Zeus: A System Description of the Two-Time Winner of the Collegiate SAE AutoDrive Competition

Keenan Burnett[*]   **Jingxing Qian**   **Xintong Du**   **Linqiao Liu**   **David J. Yoon**

Tianchang Shen   **Susan Sun**   **Sepehr Samavi**   **Michael J. Sorocky**

Mollie Bianchi   **Kaicheng Zhang**   **Arkady Arkhangorodsky**   **Quinlan Sykora**

Shichen Lu   **Yizhou Huang**   **Angela P. Schoellig**   **Timothy D. Barfoot**

Abstract

The SAE AutoDrive Challenge is a three-year collegiate competition to develop a self-driving car by 2020. The second year of the competition was held in June 2019 at MCity, a mock town built for self-driving car testing at the University of Michigan. Teams were required to autonomously navigate a series of intersections while handling pedestrians, traffic lights, and traffic signs. Zeus is aUToronto's winning entry in the AutoDrive Challenge. This article describes the system design and development of Zeus as well as many of the lessons learned along the way. This includes details on the team's organizational structure, sensor suite, software components, and performance at the Year 2 competition. With a team of mostly undergraduates and minimal resources, aUToronto has made progress towards a functioning self-driving vehicle, in just two years. This article may prove valuable to researchers looking to develop their own self-driving platform.

1 Introduction

aUToronto is a team of undergraduate and graduate students at the University of Toronto. In just two years, the team has built *Zeus*, a self-driving car that takes a step towards Level 4 autonomy on a closed course. This work has been centered around competing in the SAE AutoDrive Challenge, a collegiate competition to develop a self-driving car, in only three years. Similar to the DARPA Urban Challenge from 2008, this competition required teams to autonomously sequence several intersections while handling a realistic urban driving environment (Urmson et al., 2008).

The second year of the competition simulated a small town with tasks requiring a high degree of autonomy. Only one hour was given at the start of the competition to perform minor adjustments and prior access was prohibited. Furthermore, only two attempts were allowed for each course.

With this in mind, competing systems needed to be exceptionally reliable. Developing such a system with minimal resources in a constrained time frame required rapid development and simple designs. This article attempts to provide a complete picture of the design and development of Zeus with topics including team

---

[*]All authors are affiliated with the University of Toronto. Questions and comments can be sent to keenan.burnett@autodrive.utoronto.ca
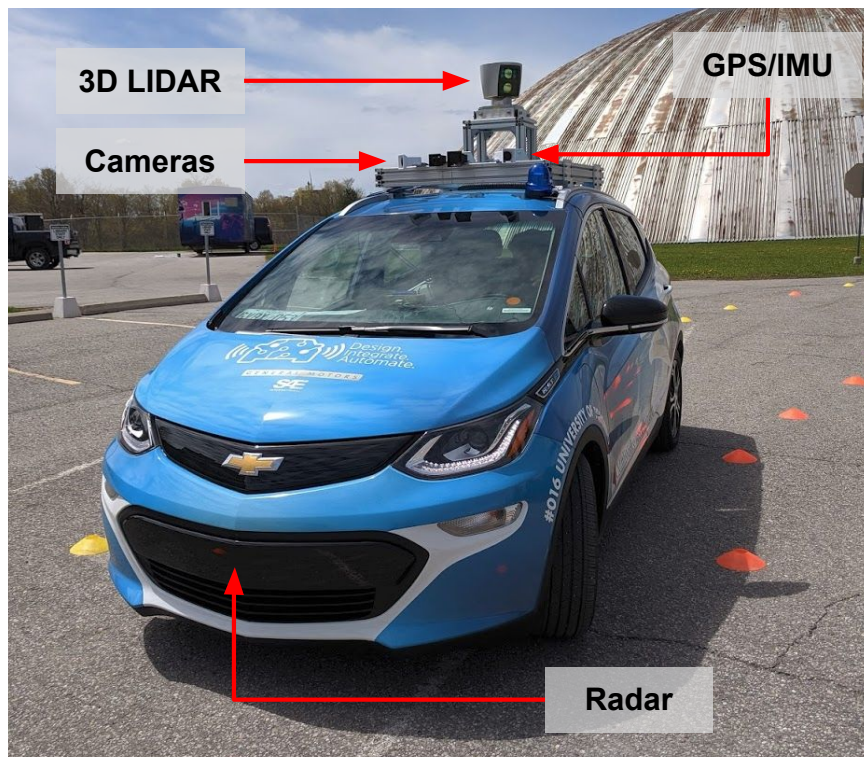
Figure 1: Our self-driving car *Zeus* at the University of Toronto. A video depicting our Year 2 competition performance can be found here: `https://youtu.be/2Z6mPKIv0TM`

organization, sensor configuration, deep learning acceleration, and data collection. Further, this article will cover the design of each major software component. This article concludes with a detailed analysis of our Year 2 competition performance and lessons learned. As a lot of current self-driving development occurs in secret industrial labs, our hope is that this comprehensive summary of aUToronto's work may prove useful to the wider robotics community.

2   Background

2.1   Related Work

The DARPA Urban Challenge competitors were the first to demonstrate the feasibility of autonomous driving in an urban setting (Urmson et al., 2008), (Montemerlo et al., 2008). This challenge required competitors to sequence several intersections while obeying the rules of the road and interacting with other autonomous vehicles. These systems relied heavily on GPS for localization, and relied on LIDAR for detecting lanes and vehicles. Few teams made significant use of vision. In the decade since, vision has become a key sensing modality. This is in part due to the proliferation of parallel computing and advances in deep learning.

More recently, these competitors provided updates to the Urban Challenge systems. In (Levinson et al., 2011), one update involved the use of high-resolution maps for online localization with centimeter-level accuracy. In (Wei et al., 2013), the authors describe an updated self-driving platform constructed using close-to-market sensors.

In 2014, the Mercedes/Bertha project succeeded in navigating 100 km autonomously using only radar and vision (Ziegler et al., 2014). The route included a mix of urban and rural driving with traffic lights, pedestrian

crossings, and intersections with traffic. In order to navigate the route successfully, they relied on a detailed digital map which encoded the road topology and lane locations. A stereo camera was used as the primary source of 3D object detections. In 2017, the same group published an update for the 2016 Grand Cooperative Driving Challenge (Tas et al., 2018). They added LIDAR sensors and provided several updates to their perception software. However, this perception software was not enabled during their competition.

There have also been efforts to open-source self-driving software, notably Autoware (Kato et al., 2015) and Baidu's Apollo (Apollo, 2019). Both groups aim to provide an open-source repository to enable Level 4 autonomous driving in an urban environment. The Apollo software suite is much more advanced, having undergone a more rigorous development and testing process. The Apollo repository is also used by numerous other groups as a basis for research and development. To date, a system-level summary of Apollo has not yet been published. The most recent update to the Autoware project focuses on implementing their software on embedded platforms (Kato et al., 2018).

Other published systems include the V-Charge project (Furgale et al., 2013), BMW's work on autonomous highway driving (Aeberhard et al., 2015), an autonomous golf cart pilot in Singapore (Pendleton et al., 2015), and the PROUD driverless car test (Broggi et al., 2015). In the V-Charge project, the goal was to develop an autonomous valet parking system using close-to-market sensors. Aeberhard et al. (2015) describe the system that was developed at BMW to perform autonomous highway driving. Their system uses a combination of LIDAR, vision and radar to detect lane markings and other vehicles. They also relied on high-precision maps of the lanes and road boundaries.

Pendleton et al. (2015) focused on demonstrating a low-speed autonomous shuttle to raise public awareness of autonomous vehicles. Since the golf carts operated continuously for long periods of time, reliability was critical. The vehicles were programmed to follow a predefined path and would stop and wait for any blockage to become clear.

Broggi et al. (2015) conducted an autonomous driving test on public roads in Parma, Italy. Their system used both vision and LIDAR to detect vehicles and relied on a highly-accurate GPS/IMU for localization. Although impressive at the time, their route included minimal interaction with other vehicles, only a single traffic light, and pedestrians were limited to a controlled situation at the end of the route.

Since the DARPA Urban Challenge, there have been several other autonomous driving competitions. The first Grand Cooperative Driving Challenge (GCDC) was held in 2011. This competition focused on the interaction between autonomous vehicles. Each vehicle was able to communicate with the others via a predetermined V2V radio interface. The 2016 GCDC included cooperative lane changes and cooperative intersection handling (Tas et al., 2018).

From 2010-2014, three autonomous vehicle competitions were held in South Korea, organized by Hyundai. The 2012 competition included a simulated urban environment with traffic lights, moving vehicles, and static pedestrians. The winning system relied primarily on 2D LIDAR to detect obstacles and a GPS/IMU to localize (Jo et al., 2015).

In China, the Intelligent Vehicle Future Challenges (IVFCs) have been held every year since 2009 with increasing complexity. In 2012, the competition included lane keeping, traffic lights and signs, pedestrian avoidance, and merging into moving traffic.

Another competition, but one that's dedicated to students, is the Formula Student Driverless competition which was first held in 2017. The competition required teams to complete 10 laps of a previously unknown track delimited by pylons. The vehicle employed was an electric 4WD car developed by AMZ for the Formula Student Electric challenge in 2015. The winning team, flüela driverless, developed a LIDAR SLAM system to localize within the track (Valls et al., 2018).

A new competition which began its initial development in 2016 is the Roborace competition. Eventually their

goal is to have as many as 10 autonomous cars on a track racing simultaneously. The vehicular platforms themselves are standardized across teams and include the numerous sensors and compute power required for the competition. The idea will be that individual teams will develop the software that will run on these platforms (Roborace, 2019).

Dataset papers may serve as useful references of sensor configurations. Currently, the most popular benchmark for autonomous driving is the KITTI dataset (Geiger et al., 2012). Other self-driving datasets include the Oxford Robotcar dataset (Maddern et al., 2017), the ApolloScape dataset (Huang et al., 2018), and NuScenes (Caesar et al., 2019). Each of these datasets provides LIDAR, camera, and GPS/IMU data for different purposes.

The SAE AutoDrive Challenge is unique in that it is a full self-driving competition aimed primarily at undergraduate students. With minimal resources and a constrained time frame, our team of students was able to develop a functioning autonomous vehicle. This work provides an up-to-date description of a self-driving car including information on team organization, sensor configuration, and deep learning acceleration. Such a complete picture of the self-driving development process has not been shown in literature before. Another factor that distinguishes this work is our use of deep learning for perception. Of the previously mentioned systems, only Autoware, Apollo, and Bertha (2016) claim to make use of deep learning. However Autoware has not been rigorously tested, Apollo has yet to publish results of their testing, and Bertha's updated perception suite was disabled for their competition. Furthermore, this work describes our closed-loop performance in detail with several figures, which is important but uncommon among existing works. Due to Intel being an exclusive computing sponsor of the AutoDrive competition, they have restricted teams from using NVIDIA GPUs for on-board computations. As a result, we have invested significant effort into leveraging CPUs and FPGAs for deep learning acceleration, which we discuss in this work.

(a) Speed Zone Course

(b) Traffic Control Sign Challenge
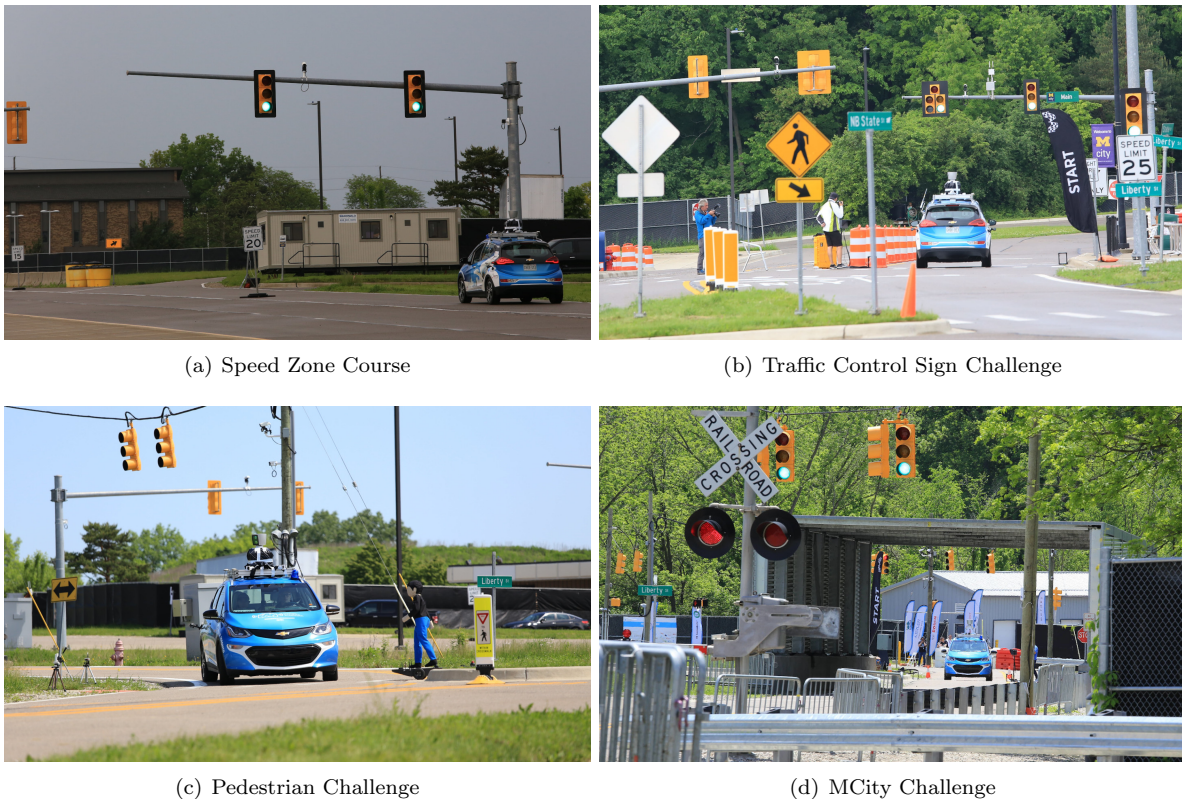
(c) Pedestrian Challenge

(d) MCity Challenge

Figure 2: Zeus during the Year 2 competition at MCity. The Year 2 competition was divided into four challenges: Traffic Control Sign Challenge (with Speed Zone Course), Pedestrian Challenge, MCity Challenge, and Intersection Challenge.

