calibration results, settings such as the Euclidean distance threshold at which two adjacent points would be clustered together and the dimension tolerance of the rectangular plane were adjusted. It was found that decreasing the cluster threshold from the default

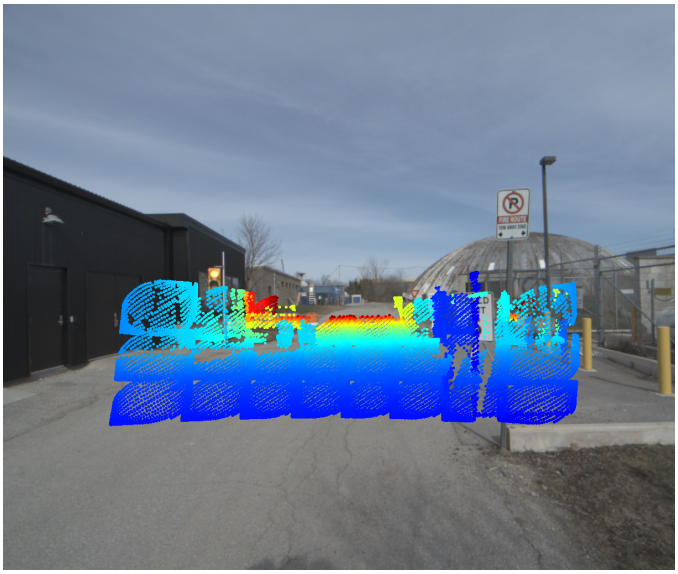

Figure 25: Overlay of LiDAR point cloud onto the 2D image once we have obtained camera intrinsics matrix and LiDAR to Camera Extrinsics.

0.5m to around 0.3m generally allowed for a greater number of detections due to the relatively high resolution of the LiDAR. On the other hand, there were more detections when the dimension tolerance was increased from the default 0.05m to around 0.07m to account for checkerboard measurement errors. The region of interest also had to be adjusted in some instances to account for the entire region where the checkerboard was located during data collection.

To verify the calibration results, the projection of the LiDAR points on the image based on the resulting transformation matrix was visualized. The translation error, rotation error, and pixel difference of the centroid coordinates between the projected LiDAR checkerboard plane and the corresponding checkerboard in the camera image were calculated and plotted, as shown in Figure 24. The LiDAR-camera pairs with high errors were removed, and the transformation matrix was recalculated with the remaining data.

The process of LiDAR-camera calibration using a hardware synchronizer to collect LiDAR-camera pairs and MathWorks' LiDAR-Camera Calibrator App allowed for an efficient way to perform LiDAR-camera calibration. In the future, the team would like to incorporate additional tools such as those provided by MathWorks' LiDAR Toolbox to automate and streamline calibration, including LiDAR-LiDAR calibration, LiDAR-INS calibration, and LiDAR-radar calibration. Figure 25 illustrates once we have obtained the camera LiDAR extrinsics matrix, camera intrinsics mtarix we are able to project any arbitrary point cloud point onto the 2D image.

5.1.5 LiDAR-LiDAR Calibration To obtain the LiDAR to LiDAR extrinsics, we utilize iterative closest point

(ICP) to match the point clouds from two LiDARs. The point-to-point ICP problem can be defined as following: We need to find a transformation $\mathbf{T}$ that minimize the objective function $E(\mathbf{T})$ where

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \|\mathbf{p} - \mathbf{T}\mathbf{q}\|^2$$

of which $\mathcal{K} = \{(\mathbf{p}, \mathbf{q})\}$ and $\mathbf{P}, \mathbf{Q}$ are point clouds. [32] In our operation, we utilize Open3D's function "registration ICP" to calculate the optimal extrinsics transform matrix. Figure 26 illustrates the aligned point cloud from 4 LiDARs.

5.1.6 Trajectory-based Calibration To obtain the transformation between LiDAR-INS and Radar-LiDAR, we utilize the idea of continuous-time trajectory matching. To obtain the Radar to Radar extrinsic matrix, we set our perception cart to be stationary and observe the RDI target in the scene. Since there should only be one object moving in the observable field, we can compare the estimated trajectory of the two and perform a point-to-point ICP matching between the trajectory of the two radars. This ICP-generated matrix describes the relationship between the two radars.