2.2 Year 2 Competition

The first year of the competition required teams to perform lane-keeping, stop at a series of stop signs and avoid static objects by performing lane-change maneuvers. This competition was held at GM's Desert Proving Grounds in Yuma, Arizona. No map of the course was given and multiple laps were not permitted. Lane detection was intended to be the sole source of localization information. Although the challenges were straightforward, only six months were given between the vehicle delivery date and the competition. In order to achieve a working system in time, we relied on simple, robust algorithms. A description of our Year 1 architecture can be found in (Burnett et al., 2018).

The second year was divided into four challenges: the Traffic Control Sign Challenge, Pedestrian Challenge, Intersection Challenge, and MCity Challenge.

The Traffic Control Sign Challenge had two segments: a speed zone and a traffic sign course. The speed zone required vehicles to perform lane-keeping while abiding by posted speed limits. Traffic sign positions were not encoded in the map given to teams. The traffic sign course required vehicles to continue straight along the road unless directed otherwise by signs present on the road. There were 13 possible sign classes in total including Right-Turn-Only and Do-Not-Enter signs. The traffic sign course ended in a pull-in parking maneuver where some spots were occupied and some had handicap parking signs associated with them.

For the other three challenges, high-level GPS waypoints were given. Waypoints were placed at the center of each intersection where a turn needed to be made. These waypoints provided a global path through MCity but were insufficient for autonomous driving by themselves.

In the Pedestrian Challenge, pedestrian dummies were placed at the side of marked crosswalks and at intersections with flashing red lights. If a pedestrian is waiting at the side of the crosswalk, vehicles are required to come to a stop and wait for the pedestrian to completely cross the road. In the case that a pedestrian is stationary for longer than five seconds, the vehicle should continue driving. The most difficult scenario involves making a left-hand turn through an intersection with flashing red lights. In this case, it is necessary to check for pedestrians crossing the road both immediately in front of the vehicle and on the left-hand crosswalk.

In the Intersection Challenge, vehicles were required to react appropriately to traffic lights at a series of intersections. This challenge was the longest, requiring teams to sequence 13 intersections to complete the course.

The MCity Challenge required vehicles to sequence several intersections while handling obstacles along the way. These obstacles included a tunnel, railroad crossing, static cyclist, and a dynamic animal. Several images of the Year 2 competition courses are shown in Figure 2.
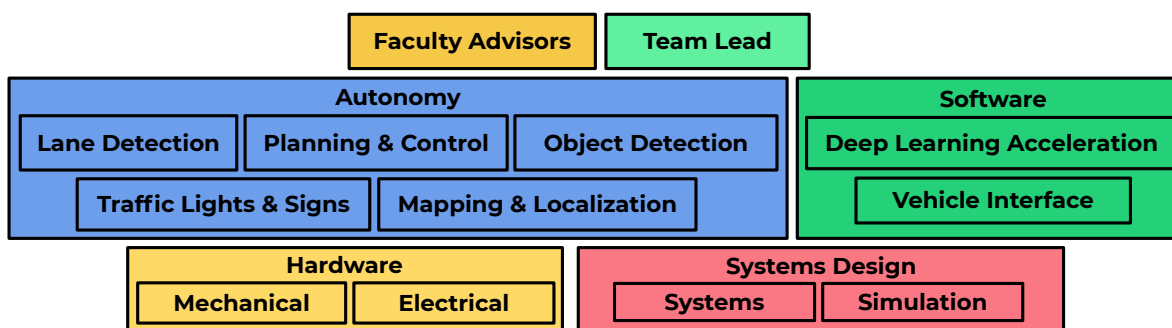

Figure 3: aUToronto's organizational chart. The team is roughly divided into the areas of autonomy, software, hardware, and systems design. This hierarchical structure was critical to avoid overwhelming the Team Lead.

3 Team Organization

aUToronto consists of close to 100 students at the University of Toronto, the majority of which are undergraduates. Figure 3 depicts the team's organizational structure. The team has twelve sub-teams that cover the broader areas of autonomy, software, hardware, and systems design. Most sub-teams focus on a single component of the software stack. Each sub-team has 1-2 team leads who take responsibility for their sub-team's progress. The sub-teams themselves range in size between three and ten general members. The Team Lead makes high-level design decisions and interfaces directly with sub-team leads. This two-tiered management structure prevents the Team Lead from being overwhelmed.

All team software was tracked using private Gitlab repositories. Where publicly available repositories were used, such as ROS sensor drivers, these repositories were cloned and kept in a frozen state to prevent regression. A single privileged repository was used to track the software that would run in realtime on Zeus. All third party software and adjacent projects were kept separate. A single ROS workspace is used for all aUToronto software, enabling a simple build process. aUToronto's continuous integration environment included a build script which would create a virtual machine, install all dependencies, and build the codebase from scratch. At the bare minimum, code was required to build before being able to merge into the main repository. aUToronto employed a plus-one system whereby all new features and bug fixes were submitted as a merge request and subsequently reviewed by a team lead. The team strived to follow the Google C++ coding guidelines but this was not strictly enforced. In general, we relied quite heavily on real-world tests conducted on Zeus. Individual components were tested in isolation using either data taken on Zeus for offline testing or online testing while Zeus was operating in open-loop. A set of regression tests was performed on Zeus on a weekly basis to verify that basic features were not deteriorating. Integration testing was conducted by testing the system in open-loop both offline and directly on Zeus. Simulation testing was limited to the validation of planning logic and semantic maps. The simulation environments we used included custom C++ ROS tests and the RightHook simulation environment (RightHook, 2020).

Weekly meetings are held by the Team Lead with all sub-team leads present. This serves to keep the entire team on track and to encourage communication between sub-teams. All work is carried out by the students with faculty advisors only providing occasional advice.

The team consists primarily of undergraduates working in their spare time. For this reason, it is critical to have a timeline that is both aggressive and realistic. aUToronto's development timeline for Year 2 of the competition is depicted in Figure 4. The timeline depicts the seven months leading up to the competition. The four months prior to this point consisted of hiring, on-boarding, research, and ramping up development efforts. The main parts highlighted in the timeline are the milestones and system test campaign.

The purpose of each milestone is to gradually improve on capabilities that can be demonstrated in real-world tests. Each milestone ends with a series of closed-loop tests at a new location. This serves to motivate the entire team to achieve a common objective in a timely manner. During the System Test Campaign, a small team worked almost daily on testing and bug fixes. This test campaign was vital for achieving good performance at the competition.
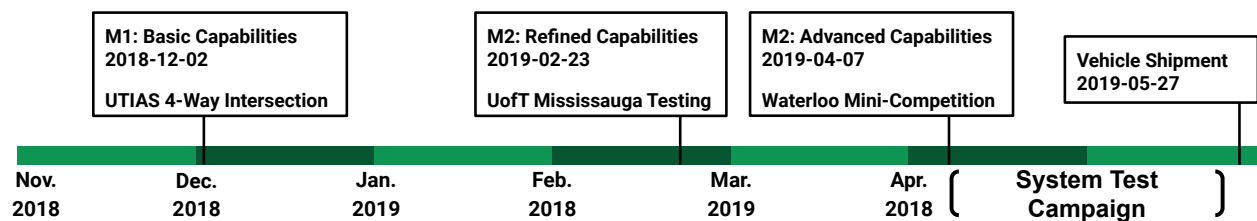

Figure 4: Year 2 development timeline. The timeline was set up so that major milestones incrementally added features to Zeus. Each milestone culminated in a series of real-world tests at a new location. Test locations included U of T's Mississauga campus, and the Clearpath Robotics office.

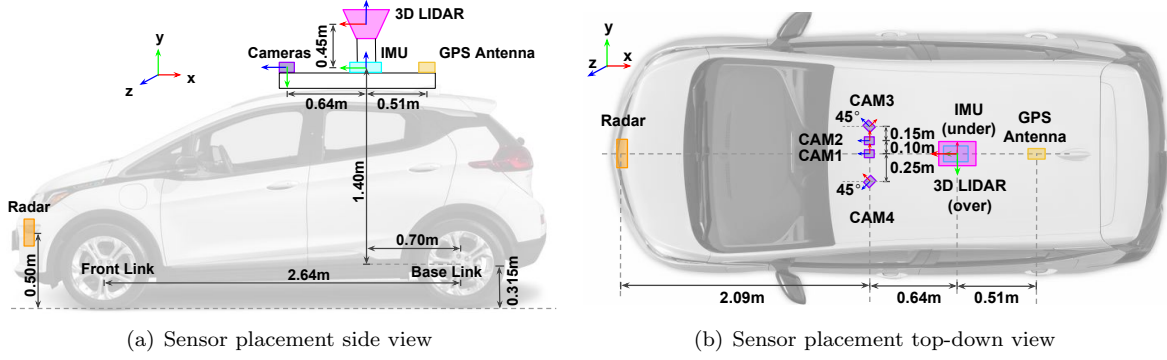(a) Sensor placement side view   (b) Sensor placement top-down view

Figure 5: Several sensors are placed along the longitudinal axis of the vehicle. Most sensors are placed so that rotation angles are multiples of 90 degrees. These two factors simplify calibration.

4 System Overview

aUToronto's self-driving car, Zeus, is a 2017 Chevrolet Bolt electric vehicle. The sensor suite includes a Velodyne HDL-64 3D LIDAR, four Blackfly S monocular cameras (5.0 MP, 75 FPS), a Novatel PwrPak7 GPS/IMU, and a Continental ARS430 radar. The compute server has two Intel Xeon E5-2699 v4 processors that together contain 44 physical cores operating at 3.6 GHz. The server also contains an Intel Arria 10 FPGA for deep learning acceleration. The total cost of the hardware added to Zeus is $175,000 CAD.

Figure 5 depicts the placement of sensors on Zeus. The primary factors that drove sensor placement include ease of calibration, field of view, and detection range. Camera 1, 3D LIDAR, IMU receiver, and GPS antenna are all placed along the longitudinal axis of the vehicle. Where possible, sensors are placed such that the rotation angles between them are a multiple of 90 degrees. These two factors make the calibration process simpler. Sensor mounts were designed so that the transformation obtained from the CAD model would be close to the calibrated value.

Figure 6 depicts sensor fields of view. Of the four monocular cameras, two point straight forwards, and two at 45-degree angles. This configuration was chosen to maximize the horizontal field of view. Cameras 1, 3, 4 have a 5-mm lens, which results in an 80-degree field of view. Camera 2 uses a 16-mm lens, which results in a 30-degree field of view. This narrow-field-of-view camera doubles the visual detection range of objects from 50 m up to 100 m.

The Velodyne HDL-64 has a rated range of 120 m. In our experiments, the effective range of pedestrian and vehicle detection is closer to 40 m and 80 m, respectively. An automotive radar sensor was installed behind the front bumper. This sensor was not used during the Year 2 competition, but is intended as an extra layer of safety in the future.

To calibrate the extrinsic transformation between Camera 1 and the Velodyne, we use the open-source toolbox described in Unnikrishnan and Hebert (2005). Calibration is performed by moving a checkerboard target in front of the vehicle and capturing image and LIDAR pointcloud pairs. For other extrinsic transformations, hand measurements and CAD model values are used. We also design our perception algorithms to be robust to minor calibration error.

Mechanical sensor mounts are attached to a rectangular rack consisting of double-wide aluminum extrusion. The rack is mounted rigidly to the vehicle's sport rails via custom-machined interface components. This structure provides a base for modular and reconfigurable sensor attachments. A central tower structure is employed to mitigate occlusion of the Velodyne HDL-64.

All electronics are powered via the Chevrolet Bolt's Auxiliary Power Supply, which can supply up to 1 kW at 12 V. A liquid cooling system was installed to regulate the temperature of the CPUs and FPGA.
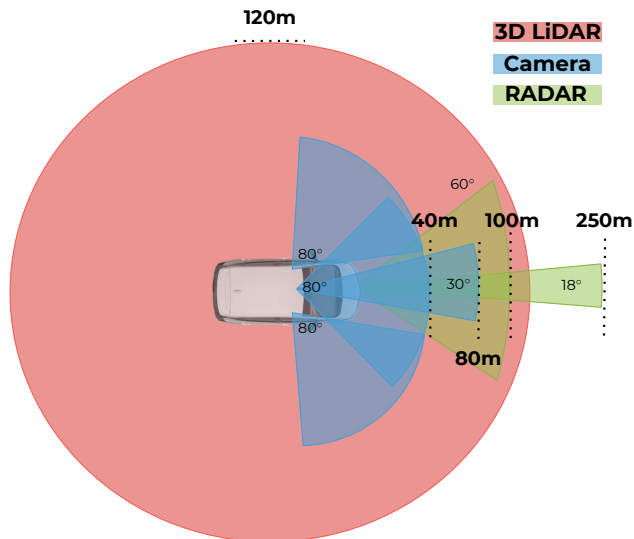
Figure 6: This figure depicts the fields of view of sensors on Zeus. Note that the front-facing cameras provide close to 170 degrees of horizontal field of view. A narrow-field-of-view camera boosts visual detection range.

Autonomous control of the vehicle is facilitated through the Vehicle Interface: a software API developed by aUToronto that enables communication between the ROS network and the vehicle. Communication is achieved using a serial data connection over the vehicle's CAN buses. This enables the autonomy software to control torque, steering, and transmission. Each team was given documentation from GM to enable the development of this interface.

Figure 7 depicts Zeus' software architecture. Sections 7,8,9,11 provide detailed information on Object Detection, Light and Sign Detection, Lane Detection, and Localization. The perception nodes generate a high-level abstraction of the environment around Zeus, and pass this information on to the planner. The planner first generates a global path for the vehicle by searching through a road graph and reaching each high-level waypoint. Internally, we convert third party semantic maps into our own format which is based on OpenStreetMaps. Our internal format has a graph-like structure which can easily be traversed using a graph search algorithm. The planner also generates a local trajectory by linking together the centerlines of the desired road segments and performing minor path smoothing. The controller then outputs torque and steering commands which are enacted by the Vehicle Interface. The design of the planner and controller is described in sections 12,13. All software was written in C++ and runs on ROS Kinetic (ROS, 2019) using the compute server described above.

5   Deep Learning Acceleration

Autonomous driving software presents a heavy computational load. The optimization of this software on our computing platform is driven by reducing end-to-end latency between sensor inputs and control actions. Zeus relies on multiple DNNs to process high-resolution camera images.

Each software component is implemented as one or more nodes using ROS as the underlying infrastructure. Many nodes must run concurrently, and the end-to-end latency from sensor inputs to control actions should not exceed 100 ms. This requirement is based on the commonly used rule-of-thumb for what constitutes a real-time system. DNNs are the most significant challenge to minimizing latency.

The Intel Crystal Rugged server has a theoretical peak floating-point operation throughput of 1.8 TFLOPS. At a glance, this appears sufficient for running inference of several DNNs in a timely fashion. However, in
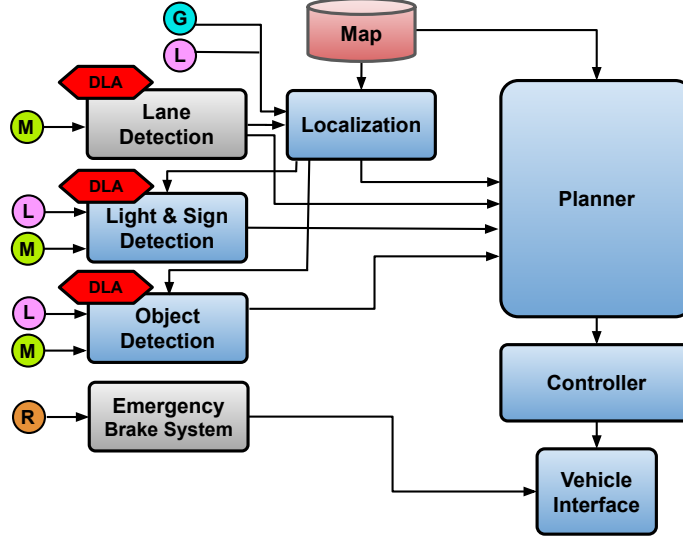
Figure 7: This figure depicts Zeus' software architecture. L: 3D LIDAR, M: Monocular Cameras, R: Radar, G: GPS/IMU, DLA: Deep Learning Acceleration. Localization is either Novatel GPS/IMU or Applanix LIDAR localization. Lane Detection and Emergency Braking (Gray) were not used at the Year 2 competition.

practice it is challenging to select a DNN architecture that is capable of running in realtime, on sufficiently large images, without a GPU. To address this constraint, our team configured an Intel Arria 10 FPGA to be used as a DNN inference accelerator.

We observed that SqueezeDet exceeded its competitors in terms of speed for medium-size images (Wu et al., 2017). A diagram of the SqueezeDet architecture is given in Figure 8. Given six CPU cores, SSD300 yielded a latency of 100 ms. A ResNet-50 backbone paired with SqueezeDet's ConvDet layer yielded a latency of 110 ms. Compared to SqueezeDet's latency of 40 ms, neither of these options were suitable. YoloV3 is a good alternative, promising improved performance and high inference speed (Redmon and Farhadi, 2018). However, YoloV3 is not natively supported by OpenVINO. Figure 9 shows that there are diminishing returns to accelerating DNNs on multiple CPUs. This places an upper limit on the complexity of a viable DNN architecture and the size of the input image.

There are couple of reasons why SqueezeDet runs so well on OpenVINO. First, it is comprised entirely of convolutional layers, making it straightforward to accelerate. Second, SqueezeDet does not use multi-scale features as is done in SSD and YoloV3. In those other networks, higher-level features in the network are concatenated to the feature volume immediately preceding bounding box regression. This allows those networks to achieve better performance across multiple scales at the expense of computation time.
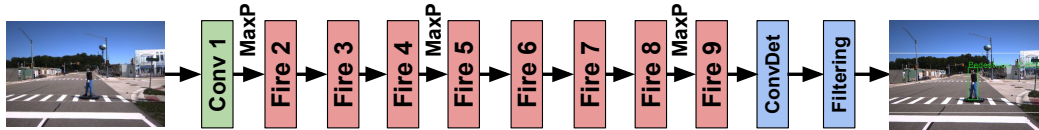

Figure 8: The SqueezeDet architecture. Fire: 1x1 convolutions followed by a RELU operation, 1x1 convolutions and 3x3 convolutions computed in parallel, and another RELU operation. ConvDet: bounding boxes are directly regressed from a dense anchor grid. Filtering: non-maximum suppression. MaxP: max pooling.

The key to leveraging our server's computing power is the Intel OpenVINO SDK. OpenVINO provides runtime libraries optimized for Intel CPUs. Operations for DNN inference can benefit from SIMD instructions and multithreading. Moreover, OpenVINO contains pre-compiled accelerator images that target the Arria 10 FPGA. We implemented a library in C++ called *Zeus DLA* that uses these features. It preprocesses
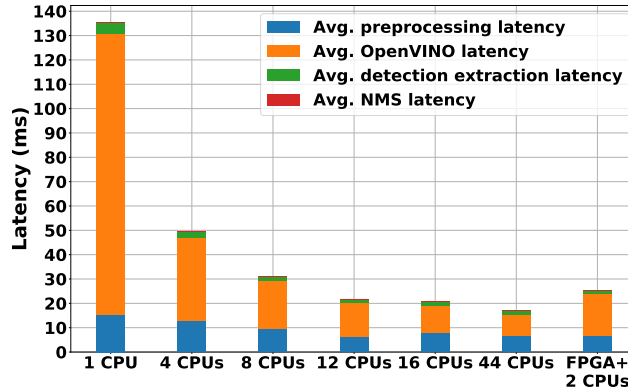
Figure 9: SqueezeDet inference on 2x Intel Xeon E5-2699 v4 processors with support of Intel OpenVINO. This figure demonstrates the diminishing returns of accelerating SqueezeDet with CPUs.

input images, performs inference using OpenVINO, and performs unsupported operations in C++.

We benchmarked the performance of Zeus DLA by running SqueezeDet with a different number of CPU cores and single-batched images on the CPUs as well as on the Arria 10 FPGA. Using 8 out of the 44 available CPU cores, our inference library can accomplish inference in 32 ms, as shown in Figure 9. Using the FPGA accelerator for SqueezeDet, the inference time with Zeus DLA is 26 ms.

During the Year 2 competition, the Pedestrian Challenge presented the most demanding computational load. In this case, three DNNs are used to detect pedestrians on three different cameras; in addition, a single DNN is used to detect traffic lights. Three of the DNNs are assigned 8 CPU cores each, and the fourth DNN is run on an FPGA. This leaves 20 CPU cores for the remainder of the software stack.

6   Machine Learning and Data Collection

Zeus' perception algorithms rely on DNNs to localize objects within 2D images. Early in development, it became apparent that using publicly available data was insufficient for performing well in real-world driving. The remainder of this section describes our experience building a dataset and fine-tuning our DNNs.

The Year 2 competition required us to detect thirteen different sign classes, three different traffic light configurations, and pedestrians. A common starting point is to use publicly available training data. Bosch has a good dataset for traffic lights (Behrendt and Novak, 2017). However, the training set was not sufficient by itself to obtain good performance in our experiments. KITTI is one of the most popular self-driving datasets, with over 7000 training images including pedestrians, cyclists, and cars (Geiger et al., 2012). Training on only KITTI data did not generalize well to experiments on our vehicle. In order to achieve better performance, a custom training set was required.

Most of our dataset was collected on public roads in Toronto in various weather conditions. Images were extracted at one frame per second. In some cases, it was not possible to obtain all the required data on public roads. In this case, replicas of the competition equipment were purchased and set up on private roads at U of T. This included the same traffic lights used at MCity, American traffic signs, and a replica of the competition pedestrian dummy. By obtaining replicas of their equipment, it was possible to optimize our DNNs for the competition environment.

All of our own data were collected in Toronto. However, the competition was to be held in Ann Arbor, Michigan. For this reason, there was some concern that our dataset might be too optimized for Toronto. To combat this, images were extracted from 4K dashcam YouTube videos of New York, Pittsburgh, and Vancouver. These videos were used to increase the variation within our traffic light dataset.

The final dataset consisted of 17000 images of traffic lights, 15000 images of pedestrians and cars, and 35000 images of traffic signs. We sought to outsource the data labelling task. Initially, we tried Amazon's Mechanical Turk but ran into several issues of label quality. A common solution to improve label quality is to have each image labeled by at least three different workers. Then, intersection-over-union (IoU) metrics can be used to automatically flag discrepancies. Even with these measures in place, the quantity of false negatives and coarsely drawn boxes increases substantially in frames with more than ten objects.

To achieve better labels, we switched to using the Scale.ai data labelling service. At a slightly higher price, they guarantee precision, recall, and IoU targets on specified objects. Switching to Scale labels resulted in 5-10% absolute improvement on our validation sets. The lesson here is that the quality of training labels can have a substantial impact on performance. Examples of pedestrian, car, and traffic light labels are depicted in Figure 10. This dataset will remain private while U of T is still competing in the AutoDrive Challenge.

aUToronto team members worked for several months to tune hyperparameters to reach at least 90% precision and recall for each visual perception task. These hyperparameters included the structure of the loss function, batch size, anchor sizes, pre-processing steps, data augmentation steps, and the probability threshold for positive detections. Our team's experimentation was bottle-necked by access to GPU resources. When object tracking was included, the closed-loop performance was qualitatively observed to output less false positives and less false negatives. In general, collecting more training images resulted in better performance but with diminishing returns. Combining our dataset with other public datasets did not substantially improve performance. We observed that reducing the number of classes led to slightly better performance. Over 2000 images of our pedestrian dummy were added to the training set to ensure that this object would be detected. In experiments where pedestrian detection failed unexpectedly, we added these images to the training set.