To abstract the LiDAR trajectory of an object, we utilize the clustering node from our 3D object detection to generate a single point, the centroid, for the object. Next, similar ICP matching as radar to radar calibration is performed to find the extrinsic matrix from the LiDAR to Radar.

To obtain the LiDAR to INS extrinsic matrix, we move the perception cart system to obtain the perception cart's trajectory from the INS sensor directly. Utilizing LiDAR odometry, we can also obtain the perception cart's trajectory via LiDAR as an observer. Utilizing ICP matching, we are able to find the extrinsic matrix between LiDAR and INS.

5.2 SENSOR FUSION AND OBJECT TRACKING SOFTWARE ARCHITECTURE

Sensor fusion and object tracking require associating the current measurements to the previously tracked objects to update the estimated states for all observed objects in the scene. The simplest approaches associate observations extracted from the current LiDAR scan to the tracked objects. Affinity scores between the tracked objects and the observations can be computed using 3D Intersection of Union (IoU) or negative Euclidean distances. The optimal association problem or minimum weight bipartite matching is formed as a binary linear integer program and can be solved by greedy nearest neighbour or Hungarian algorithm variants such as the modified Jonker-Volgenant method [33]. After association, unmatched objects are removed and unmatched observations are used to spawn new objects. Finally, a 3D Kalman filter updates the tracked object states using associated observations. This was cohesively formalized by AB3DMOT [34].

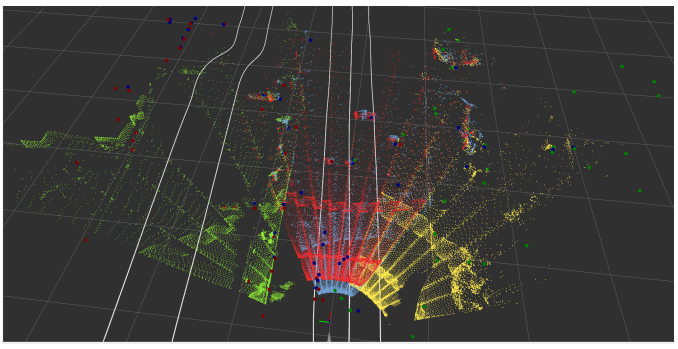

Figure 26: LiDAR-to-LiDAR ICP based alignment, Green: Top Right LiDAR, Red: Top Center LiDAR, Yellow: Top Left LiDAR, Blue: Bottom Center LiDAR.

Many tracking algorithms were reviewed with a focus on those that perform sensor fusion and three more are mentioned here. Prob3DMOT-M [35] used fusion of 2D and 3D features with Mahalanobis distance to produce affinity scores and learned confidence scored for initializing new tracks. EagerMOT [36] implemented a two-stage association approach by maintaining 2D MOT tracks and 3D MOT tracks in parallel. AutoTrack [37], an algorithm developed for handling sparse MOT tracking during the SAE AutoDrive Challenge, stands out for its lightweight dependencies and fast operation speed. These methods were taken as inspiration for our tracking algorithm.

The solutions mentioned above do not provide a straightforward tracking solution integrating cameras, LiDARs, and Radars. Therefore, we determined to build a new sensor fusion and tracking system to accommodate the need for object tracking for the AutoDrive Challenge. Our AutoDrive Challenge II tracker uses a similar idea presented in AutoTrack [37] and EagerMOT [36], consisting of two main components: Sensor Fusion, where all the instantaneous sensor observations within each sensor group are aggregated; and Tracker, where the aggregated observations from each sensor group are used to maintain state estimates for our global object tracks.