The loss function used to train SqueezeDet is given in (1) below. Bounding boxes were obtained by regressing four parameters: $(\delta x_{ijk}, \delta y_{ijk}, \delta w_{ijk}, \delta h_{ijk}) = R$ relative to K anchors at each grid location (i,j) in the output. $I_{ijk} = 1$ if the k-th anchor at (i,j) has the largest overlap with the ground truth bounding box. γ_{ijk} is the predicted confidence score of the DNN. γ_{ijk}^G is obtained by computing the IOU between the predicted bounding box and the ground truth. We found SmoothL1 loss on the bounding box parameters to achieve slightly better performance than simple squared error. The final term in the loss function corresponds to cross entropy over the box classes where $\ell_c^G \in \{0,1\}$, $p_c \in [0,1]$, $c \in [1,C]$. We used AdamOptimizer to train SqueezeDet using an initial learning rate of 0.001 for five epochs with a batch size of 20. We set $\lambda_{bbox} = 5$, $\lambda_{conf}^+ = 7.5$, $\lambda_{conf}^- = 2.5$. W and H correspond to the width and height of the input image. Our tuned threshold for positive detections was a confidence score of 0.4.

$$
\begin{aligned}
\mathcal{L} = {} & \frac{1}{N_{obj}}\lambda_{bbox} \sum_{i,j,k,r \in R} I_{ijk}\text{SmoothL1}(\delta r_{ijk} - \delta r_{ijk}^G) \\
& + \sum_{i,j,k} \left[\frac{\lambda_{conf}^+}{N_{obj}} I_{ijk}(\gamma_{ijk} - \gamma_{ijk}^G)^2 + \frac{\lambda_{conf}^-}{WHK - N_{obj}}(1 - I_{ijk})\gamma_{ijk}^2 \right] + \frac{1}{N_{obj}} \sum_{i,j,k} I_{ijk}\ell_c^G \log(p_c)
\end{aligned} \tag{1}
$$

(a) Pedestrians and Cars   (b) Traffic Lights

Figure 10: This figure shows examples of the quality of Scale annotations for the tasks of object detection and traffic lights respectively. Scale annotations tend to be tight and have exceptional precision and recall.
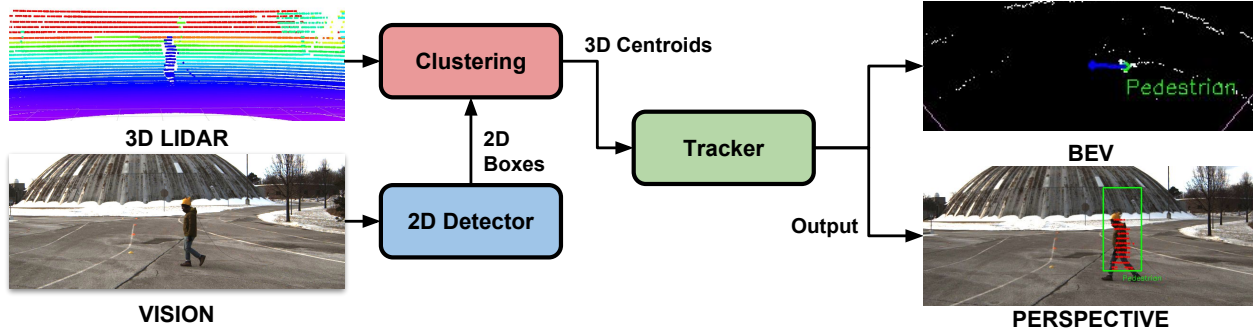
Figure 11: aUToTrack: our pipeline for 3D object detection and tracking.

7   Object Detection and Tracking

aUToTrack consists of an off-the-shelf vision-based 2D object detector paired with a LIDAR clustering algorithm to extract a depth for each object. Ego-vehicle localization is then used to localize objects in a static global reference frame. Given these 3D measurements, we then used greedy data association and a linear Kalman filter to track the position and velocity of each object. Figure 11 illustrates the aUToTrack pipeline. In Burnett et al. (2019), we demonstrated that aUToTrack accurately estimates the position and velocity of pedestrians on both the KITTI Object Tracking benchmark and our own dataset, UofTPed50. We have made this dataset publicly available, and it can be accessed using the link below [1]. An updated version of our pipeline runs in less than 50ms on CPUs.

The SAE AutoDrive Challenge has so far restricted GPUs from being used for on-board computations. Thus, significant effort was invested into designing a lightweight pipeline capable of running on CPUs. With this in mind, SqueezeDet was chosen as our 2D detector for the competition (Wu et al., 2017).

7.1   Related Work

R-CNN was the work that first established CNNs as the state of the art for object detection in images (Girshick et al., 2013). In their subsequent work on Faster R-CNN, the two-stage approach to object detection was introduced (Ren et al., 2015). In a two-stage approach, a set of bounding box proposals is first generated by a region proposal network. In the second stage, these proposals are refined and classified.

YOLO and SSD are credited with popularizing the notion of a single-stage detector (Redmon et al., 2016) (Liu et al., 2016). SSD introduced the concept of anchor boxes in which each position in the output feature tensor has a set of predetermined box shapes associated with it. Detection then consists of determining a score for each anchor box and regressing offsets with respect to the anchor boxes. Single-stage detectors tend to be more computationally efficient but have historically achieved lower accuracy than two-stage approaches.

3D detectors estimate the centroid and volume of objects using a 3D bounding box. Before deep networks were applied to this problem, traditional robotics pipelines for object detection involved several common steps. First, the ground plane is extracted. Second, the remaining points are clustered. Third, the clusters are classified using a classical machine learning approach such as a support vector machine. Finally, the objects are tracked using Kalman filtering techniques (Himmelsbach et al., 2008) (Moosmann et al., 2009).

As of this writing, the top approaches for 3D object detection fuse both vision and LIDAR data together to take advantage of both the accurate metric information that LIDAR provides and the semantic information from vision. Two such networks are Frustum PointNets (Qi et al., 2018) and AVOD (Ku et al., 2018).

---

[1] `www.autodrive.utoronto.ca/uoftped50`

AVOD uses a feature pyramid network to extract features from an image and a top-down representation of a pointcloud. These features are then fused by cropping and resizing the sections of the input feature space that correspond to the projected 3D anchor boxes. The network then generates a set of 3D proposals, repeats the crop and fusion step, and finally regresses a set of 3D boxes using fully connected layers.

Frustum PointNets first obtains a 2D bounding box using a vision-based detector. They then use a PointNet to process the points which fall within the bounding box when projected onto the image plane. The PointNet is trained to regress a single 3D bounding box from this subset of the pointcloud (Qi et al., 2017).

aUToTrack takes inspiration from Frustum PointNets by using a 2D detector to obtain initial object locations. This greatly reduces the dimensionality of the pointcloud and allows us to employ traditional robotics approaches in realtime. Our tracking approach takes inspiration from Bewley et al. (2016).

7.2   Clustering

We restrict our attention to points in front of the vehicle up to 40 m away, and 15 m to each side. A range of 40 m was determined experimentally to work well while also providing enough time for a vehicle travelling at our maximum speed of 40 km/h to stop within our desired maximum acceleration of 2 m/s^2. We subsequently segment and extract the ground plane using RANSAC, followed by Linear Least Squares for refinement. We smooth the ground plane parameters temporally using a Kalman filter. The remaining points are then transformed into the camera frame using a pre-calibrated extrinsic transformation matrix $\mathbf{T}_{cv}$. A corresponding set of image locations is obtained by projecting the LIDAR points onto the image using a perspective projection and the intrinsic camera matrix $\mathbf{K}$.

We retrieve the LIDAR points that lie inside each 2D bounding box. We then use PCL's Euclidean clustering on the corresponding 3D LIDAR points (Rusu and Cousins, 2011). Several heuristics are used to choose the best cluster. These heuristics include comparing the detected distance to the expected size of the object and counting the number of points per cluster. The position of the object is then computed as the centroid of the points in the cluster. This algorithm is given in pseudo-code below.

Algorithm 1 aUToTrack Clustering

Input: pointcloud $\mathbf{p} \in \mathcal{R}^{3 \times n}$ with n points, a set of 2D bounding box detections $\mathbf{B} \in \mathcal{R}^{4 \times m}$
Output: A list of K object centroids $\mathbf{y}_k = [x \ \ y \ \ z]^T$
  1: $\mathbf{p} \leftarrow \text{Passthrough}(\mathbf{p}, W, L, H)$
  2: $\mathbf{g} \leftarrow \text{Passthrough}(\mathbf{p}, W_2, L_2, H_2)$
  3: $(a, b, c, d) \leftarrow \text{RANSAC}(\mathbf{g})$
  4: $(\hat{a}, \hat{b}, \hat{c}, \hat{d}) \leftarrow \text{Kalman-Filter}(a, b, c, d)$
  5: $\mathbf{g} \leftarrow \text{inliers}(\mathbf{g}, \hat{a}, \hat{b}, \hat{c}, \hat{d})$
  6: $\{\mathbf{p}\} \leftarrow \{\mathbf{p}\} - \{\mathbf{g}\}$
  7: $\bar{\mathbf{p}} \leftarrow \mathbf{T}_{cv}\bar{\mathbf{p}}$
  8: $\mathbf{u} \leftarrow \mathbf{K}\bar{\mathbf{p}}/\bar{\mathbf{p}}_z$
  9: **for** $\mathbf{b} = (c_x, c_y, w, h) \in \mathbf{B}$ **do**
 10:     $\mathbf{p}_b \leftarrow \text{Passthrough}(\bar{\mathbf{p}}, \mathbf{u}, \mathbf{b})$
 11:     clusters $\leftarrow \text{Euclidean-Clustering}(\mathbf{p}_b)$
 12:     best-cluster $\leftarrow \text{Heuristics(clusters)}$
 13:     $\mathbf{y}_k = \text{centroid(best-cluster)}$
 14: **end for**

7.3   Tracker Setup

For each object, we keep a record of the state, $\hat{\mathbf{x}}$, covariance, $\hat{\mathbf{P}}$, class, object shape (w, l, h), confidence level, and counters for track management. The state is defined in Equation (2), where (x, y, z) is the position, $(\dot{x}, \dot{y})$ is the velocity within the ground plane. The position and velocity are tracked in a static map frame

external to the vehicle. The confidence is obtained from the 2D detector and filtered temporally.

A constant velocity motion model is used for all objects:

$$\mathbf{x} = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} \end{bmatrix}^T \tag{2}$$

$$\mathbf{y} = \begin{bmatrix} x & y & z \end{bmatrix}^T \tag{3}$$

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \boldsymbol{\omega} \tag{4}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \boldsymbol{n} \tag{5}$$

$$\boldsymbol{\omega} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \tag{6}$$

$$\boldsymbol{n} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \tag{7}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & T & 0 \\ 0 & 1 & 0 & 0 & T \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{9}$$

In this setup, $\mathbf{x}$ is the state of the object, $\mathbf{y}$ is the measurement, $\boldsymbol{\omega}$ is the system noise, $\boldsymbol{n}$ is the measurement noise, $\mathbf{A}$ is the state matrix, and $\mathbf{C}$ is the observation matrix. $\mathbf{Q}$ and $\mathbf{R}$ are the system noise and measurement noise covariances, which we assume to be diagonal. The diagonal entries of $\mathbf{Q}$ and $\mathbf{R}$ are used to tune the tracker for responsiveness vs. smoothness. The equations that describe the linear Kalman filter used here can be found in Section 3.3 in (Barfoot, 2017) The remaining 1D variables including the width, length, height, and confidence are filtered using an Alpha-Beta filter for each.

Table 1: Runtime of each component in aUToTrack. The total runtime is sum of one instance of SqueezeDet running on 16 threads followed by the clustering and tracking run sequentially on one thread each. The runtime of SqueezeDet on a 1080Ti GPU and Arria 10 FPGA are provided for comparison.

Component	Run Time	Hardware
SqueezeDet (Pedestrian Detection)	18 ms	NVIDIA GTX1080Ti GPU (For Comparison Only)
SqueezeDet (Pedestrian Detection)	26 ms	Arria 10 FPGA (For Comparison Only)
SqueezeDet (Pedestrian Detection)	**32 ms**	Intel Xeon E5-2699R (16 threads)
Clustering	**15 ms**	Intel Xeon E5-2699R (1 thread)
Tracker	**<1 ms**	Intel Xeon E5-2699R (1 thread)
Total Run Time:	**47 ms**	Intel Xeon E5-2699R Only

7.4   Data Association and Track Management

Our data association is based on metric information only. This works well for 3D objects such as pedestrians and cars that tend to be separated by significant distances. We use static gates to associate new detections to existing tracks. The gates are designed using the maximum possible inter-frame motion. Assuming a maximum speed of 5 m/s for pedestrians and a 0.1 s time step, we have a gating region of 0.5 m.

We used a greedy approach to associate measurements with tracked objects. For each tracked object, we evaluated the distance between the object and observations within its gate to find the nearest neighbor. The nearest neighbor is then assigned to the tracked object, and removed from the list.

We employed a strategy of greedy track creation and lazy deletion for managing tracks. In greedy track creation, every observation becomes a new track. However, every track must go through a trial period. While objects are in their trial period, they are removed from the list of objects being tracked if they miss a single frame. Once objects are promoted from their trial period, we count the number of consecutive frames

Table 2: Position and Velocity Estimation Error vs. Target Distance

Target Distance(m)	5	10	15	20	25	30	35
Position RMSE (m)	0.14	0.18	0.21	0.26	0.22	0.27	0.37
Velocity RMSE (m/s)	0.20	0.19	0.18	0.23	0.32	0.29	0.55

that an object has been unobserved for. In order for a non-trial track to be removed from the list, there must be no associated measurements for several consecutive frames. Due to the simplicity of the algorithms we employed, our pipeline runs exceptionally fast, as shown in Table 1.