5.2.1 Sensor Fusion Our perception cart is equipped with 11 sensors. These sensors include 3 wide-angle cameras (WAC), 1 long-range camera (LRC), 4 LiDARs, and 3 Radar. We perform sensor fusion by splitting these sensors into 4 groups, as seen in Table 9. The goal for the grouping is to ensure each observation we pass down to the Tracker consists of measurements from

Table 9: Sensor Fusion Groups

Groups	LiDAR Sensors	Camera Sensors	Radar Sensors
Left	Left LiDAR	Left WAC	Left Radar
Center Top	Center Top LiDAR	Center WAC, Center LRC	Center Radar
Center Bottom	Center Bottom LiDAR	Center WAC, Center LRC	Center Radar
Right	Right LiDAR	Right WAC	Right Radar

Table 10: Fusion Message Filtering Comparison

Criteria	Weight	Message Filter Libraries	Custom Message Fusion
Development Costs			
Time to develop	2	1	-1
Robustness throughout development	2	1	-1
Input Streams			
Number of input streams	5	-1	1
Types of input streams	5	-1	1
Flexibility			
Ability to add new input streams	1	-1	1
Software Complexity			
Difficulty to change algorithms	1	1	-1
Runtime efficiency			
Compute efficiency with limited resources	3	1	0
Totals		-3	6
Rank		2	1

different sensor modalities, including camera, LiDAR, and Radar. For the center group, both the wide-angle and long-range cameras are included. This fusion step shall not be confused with the association step in the Tracker, as it is an aggregation of information solely. Each sensor group is handled by a designated fusion node to achieve parallelism in the operation and minimize pipeline latency.

In order to fuse all the incoming messages within the sensor group, first a method of combining and organizing these messages is needed. A couple of options are available including the ROS `message_filters` utility. However, the team elected to develop a custom queue to avoid the constraints of existing libraries. Our custom queue system allows messages fusion from as many upstream sensors as required. Nevertheless, the custom pipeline will be advantageous when scaling our system in the long run. The Pugh table in Table 10, explores the trade-offs in our design decisions.

Within each sensor group, the fusion node emplaces all incoming sensor messages in a custom queue, in ascending order, based on their timestamps. A timestamp-based selector is employed both to minimize the process latency and buffer enough messages with close enough timestamps from different sensor modalities for fusion. These messages form a batch that typically contains one set of 3D bounding boxes and an arbitrary number of 2D bounding boxes and radar targets. Each batch uses the 3D bounding boxes to localize the objects in the world coordinate. If additional 2D bounding boxes or radar targets are available, this information is fused with the 3D bounding boxes to form the aggregated observations containing object class, confidence, and velocity information that is used in the downstream Tracker.

For the 2D to 3D fusion, a greedy association is done on the projected 3D bounding boxes on the image plane. A 2D box is created from the projections of the 8 corners from the 3D boxes and the actual 2D bounding boxes

Table 11: 2D to 3D Fusion Projection Comparison

Criteria	Weight	3D Bounding Box Projection	3D Point Cloud Projection
Simplicity			
Ease of development	2	1	-1
Accuracy			
Accuracy of projections	3	-1	1
Efficiency			
Amount of compute required	3	1	-1
Small message size	1	1	-1
Totals		3	-3
Rank		1	2

with the highest IOU are greedily associated. Then the relevant information from each associated 2D bounding box is added to the corresponding 3D bounding box. For the remaining 3D boxes not associated with a 2D box, the type provided from the 3D box based on size is used instead. Despite being simple comparing to other methods in the literature, our approach appears to work well. Table 11 provides a Pugh matrix comparison with other methods.

We chose to associate radar targets directly to the 3D bounding boxes in order to reduce the total calculation needed. Table 12 gives a Pugh matrix comparison between the methods. Radar to 3D fusion was simply done by associating the radar targets with the 3D bounding boxes they fell within. If multiple radar targets were associated with the same 3D box, their radial velocity measurements were averaged and applied to the box. Additionally, for 3D bounding boxes without a known class label, the velocity information from the radar targets can be exploited to infer the object type.

After the above-mentioned fusion steps, one set of unified 3D bounding boxes (observations) remains in the queue. These boxes are converted from their local LiDAR frame to the world frame based on the current GNSS measurement. Finally, the world frame 3D bounding boxes are sent to the Tracker to update the tracked objects. The pseudocode for a single fusion node can be seen in Algorithm 2.