7.5 Performance

In order to assess the performance of our approach prior to the competition, a dataset was collected with GPS-based ground truth for pedestrian motion. This new dataset, named UofTPed50, is used for benchmarking 3D object detection and tracking of a single pedestrian. UofTPed50 consists of 50 sequences of varying distance, trajectory shape, pedestrian appearance, and ego-vehicle velocities. Position and velocity information are reported in a static global reference frame. Hence, a stationary pedestrian corresponds to a velocity of 0 m/s.

In several sequences in UofTPed50, the pedestrian walks laterally from one side of the vehicle to the other at evenly spaced distances. Using these sequences, one can measure the impact of varying target distance on performance. We use Root Mean Squared Error (RMSE) as our error metric for both position and velocity estimation. As summarized in Table 2, our position and velocity estimation error tends to increase with distance. We achieve consistent velocity estimation accuracy up to 30 m, but the performance drops off around 35 m. This is potentially due to the impact of a poor calibration further from the LIDAR.

Figure 12 demonstrates a challenging Zig-Zag trajectory where the heading and velocity of the pedestrian changes quickly over time. aUToTrack is able to track the reference trajectory with reasonable accuracy. However, the position and velocity estimation appear to overshoot and lag behind the ground truth. This is likely due to estimator dynamics and could be addressed with parameter tuning.

It should be noted that the centroid estimation technique we use is not as accurate for cars. This is likely because the majority of the points being clustered are from the side of the object facing the LIDAR. For this reason, our centroid estimation may be off by up to 1 m for cars. For safe autonomous driving, a more accurate centroid estimate is likely required. Candidate approaches for 3D detection of vehicles include recent LIDAR-only approaches such as PointPillars (Lang et al., 2019) and PIXOR (Yang et al., 2018). Nevertheless, our approach is still accurate enough for the AutoDrive Challenge.
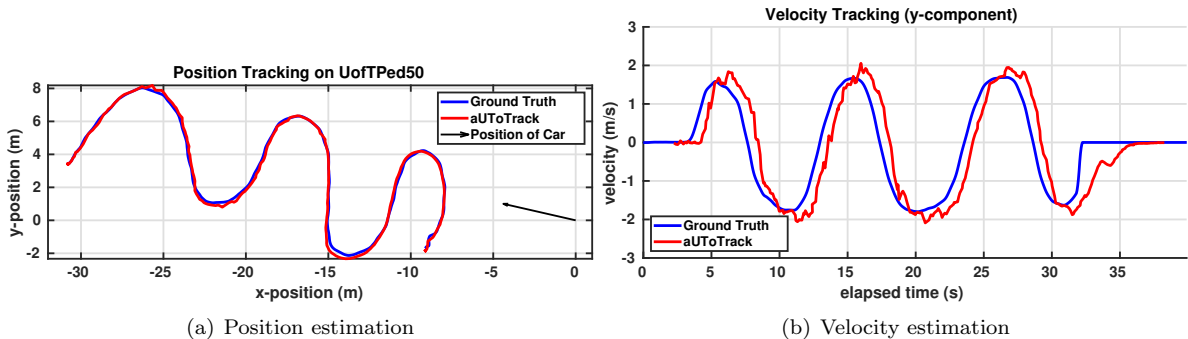

(a) Position estimation

(b) Velocity estimation

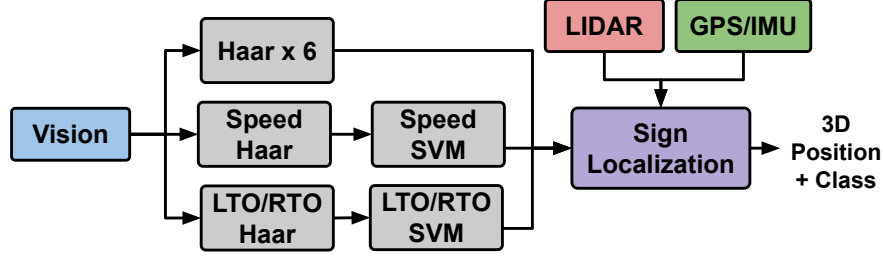Figure 12: Position and velocity estimation of the Zig-Zag scenario.

Figure 13: This figure depicts our traffic sign detection pipeline. A generic Haar cascade detector is used to identify speed limit signs and Left-Turn-Only vs. Right-Turn-Only signs. These detections are then classified by a support vector machine for each set. The other six sign classes are detected and classified by a single Haar cascade for each sign.

8 Traffic Light and Sign Detection

SqueezeDet was used to detect and classify the state of traffic lights. In order to achieve good performance, SqueezeDet was trained on a custom dataset of over 17000 images. Replicas of the competition traffic lights were used to optimize our detector for MCity. Only two traffic light states were trained on: red and green.

SqueezeDet is lightweight, but lacks the expressive power to accurately detect and classify 13 different traffic signs. For this reason, we opted to use Haar cascades instead (Viola and Jones, 2001). Traffic signs are generally easier to detect than pedestrians as their appearance remains consistent across viewpoint and illumination. For this reason, simpler detectors that take advantage of template matching work well.

In some cases, Haar cascades struggle to differentiate between signs with similar appearances. This included Left-Turn-Only (LTO) and Right-Turn-Only (RTO) text signs, and speed limit signs. To resolve this issue, sign detection was divided into two stages: an initial generic sign detector for LTO/RTO and speed limit signs followed by an SVM for classification. Figure 13 depicts this architecture. This method allowed us to exceed 90% precision and recall on our validation set. When tracking and pruning heuristics are included, performance improves 5-10%.

Once 2D bounding boxes for traffic lights and signs are obtained, they must be localized with respect to the vehicle so that higher-level decisions can be made by the planner. To localize signs, we use a pipeline that is very similar to our pedestrian detection. We project LIDAR points onto the image plane and extract the points that correspond to the sign's bounding box. We then run Euclidean clustering and return the cluster that is closest to the vehicle. By comparing the expected bounding box size with the detected bounding box, false positives can be pruned out. To smooth sign classes temporally, we treat signs as generic sign objects, and keep track of the previous 10 classes associated with that sign. When publishing the detected signs, we return the most common class observed within the last 10 detections of that sign. Temporal filtering is essential for smooth and reliable performance.

Traffic lights were included in the semantic map used at the competition. As the vehicle approaches an intersection, the expected positions of traffic lights in 3D space are projected onto the image plane. Figure 14 depicts this process. By associating raw detections with the expected positions, the exact position of traffic lights and their associated lane can be obtained easily from the map. To make these associations, we minimize a cost function based on Euclidean distance in the image plane and difference in heights. We also apply a maximum association distance of 3 m within the image plane. The corresponding pixel value will change depending on the expected distance to the upcoming lights.

We assume that all traffic lights are red by default. If there are several relevant traffic lights at an upcoming intersection, they must all be detected as green before the vehicle will proceed. We also keep track of the 10 previous detections associated with each expected traffic light to smooth detections temporally. This is a very cautious approach, but ensured that Zeus stopped at each red light at the competition.
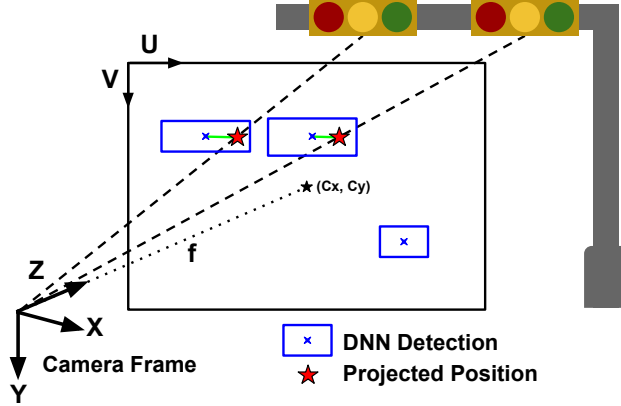
Figure 14: In this figure there are two upcoming traffic lights. Using our position in the semantic map, the locations of these traffic lights are projected onto the image plane (denoted as the red stars). The DNN has output two true positive detections and one false positive in this frame (denoted as blue boxes). The green lines denote the association between detections and expected positions.

A timer was included to prevent the vehicle from waiting at a set of traffic lights forever. In our case, the vehicle was set to wait for a maximum of 60 seconds. This timeout prevented the vehicle from becoming stuck during the Intersection Challenge. Heuristics like this were key to winning the competition.

Flashing red lights were a challenging aspect of the Year 2 competition. Initially, our approach was to keep track of the previous 20 detections, and to simply calculate the duty cycle of red vs. off to determine if the lights were flashing. In our case, if the duty cycle of red was between 35% and 65%, then a light was considered flashing. Since our traffic light detector was only trained to recognize red and green traffic lights, the class output for off traffic lights was close to random. To overcome this problem, we designed a hand-crafted computer vision algorithm based on OpenCV libraries to determine whether a traffic light was truly off. This hand-crafted method underperformed in varying illumination conditions and ultimately was not enabled during the competition. The lesson learned here is that 'off' traffic lights should be an additional traffic light state trained on in order to achieve good flashing light performance. At the competition, the Pedestrian Challenge was the only challenge to include flashing red lights. As such, we were able to simply treat flashing red lights as solid red lights with a 5 second timeout.
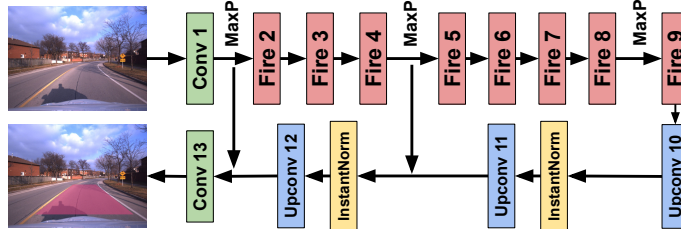

Figure 15: Our proposed semantic segmentation architecture.

9   Lane Detection

Zeus did not use lane detection at the Year 2 competition. However, due to the importance of this component in Year 1 and it's potential usage in future years of the competition, it continues to be an active area of research for aUToronto.

In the first year of the competition, no map was given; as such, lane detection was the sole source of localization information. This system needed to be reliable and robust to changing illumination conditions.

aUToronto developed three different lane detection approaches that each ran in realtime. These included steerable filters, a convolutional neural network, and a LIDAR-based approach (Burnett et al., 2018).

The Year 1 system was optimized for tight corners and lanes with quickly varying curvature. At this time, lane lines were always present where the vehicle was expected to drive. At the Year 2 competition, that system was no longer appropriate. Lanes can be faded in some locations and may not be present on some roads. Further, bad weather and illumination can make it difficult to detect lane lines consistently.

For Year 2, we originally planned to use GPS/IMU positioning and to use lane detection for lateral corrections. However, to use lane detection this way, it must be shown to be as reliable as the GPS/IMU. Otherwise, these corrections could add error to the position estimates. An altnerative way to use lane detection is to compare the output against the expected lane positions reported by the semantic map. This approach allows for the bias between the GPS coordinate system and the map coordinate system to be calibrated. In this case, lane detection does not need to be constantly running with a high reliability, but rather only needs to run occasionally with a few discrete measurements to obtain this bias correction. This means that the lane detection module can use a large, accurate, and potentially slow DNN for semantic segmentation. This project was moved to future work for the Year 3 competition and beyond.

In an attempt to achieve real-time performance on CPUs, we designed a lightweight segmentation network based on Squeeze-SegNet (Nanfack et al., 2018). Our modified structure is shown in Figure 15. The contracting layers in the network are initialized using SqueezeNet weights pre-trained on ImageNet. Due to its limited number of layers, SqueezeNet has a very small receptive field, making it difficult to distinguish the ego-lane from adjacent lanes. To combat this, some convolution layers have been replaced with dilated convolution layers. Further, all batch normalization layers were replaced with instance normalization layers to improve the robustness to intensity change and color shift.

We trained our model on the large BDD100k dataset and tested on BDD's evaluation set (Yu et al., 2018). We achieve an accuracy of 0.95 and an mIOU of 0.79 on BDD100k whereas our baseline Squeeze-SegNet resulted in an accuracy of 0.83 and an mIOU of 0.69. Furthermore, our approach is capable of running at up to 10 FPS on CPUs. Evaluation was conducted at 256 x 256 image resolution, which we found to be a suitable balance between accuracy and performance.