Table 12: Radar Fusion Method Comparison

Criteria	Weight	Radar to 3D Fusion	Radar to 2D Fusion
Simplicity			
Ease of development	1	1	-1
Accuracy			
Accuracy of association	1	1	-1
Efficiency			
Amount of compute required	1	1	-1
Totals		3	-3
Rank		1	2

Algorithm 2 Single Fusion Node

Input: Sensor messages include lists of 2D Bounding Boxes $x_{2D_k} = [cx, cy, h, w, type, confidence]^T$, lists of 3D Bounding Boxes $x_{3D_k} = [x, y, z, l, w, h, yaw, type, confidence]^T$, and Radar targets $x_{radar_k} = [x, y, z, v_{rad}]^T$ and odometry information $odom$.

Output: A list of K detection objects $y_k = [x, y, z, v_{rad}, l, w, h, yaw, type, confidence]^T$

while true **do**
  Push message x to queue whenever x arrives

  Extract Batch from queue: $(x_{3D}, [x_{2D}], [x_{radar}])$
  $y \leftarrow x_{3D}$
  $y \leftarrow Fuse2DTo3D(y, [x_{2D}])$
  $y \leftarrow FuseRadarTo3D(y, [x_{radar}])$
  $y \leftarrow TransformToOdom(y, odom)$
  Send y to tracker
end while

5.2.2 Object Tracking The Tracker receives the 3D observations from all four sensor groups to update a list of tracked objects in the world frame. The observations are first associated with the tracked objects with the Hungarian algorithm. This algorithm was the obvious choice due to its robustness and optimality while requiring minimal implementation effort, as seen in Table 13.

After the observations are associated with the tracked objects, we can estimate the position and velocity of the tracked objects using an Extended Kalman Filter (EKF) [38]. The EKF was the selected for its capability to handle the non-linearities in the radar radial velocity measurements. Table 14 is another Pugh Matrix comparing the other potential state estimation methods that were considered.

The Hidden Markov Model (HMM) [39] is used to obtain stable estimates of detection class and accurate confidence estimates. Two matrices, the transition probabilities and the observation probabilities, are tunable parameters of the HMM. The transition probabilities describe the likelihood of one class label transitions to another for one object between consecutive measurements, and the observation probabilities describe the likelihood that a class label is assigned to an object.

Table 13: Observation Association Method Comparison

Criteria	Weight	Fully Learned Object Tracking	Learned Data Association	Hungarian Method	Greedy Assignment
Simplicity					
Ease of development	2	-1	-1	0	1
Algorithm design simplicity	3	-1	-1	1	1
Accuracy					
Accuracy of association	4	1	1	1	-1
Efficiency					
Algorithm time complexity	1	0	0	0	1
Totals		-1	-1	7	2
Rank		3	3	1	2

Table 14: State Estimation Method Comparison

Criteria	Weight	Linear Kalman Filter	Extended Kalman Filter	Unscented Kalman Filter	Robust Kalman Filter	Information Filter	Square Root Filter
Accuracy							
Prediction accuracy	2	0	1	1	1	0	0
Robustness							
Compatiblity with Non-Linear Models	3	0	1	1	0	0	0
Simplicity							
Ease of development	1	1	0	-1	-1	0	-1
Efficiency							
Algorithm time complexity	1	0	-1	-1	0	1	-1
Totals		1	4	3	1	1	-2
Rank		3	1	2	3	3	6

Finally, the tracked objects will be pruned if they are overlapping or have not been observed for a set amount of time. For overlapping objects, the object with the highest confidence is the one kept while the other is discarded. Objects with no observations will be put on trial after 500ms when they are no longer outputted from the tracker. However, objects on deletion trial may still be associated with incoming detections. After two seconds with no association in the deletion trail state, the object track will be deleted. The pseudocode for the tracker can be seen in Algorithm 3.