In many scenarios, the ego-lane is ill-defined such as at an intersection. These scenarios present difficulties for deep learning systems. Thus, we experimented with extending our model with an error prediction component based on Bayesian Deep Learning. Our model uses Monte Carlo dropout to calculate entropy as is done in Gal and Ghahramani (2016). Using this approach, entropy is high when the input sample is significantly different from the training samples. In these cases, the ego-lane tends to be ill-defined. Hence, a threshold on entropy can be used to identify where lane detection should be ignored, as shown in Figure 16.
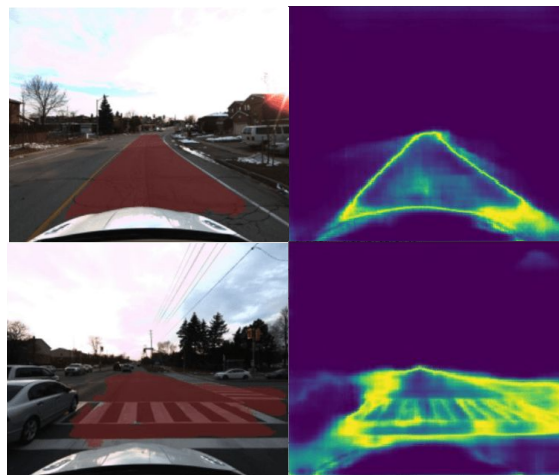

Figure 16: Top: the driving lane is clearly visible. The associated entropy is low. Bottom: the vehicle approaches an intersection. The entropy is high, indicating the lane detection output should be ignored.

(a) Post-processed LIDAR map of UTIAS by Applanix

(b) Trajectory output by LIDAR localization

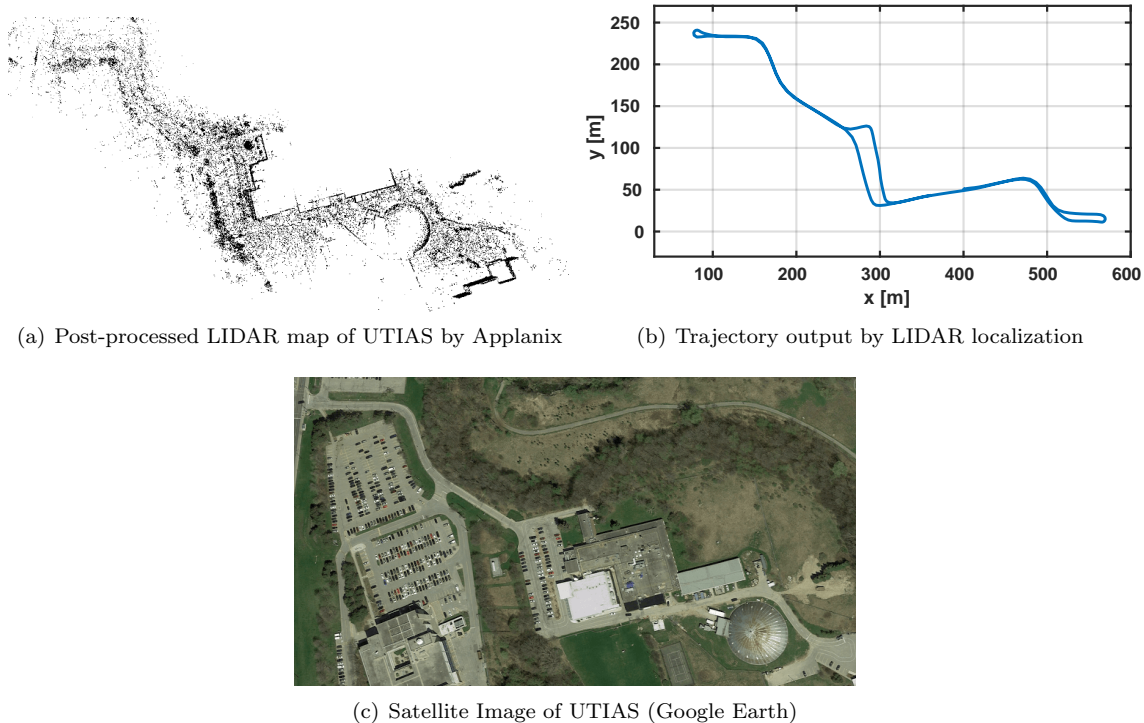(c) Satellite Image of UTIAS (Google Earth)

Figure 17: (a) key-frame-based LIDAR map built using Applanix software. (b) trajectory estimated using only LIDAR localization. (c) Satellite image of UTIAS.

10   Mapping

There are two types of maps we are concerned with: semantic maps and LIDAR maps. Semantic maps encode the location of lanes, traffic lights, and more. LIDAR maps may consist of a single aligned pointcloud for an entire area or a set of pointclouds with GPS coordinates. An example of a LIDAR map is shown in Figure 17 alongside the resulting trajectory estimate and Google Earth image for reference.

Semantic maps can be used to off-load a significant fraction of active perception to an initial map-building phase that can be done offline. This reduces the problem of self-driving from needing to perceive all features in realtime to only needing to perceive features that can't be encoded in a static map, such as the location of other traffic participants and the state of traffic lights.

In Year 2 of the competition, we made the assumption that our semantic map contained no major mistakes. Thus, we simply needed to localize ourselves within that map to take advantage of all the information it provided. In cases where the operational area is small, or a repeated route is used, this assumption generally holds. However, if the map that is being used encompasses an entire city, it becomes critical to relax this assumption and actively look for inconsistencies in the map such as a new construction zone.

Semantic maps are often represented using geometry such as points, lines, and polygons. The data formats provided to our team were not immediately conducive to running planning algorithms such as A*. It was helpful to convert these maps into a graph-based format to simplify the planning software. For this purpose, we developed a pipeline which allowed us to convert semantic maps into our own internal format which is based on (OpenStreetMap, 2020).

For the competition, we chose Carmera to be our map supplier. They report a low relative error between points in their map. Further, they supplied us with a LIDAR map of MCity which enabled us to use LIDAR localization.
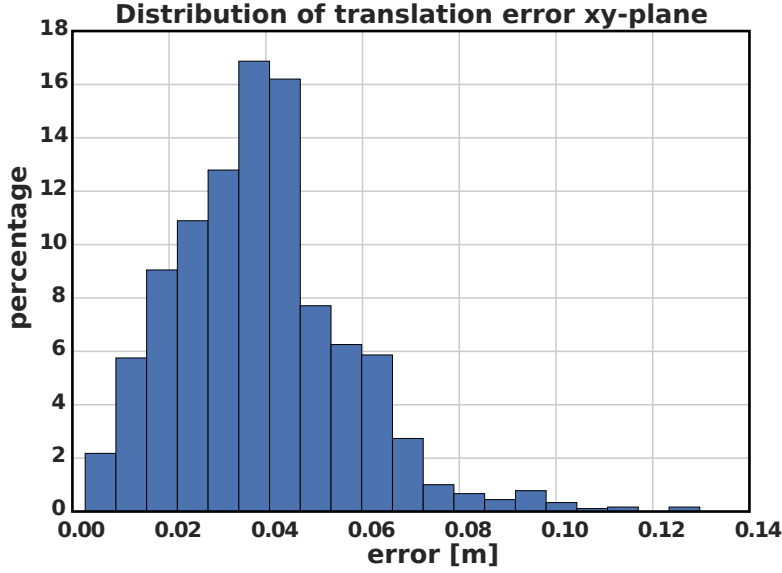
Figure 18: Position error histogram (xy-plane) of LIDAR-only localization along a trajectory (see Figure 17) compared to ground truth obtained from post-processed Applanix POSLV GPS/IMU output.

11   Localization

As mentioned in Section 9, lane detection was not used at the Year 2 competition. This was mainly due to concerns about reliability and the fact that lanes were not present everywhere at the Year 2 competition.

A key design problem in Year 2 required dealing with a potentially GPS-denied tunnel. Further, over the course of a year of testing we experienced several position jumps greater than 50 cm. In order to meet competition requirements, these problems needed to be addressed.

The two options we compared for localization at MCity were: LIDAR localization provided by Applanix, and GPS/IMU localization provided by Novatel (with a Terrastar satellite subscription).

One drawback to GNSS systems is that they require a clear view of the sky. During milestone testing in a heavily wooded area in Mississauga, our precision frequently dropped and took an excessive period of time to converge.

On the other hand, LIDAR localization presents numerous advantages. LIDAR is robust to ambient light change. In addition, Applanix's LIDAR localization is robust to minor changes in scene geometry such as moving vehicles. As long as large structures like buildings remain fixed, their LIDAR localization will work. Since the sensor data are relative to the vehicle, it does not suffer the same shortcomings as GNSS.

Figure 18 shows an error histogram plot for LIDAR localization. In this case, the ground truth is Applanix POSLV data post-processed with their POSPac suite. It should be noted that post-processing mitigates the shortcomings of GNSS by performing a batch optimization over the entire trajectory after a data-taking run has ended. Since the batch optimization incorporates both future and past data, it cannot be used in realtime. In this experiment, the majority of errors are under 10 cm and there were no errors over 14 cm. This is comparable to the accuracy reported by the Novatel GPS/IMU. The resulting LIDAR-based localization is not susceptible to GPS dropout.
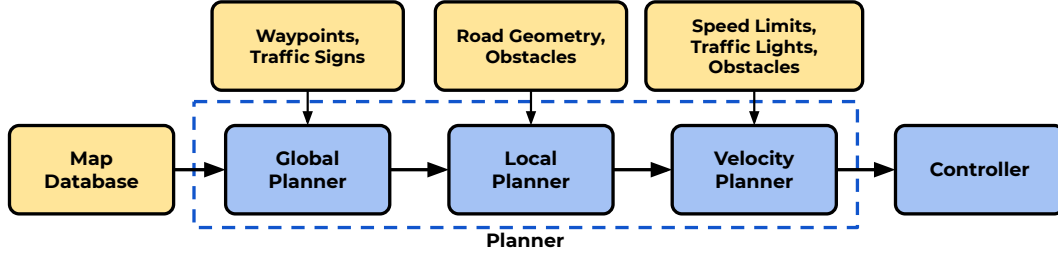
Figure 19: The hierarchical structure of our motion planner. The Global Planner outputs a high-level route, the Local Planner produces a path, and the Velocity Planner generates a velocity profile.

12 Planning

The Year 2 challenges included point-to-point navigation in an urban environment and complex driving scenarios. The planner needed to respond to external stimuli such as traffic lights, traffic signs and pedestrians. At the same time, the planner needed to generate maneuvers that are safe and comfortable in realtime. Our hierarchical planner consisted of a Global Planner, Local Planner and a Velocity Planner, as is shown in Figure 19. Similar hierarchical designs were deployed by the top-ranked teams in the DARPA Urban Challenge (Urmson et al., 2008).

12.1 Global Planner

The Global Planner operates on a connectivity-graph and selects future destinations depending on the current challenge. We classify the challenges into two groups: *Sign Following* and *Intersection Traversal*. Since the two groups require different map traversal strategies, two different global planning algorithms are implemented. Nevertheless, both algorithms aim to extract a subset of connected road segments from the semantic map to form a high-level mission plan.

Sign Following requires the vehicle to follow the direction of traffic signs: Left-Turn-Only, Right-Turn-Only, and Do-Not-Enter. The Global Planner first retrieves the vehicle's current road segment and the desired action from any detected traffic signs. The planner then looks up the successors of the current road segment. If a successor's direction and location coincides with a detected sign, the successor will be added to the plan. Otherwise, a successor is chosen based on a predetermined priority where straight has the highest priority. This process is repeated until the number of road segments in the plan reaches a predetermined limit or a terminating node in the map is reached.

The pull-in parking maneuver required its own specialized logic and maneuvers. The course was set up to end by driving through a region with diagonal parking spots on the right hand side. Some spots were occupied by vehicles and one spot had a handicap parking sign. The location and shape of the spots were contained in the semantic map. Perception nodes monitored the occupancy of each spot and the 3D location of handicap signs. Once an available spot was found, a pull-in parking maneuver was generated.

Intersection Traversal requires the vehicle to visit a list of waypoints at intersections. The planner aims to find an optimal (shortest) route that will guide the vehicle to visit all the waypoints in the correct order. To achieve an optimal path, we use a dynamic programming approach that utilizes A* as the backbone. The planner first finds the intersections enclosing the desired way-points. These intersections contain several road segments that enter and exit it. An algorithm based on A* that incorporates length, curvature, and lane change penalties is run between all exiting-entering road segment pairs between consecutive intersections. The length of each route is recorded to construct a new graph. Value iteration is executed on the new graph to find the optimal route. The Global Planner only needs to run once whenever the high-level mission is changed or the topological structure of the map is altered (e.g. a road is blocked). The purpose and output of the global planner is explained further in Figure 20.

Stop signs and railroad crossing both require the vehicle to stop and wait for a fixed period of time before

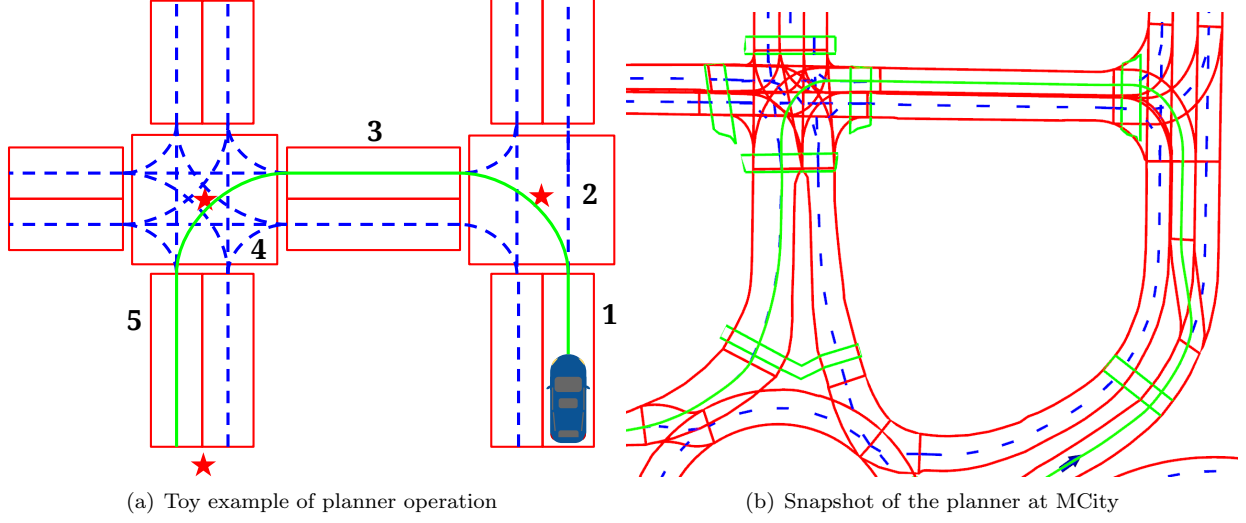(a) Toy example of planner operation   (b) Snapshot of the planner at MCity

Figure 20: (a) Our semantic maps consist of road segments (red polygons) and centerlines (blue dashed lines). The red stars represent high-level waypoints centered at the intersections that need to be visited. The global planner starts at the current vehicle's position and analyzes all the road segments between successive pairs of waypoints. The global planner uses dynamic programing and A* to determine the best sequence of road segments to take in order to reach each waypoint. The local planner then stitches together the centerlines and turning arcs of those road segments and performs minor smoothing to output the desired path (given as the green line). (b) using the same color scheme, this figure depicts the actual path planned by Zeus during the first pedestrian challenge starting at the bottom right (blue arrow).

proceeding. A pseudo-obstacle is used to force the vehicle to stop at the desired location. Pedestrians waiting to cross or actively crossing require the vehicle to stop. If pedestrians are waiting at the curb, we place a pseudo-obstacle before the crosswalk with a timeout of 5 seconds. If the pedestrian begins crossing within this time, a permanent pseudo-obstacle is placed until the pedestrian has reached the other side.

12.2 Local Planner

The Local Planner receives route information from the Global Planner. To achieve a safe and comfortable path, the Local Planner aims to generate a collision-free and low-curvature path. We use an optimization algorithm based on Montemerlo et al. (2008) that minimizes a nonlinear cost function E. The goal is to generate a path $\mathbf{P} = [\mathbf{x}_1 \ ... \ \mathbf{x}_N]$ that minimizes the cost function. Each pose in the path is defined as $\mathbf{x}_i = \begin{bmatrix} x_i & y_i & \theta_i \end{bmatrix}^T$ The cost function, which has three terms for each pose, is defined below:

$$E = \sum_{i=1}^{N} E_{i,cur} + E_{i,dev} + E_{i,obs} \tag{10}$$

The curvature term E_{cur} minimizes abrupt steering actions by penalizing heading change at each index of the path. The curvature term is defined below:

$$E_{i,cur} = w_{cur} \left(\frac{\Delta \theta_i}{|\Delta \mathbf{x}_i|} \right) \tag{11}$$

where w_{cur} is the weight of the curvature term, $\Delta \theta_i = \cos^{-1} \frac{\mathbf{v}_{i-1,i} \cdot \mathbf{v}_{i,i+1}}{|\mathbf{v}_{i-1,i}||\mathbf{v}_{i,i+1}|}$ is the heading change at $\mathbf{x}_i$, and $\mathbf{v}_{i-1,i} = \mathbf{x}_i - \mathbf{x}_{i-1}$, $\mathbf{v}_{i,i+1} = \mathbf{x}_{i+1} - \mathbf{x}_i$ are the position changes between $\mathbf{x}_{i-1}, \mathbf{x}_i$ and $\mathbf{x}_{i+1}$. The deviation
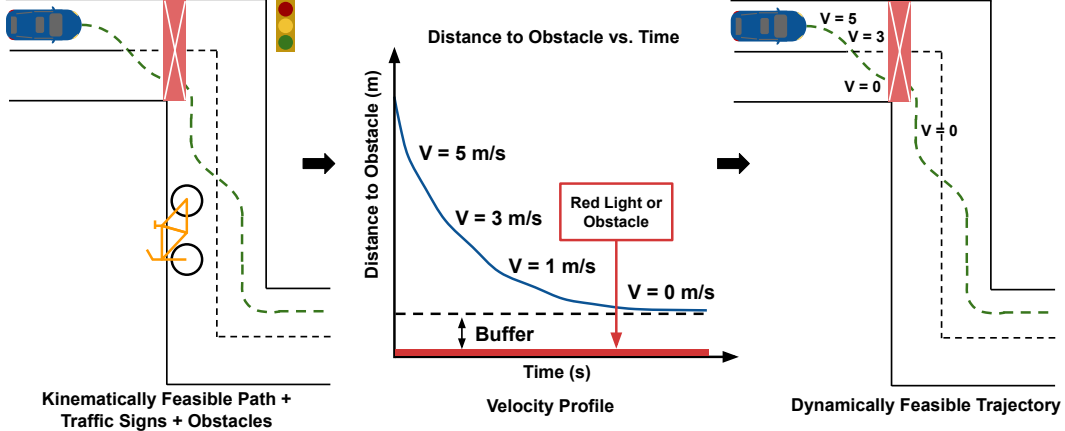
Figure 21: The Velocity Planner uses control points to manage velocity profile along the path.

term E_{dev} ensures the final path is close to the center of road by penalizing path deviation. The deviation term is defined below:

$$E_{i,dev} = w_{dev}|\mathbf{x}_i - \mathbf{c}_i| \tag{12}$$

where w_{dev} is the weight of the deviation term and $\mathbf{c}_i$ is the closest point to $\mathbf{x}_i$ on the road center line. The obstacle term keeps the path far from obstacles by penalizing components of the path that are within some threshold d from obstacles. The obstacle term is defined below:

$$
\begin{aligned}
E_{i,obs} &= 0 \quad \text{for} \quad |\mathbf{x}_i - \mathbf{o}_i| \geq d \\
E_{i,obs} &= w_{obs}\left(\frac{d_{obs}}{|\mathbf{x}_i - \mathbf{o}_i|} - 1\right)^2 \quad \text{for} \quad |\mathbf{x}_i - \mathbf{o}_i| < d
\end{aligned}
\tag{13}
$$

where w_{obs} is the weight of the obstacle term and $\mathbf{o}_i$ is the distance to the closest obstacle to $\mathbf{x}_i$. Since we did not need to avoid obstacles in Year 2, this term is ignored for faster computation. Finally, the optimization problem can be solved iteratively using gradient descent. The initial guess $\mathbf{P}^{(0)}$ can be generated by connecting the center-lines of the road segments. The gradient descent formulation is given below:

$$\mathbf{P}^{(t+1)} = \mathbf{P}^{(t)} - \eta\frac{\partial E(\mathbf{P}^{(t)})}{\partial \mathbf{P}^{(t)}} \tag{14}$$

The gradient is approximated using the equation below:

$$\frac{\partial E\left(\mathbf{P}^{(t)}\right)}{\partial \mathbf{x}_i^{(t)}} \approx \frac{E\left(\mathbf{P}^{(t)} + \Delta\mathbf{P}_i\right) - E\left(\mathbf{P}^{(t)} - \Delta\mathbf{P}_i\right)}{2\Delta\mathbf{x}_i^{(t)}} \tag{15}$$

where $\Delta\mathbf{P}_i$ is a small disturbance to $\mathbf{x}_i$. The Local Planner only needs to run once when a new route is generated by the Global Planner or when an object detection is received.
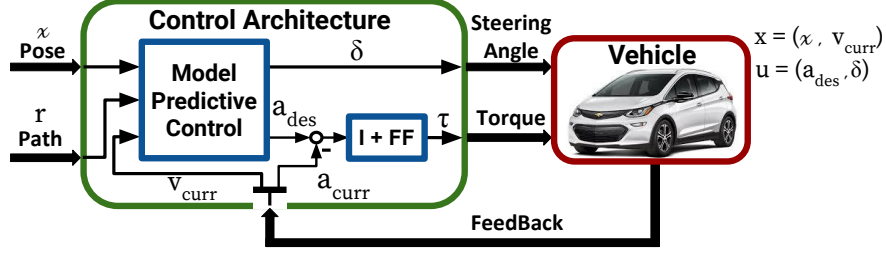
Figure 22: Control architecture for Zeus. x: current vehicle state. u: control input.

12.3 Velocity Planner

The Velocity Planner assigns a speed to each pose in the path generated by the Local Planner. Efficient online velocity profiling is achieved by placing control points along the path and connecting them with constant-acceleration motion primitives. For example, when the vehicle needs to stop for a red light, the Velocity Planner places a 0-speed control point at the stop line. An illustration of this process is shown in Figure 21.

13 Control

The high-level architecture of our controller is shown in Figure 22. The inputs to the controller are a desired path, velocity profile, and the current vehicle state. The controller then outputs steering wheel angle δ and torque τ for the vehicle. We designed a Nonlinear Model Predictive Controller (NMPC) for this purpose. To reject disturbances such as slopes, we added an integral controller on acceleration error in addition to a feed-forward term that directly maps the desired acceleration to torque.

Compared with other control approaches, such as feedback linearization (Novel et al., 1995), MPC offers several advantages for autonomous driving. First, MPC only makes mild assumptions about the motion model which allows it to be generalized to more complex models required in some challenging driving scenarios (Paden et al., 2016). Second, by decoupling controller design from vehicle modelling, less effort is required to improve upon a design. Lastly, MPC makes it easier to incorporate motion constraints. This allows it to avoid the common pitfalls that impact other control approaches.

MPC also presents a couple of drawbacks. Stability is not guaranteed unless stronger assumptions about the motion model are made. However, this turns out to be less of an issue in practice. MPC also presents a higher computational cost than other control approaches, but this was not an issue for our team due to our ample compute resources on-board.

13.1 NMPC Formulation

NMPC aims to track a timed reference signal $\mathbf{r}$ derived from the desired path given by the planner. NMPC predicts how vehicle states $\mathbf{x}$ will evolve over a finite time window of size N given control commands $\mathbf{u}$, nonlinear motion model $\mathbf{F}$ and current state $\mathbf{x}_o$. Using the predicted states, NPMC evaluates the cost of the control effort $\mathbf{J}_u$ and the tracking errors resulting from the predicted states $\mathbf{J}_e$. It obtains the best control input sequence $\mathbf{u}^*$ by minimizing the cost function while respecting the vehicle state and control command constraints $\mathbf{H}$. Following the approach in Paden et al. (2016), we formulate this problem as a nonlinear constrained optimization given as:

$$\min_{\mathbf{x}_n, \mathbf{u}_n} \sum_{n=k+1}^{k+N} \mathbf{J}_e(\mathbf{x}_n, \mathbf{r}_n) + \sum_{n=k}^{k+N-1} \mathbf{J}_u(\mathbf{u}_n, \mathbf{u}_{n-1})$$

$$\text{s.t.} \quad \mathbf{x}_{n+1} = \mathbf{F}(\mathbf{x}_n, \mathbf{u}_n), \quad n = k, k+1 \ldots k+N-1, \tag{16}$$

$$\mathbf{H}(\mathbf{x}_{k+1:k+N}, \mathbf{u}_{k:k+N-1}) \leq 0,$$

$$\mathbf{x}_k = \mathbf{x}_o, \ \mathbf{u}_{k-1} = \mathbf{u}_{k-1}^*$$

where the scalar subscripts are time indices. In a fashion known as *receding horizon*, only the first optimal command of the sequence (i.e. $\mathbf{u}_k^*$) is applied and the process is repeated at the next control time step.