Algorithm 3 Object Tracking Process

Input: List of K detection objects x: $x_k = [x, y, z, v_{rad}, l, w, h, yaw, type, confidence]^T$

Output: List of N tracked objects y: $y_n = [x, y, z, \dot{x}, \dot{y}, l, w, h, yaw, type, confidence]^T$

$costMatrix \leftarrow GenerateCostMatrix(y, x)$
(y_i, x_i) pairs $\leftarrow Hungarian(costMatrix)$
$y \leftarrow (y_i, x_i)$ pairs $+ x_i$ and y_i without pairs ▷ Observation Association

for $i \leftarrow 1$ to N **do** ▷ Extended Kalman Filter
  $Predict(y_i)$
  if y_i has x_i: $Update(y_i, x_i)$
end for

for $i \leftarrow 1$ to N **do** ▷ Class Confidence Handler
  if y_i has x_i: $HMMFilter(y_i, x_i)$
end for

$y \leftarrow PruneTracks(y)$ ▷ Remove old and overlapping tracks

return y

6 CONCLUSION

This report introduces the software and hardware designs of the aUToronto Round II Year 1 perception cart. We are confident that we have designed a robust and future-proof perception stack by analyzing various sensor modalities and configurations based on worst-case scenarios, performance, cost, development effort, and other metrics. To build an extendable software architecture for future years, we considered performance, maintainability, and required development resources. With our perception stack and modular software architecture, we are optimistic that we can build toward an SAE Level 4 automated driving system in the next four years.

References

[1] *Sae autodrive challenge ii year 1 rule*, 2022.

[2] Y. Choi, N. Kim, S. Hwang, *et al.*, "Kaist multi-spectral day/night data set for autonomous and assisted driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 3, pp. 934–948, 2018. DOI: 10.1109/TITS.2018.2791533.

[3] K. Burnett, D. J. Yoon, Y. Wu, *et al.*, *Boreas: A multi-season autonomous driving dataset*, 2022. DOI: 10.48550/ARXIV.2203.10168. [Online]. Available: https://arxiv.org/abs/2203.10168.

[4] L. Reddy Cenkeramaddi, J. Bhatia, A. Jha, S. Kumar Vishkarma, and J. Soumya, "A survey on sensors for autonomous systems," in *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2020, pp. 1182–1187. DOI: 10.1109/ICIEA48937.2020.9248282.

[5] M. Gehrig, W. Aarents, D. Gehrig, and D. Scaramuzza, "Dsec: A stereo event camera dataset for driving scenarios," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4947–4954, 2021. DOI: 10.1109/LRA.2021.3068942.

[6] *Tech brief - sony 4th gen pregius s*. [Online]. Available: https://thinklucid.com/tech-briefs/sony-4th-generation-pregius-s/.

[7] *Sensors and lenses: Edmund optics*. [Online]. Available: https://www.edmundoptics.com/resource-page/application-notes/imaging/sensors-and-lenses/.

[8] *Ipc/whma-a-620*. [Online]. Available: https://whma.org/ipcwhma-a-620/.

[9] S. Kato, S. Tokunaga, Y. Maruyama, *et al.*, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, ser. ICCPS '18, Porto, Portugal: IEEE Press, 2018, pp. 287–296, ISBN: 9781538653012. DOI: 10.1109/ICCPS.2018.00035. [Online]. Available: https://doi.org/10.1109/ICCPS.2018.00035.

[10] *Baidu Apollo team (2017), Apollo: Open Source Autonomous Driving, howpublished = https://github.com/apolloauto/apollo, note = Accessed: 2019-02-11*.

[11] G. Jocher, A. Stoken, J. Borovec, *et al.*, *Ultralytics/yolov5: V5.0 - yolov5-p6 1280 models, aws, supervise.ly and youtube integrations*, 2021. DOI: 10.5281/ZENODO.3908559. [Online]. Available: https://zenodo.org/record/3908559.

[12] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, *You only learn one representation: Unified network for multiple tasks*, 2021. arXiv: 2105.04206 [cs.CV].

[13] J. Redmon and A. Farhadi, *Yolov3: An incremental improvement*, 2018. arXiv: 1804.02767 [cs.CV].

[14] RangiLyu, *Nanodet-plus: Super fast and high accuracy lightweight anchor-free object detection model.* https://github.com/RangiLyu/nanodet, 2021.

[15] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[16] Y. Chai, S. Wei, and X. Li, "The multi-scale hough transform lane detection method based on the algorithm of otsu and canny," *Advanced Materials Research*, vol. 1042, pp. 126–130, Oct. 2014. DOI: 10.4028/www.scientific.net/AMR.1042.126.

[17] H. Wang, Y. Wang, X. Zhao, G. Wang, H. Huang, and J. Zhang, "Lane detection of curving road for structural highway with straight-curve model on vision," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, pp. 5321–5330, 2019. DOI: 10.1109/TVT.2019.2913187.

[18] Z. Kim, "Robust lane detection and tracking in challenging scenarios," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 1, pp. 16–26, 2008. DOI: 10.1109/TITS.2007.908582.

[19] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, ISSN: 0001-0782. DOI: 10.1145/358669.358692. [Online]. Available: https://doi.org/10.1145/358669.358692.

[20] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, *Spatial as deep: Spatial cnn for traffic scene understanding*, 2017. DOI: 10.48550/ARXIV.1712.06080. [Online]. Available: https://arxiv.org/abs/1712.06080.

[21] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[22] M. Himmelsbach, F. v. Hundelshausen, and H.-J. Wuensche, "Fast segmentation of 3d point clouds for ground vehicles," in *2010 IEEE Intelligent Vehicles Symposium*, 2010, pp. 560–565. DOI: 10.1109/IVS.2010.5548059.

[23] A. Paigwar, Ö. Erkent, D. S. González, and C. Laugier, "Gndnet: Fast ground plane estimation and point cloud segmentation for autonomous vehicles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[24] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[25] H. Caesar, V. Bankiti, A. H. Lang, *et al.*, *Nuscenes: A multimodal dataset for autonomous driving*, 2020. arXiv: 1903.11027 [cs.LG].

[26] J. Houston, G. Zuidhof, L. Bergamini, *et al.*, *One thousand and one hours: Self-driving motion prediction dataset*, 2020. eprint: arXiv:2006.14480.

[27] P. Sun, H. Kretzschmar, X. Dotiwalla, *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2446–2454.

[28] T. Yin, X. Zhou, and P. Krähenbühl, *Center-based 3d object detection and tracking*, 2021. arXiv: 2006.11275 [cs.CV].

[29] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 697–12 705.

[30] *Lidar camera calibrator*. [Online]. Available: https://www.mathworks.com/help/lidar/ug/lidar-camera-calibration-guidelines.html.

[31] L. Zhou, Z. Li, and M. Kaess, "Automatic extrinsic calibration of a camera and a 3d lidar using line and plane correspondences," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 5562–5569. DOI: 10.1109/IROS.2018.8593660.

[32] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992. DOI: 10.1109/34.121791.

[33] D. F. Crouse, "On implementing 2D rectangular assignment algorithms," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 4, pp. 1679–1696, Aug. 2016, ISSN: 1557-9603. DOI: 10.1109/TAES.2016.140952.

[34] X. Weng, J. Wang, D. Held, and K. Kitani, "3D Multi-Object Tracking: A Baseline and New Evaluation Metrics," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 10 359–10 366. DOI: 10.1109/IROS45743.2020.9341164.

[35] H.-k. Chiu, J. Li, R. Ambrus, and J. Bohg. "Probabilistic 3D Multi-Modal, Multi-Object Tracking for Autonomous Driving." arXiv: 2012.13755 [cs]. (Oct. 10, 2021).

[36] A. Kim, A. Ovsep, and L. Leal-Taixé. "EagerMOT: 3D Multi-Object Tracking via Sensor Fusion." arXiv: 2104.14682. (2021).

[37] K. Burnett, S. Samavi, S. L. Waslander, T. D. Barfoot, and A. P. Schoellig, *Autotrack: A lightweight object detection and tracking system for the sae autodrive challenge*, 2019. eprint: arXiv:1905.08758.

[38] T. D. Barfoot, *State Estimation for Robotics*. Cambridge University Press, 2017.

[39] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *The annals of mathematical statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.