A simple 2D kinematic bicycle model was used, whose state vector $\mathbf{x} := [x \ y \ v \ \theta]^T$ consists of planar position, speed and heading. The control input vector $\mathbf{u} := [a \ \delta]^T$ consists of acceleration and steering angle. The cost functions for tracking error and control effort are given below:

$$\mathbf{J}_e(\mathbf{x}_n, \mathbf{r}_n) = (\mathbf{r}_n - \mathbf{x}_n)^T \mathbf{Q}_n (\mathbf{r}_n - \mathbf{x}_n),$$
$$\mathbf{J}_u(\mathbf{u}_n, \mathbf{u}_{n-1}) = (\mathbf{u}_n - \mathbf{u}_{n-1})^T \mathbf{S}_n (\mathbf{u}_n - \mathbf{u}_{n-1}) \tag{17}$$

where tracking error is defined as the Euclidean distance between vehicle state $\mathbf{x}$ and its desired state $\mathbf{r}$. The rate of change of control inputs is interpreted as the control effort. $\mathbf{Q}_n$ and $\mathbf{S}_n$ are positive semi-definite diagonal matrices trading off the relative importance of each element in their associated vectors. $\mathbf{H}$ includes the linear constraints on velocity, longitudinal acceleration, jerk, and steering angle at each predicted time step. $\mathbf{H}$ also includes nonlinear constraints on lateral acceleration.

13.2 Sequential Quadratic Programming

We use Sequential Quadratic Programming (SQP) to solve a nonlinear optimization problem as in Carvalho et al. (2013). SQP is an iterative approach where a sequence of quadratic programming problems is constructed and solved until the solution converges. To put (16) into quadratic form, we linearize the motion model and nonlinear constraints in $\mathbf{H}$ about the optimal solution $(\mathbf{x}_n^*, \mathbf{u}_n^*)$ found in the previous iteration. In the first iteration of SQP, we use the solution from the previous control time step. The resulting quadratic problems can be solved efficiently using an off-the-shelf solver.

13.3 Timed Reference Generation

Our NMPC formulation (16) is a trajectory tracking controller that requires a timed reference signal $\{\mathbf{r}_n\}_{n=k+1}^{k+N}$, which can be derived from the reference path and its associated velocity profile. We set $\mathbf{r}_{k+1}$ to be the closest waypoint on the desired path and simulate forward using a kinematic bicycle model to obtain subsequent desired states.

Our formulation provides a straightforward way to tune parameters, but it can still be tricky in practice. A bad choice of parameters can lead to jerky commands, poor tracking performance, or instability. A good starting point is to make sure that the structure of the cost function is well-understood.

Table 3: Year 2 Competition Results

University	Total Points
University of Toronto	**885**
North Carolina A&T State University	523
Texas A&M University	515
Michigan Technological University	471
Kettering University	437
Virginia Tech	430
Michigan State University	352
University of Waterloo	330

14 Year 2 Competition Performance

Zeus placed first in each dynamic challenge by a significant margin. U of T also placed first in all but two static event categories. This included social responsibility, concept design presentation, a simulation challenge, and a mapping challenge. The static non-driving events constituted 60% of the 1000 total possible points for Year 2. The total points resulting from the competition are shown in Table 3.

The first two days of the competition consisted of unpacking, safety inspections, and the installation of OXTS GNSS systems for scoring. The OXTS systems were mounted in each vehicle to monitor kinematic variables, lane boundary crossings, and whether the vehicle stopped appropriately. These data were analyzed by judges in order to assign points. In general, points were awarded for performing the expected behavior, such as attaining a speed limit or waiting for a pedestrian to cross the road. Points were subtracted for breaking Ann Arbor driving code or exceeding the prescribed kinematic envelope.

The third day of the competition included one hour of practice time in MCity. This was the only time that teams were given to tune algorithms. Zeus did not work perfectly during this practice run and several last-minute bug fixes were required. The most notable bug was an unforeseen error in LIDAR localization. This was addressed by switching to GPS/IMU localization and using the output of LIDAR localization to calibrate the offset between the Novatel GPS output and the desired position in the Carmera map frame. This issue is outlined in more detail in Section 14.6.

The last three days of the competition included the Traffic Control Sign Challenge, MCity and Pedestrian Challenge, and the Intersection Challenge, respectively.

The remainder of this section provides details on Zeus' performance in each dynamic challenge. In doing so, we will describe some of the noteworthy errors our system experienced and the subsequent lessons learned. We will also discuss what did and did not go well, last-minute bug fixes, and our perspective on why we think we won.

14.1 Speed Zone Challenge

The speed zone challenge required vehicles to drive straight along a section of highway and abide by the posted speed limits while staying within the lane lines. The speed limit positions were not encoded in the semantic map and as such had to be actively perceived and localized. At the competition, there were only two speed limit signs: a 20 mph (9 m/s) sign followed by a 15 mph (7 m/s) sign. Note that our default driving speed at the start of the course was 5 m/s. In order to receive full points, vehicles were required to reach the posted speed limit within 30 ft of the sign's position along the road.

Figure 23 depicts Zeus' performance during the Speed Zone Challenge. At the start of the challenge, Zeus was not aligned properly in the lane and had to correct itself. Then, Zeus accelerated up to a default speed
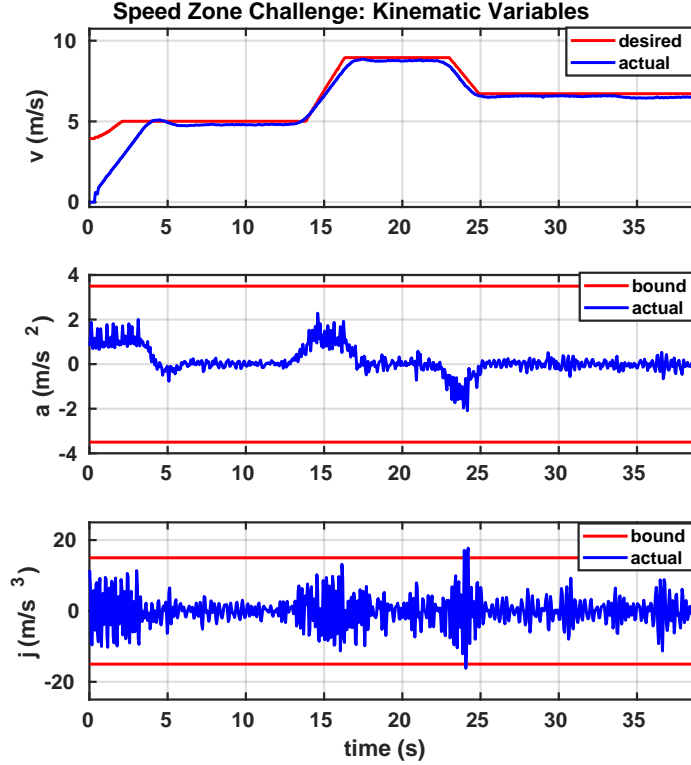
Figure 23: This figure depicts Zeus' performance during the Speed Zone Challenge. The top figure shows the desired velocity alongside the actual velocity of the vehicle. The middle and bottom figures show that we stayed within our acceleration and jerk constraints for the entire course. This data were extracted from Zeus' own sensors during this challenge.

of 5 m/s. Zeus followed the center of the lane quite closely during the challenge but with a constant bias. We determined that this constant was due to both an error in our GPS/IMU offset calibration as well as a bug in the controller. Fortunately, the vehicle was still able to drive within 10 cm of the centerline for the entire challenge. Figure 23 shows that Zeus reached both speed limits correctly: 19.7 mph for the first speed limit and 14.8 mph at the second speed limit.

In order to detect speed limit signs sufficiently far in advance, we relied on an additional forward-facing camera with a 16-mm lens and 30-degree field of view. This camera effectively doubled the range of our sign detector by simply having a higher resolution further from the vehicle. At the competition, speed limit signs were detected over 80 m away, although we limited the detection range to 50 m due to the sparsity of LIDAR points beyond this distance.

Zeus received the maximum number of points for this challenge by staying within the prescribed acceleration and jerk envelope and attaining the posted limits within the required distance. Zeus stayed within the starting lane for the entire challenge.
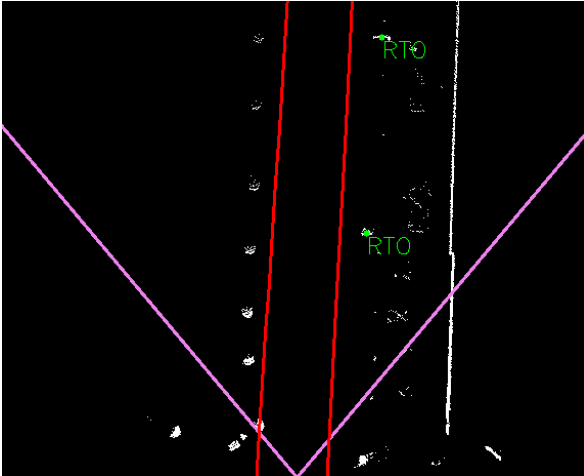
14.2 Traffic Control Sign Challenge

The Traffic Control Sign Challenge required the vehicle to continue straight unless directed otherwise by signs present on the road. At the competition, this challenge started with two semantically equivalent but visually different Right-Turn-Only (RTO) signs. Zeus correctly detected and localized these signs from the start line. After seeing these signs, our global planner triggered a re-plan to turn right at the upcoming

(a) Perspective View of Sign Detection

(b) Perspective View of Parking Spots

(c) Bird's Eye View of Sign Localization

(d) Bird's Eye View of Parking Occupancy

Figure 24: (a) Perspective view of signs detected in image. (c) Position of signs relative to the vehicle. (b) A handicap parking sign was detected and associated with a parking spot. (d) The bird's eye visualization of the parking occupancy node.

intersection and then continue driving straight. Figure 24 (a,c) depicts the sign detection at the start of the challenge from both the perspective view and bird's eye view. Figure 24 (a) depicts the raw detections in red and the tracked detections in green. For the second sign, the pointcloud-to-detection association is a little off, causing it to appear misaligned in the perspective image. The bottom image depicts the localization of both signs relative to the vehicle. The red lines correspond to the ego-lane and the purple lines are the field of view of the main camera.

Due to some careful analysis of the rules and the layout of MCity, we determined that if Zeus drove through the parking section during this challenge, it should automatically start looking for a valid parking spot. Figure 24 (b) depicts the sign detection and parking occupancy once Zeus rounded the first corner. Two of the parking spots were occupied with dummy cars. A third parking spot was blocked by the presence of a handicap parking sign.

Upon turning the corner, our occupancy detection node began counting the number of points in each spot. Due to our temporal smoothing setup, each spot starts in an *unknown* state and requires several *free* detections before being marked as unoccupied. Because of this, Zeus almost did not detect a free parking spot in time. Figure 24 (d) depicts the output of the occupancy node after enough detections had been received to correctly determine which spots were occupied. At this point, a specialized planner generated a trajectory that would place Zeus in the center of a spot.

During our one-hour practice, the parking maneuver did not initiate due to a bug in our planner. Hence, this
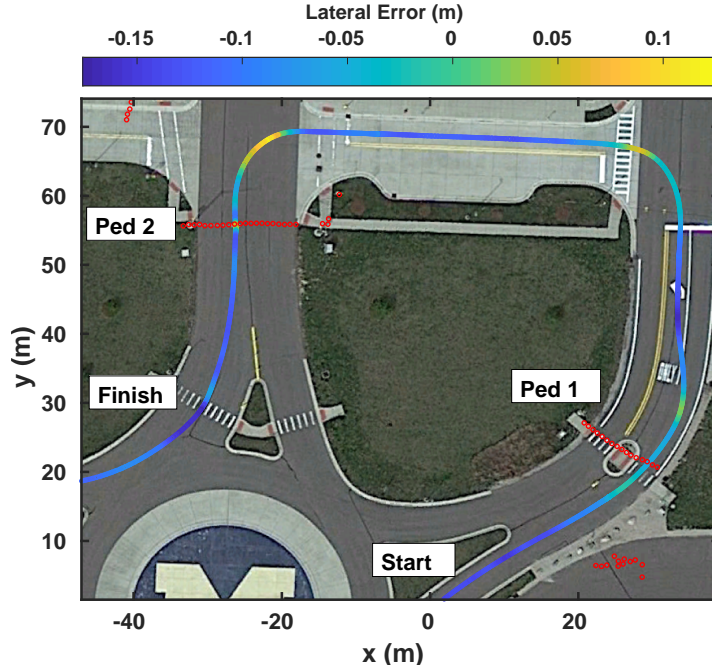
Figure 25: This figure depicts Zeus' trajectory during Pedestrian Challenge course 1. The color of the trajectory corresponds to lateral error. The red dots correspond to detected pedestrian locations. Note that the satellite image is meant to provide context only as it does not align perfectly with the map. Also note that the error does not have a mean of zero. This was due to a bug in our controller at MCity.

maneuver had not yet been tested at MCity before this challenge. Several tuning parameters that had been validated at the University of Toronto were modified slightly given the shape of the spots at MCity and the results of simulation tests. These simulation tests were conducted using our own internal simulation tools (using C++ and ROS) which simply create a model of the vehicle for the controller to interact with and allow the vehicle to move around within the semantic map. This simple simulator allows us to test planning logic and validate that the paths output by the planner are smooth and correct. Our desired velocity was lowered to 1.5 m/s during the parking maneuver to ensure that our controller would perform with high accuracy. Given these last-minute changes and a little luck, we were able to park almost perfectly during the competition run. Zeus received the maximum possible points for this challenge.

14.3 Pedestrian Challenge

The Pedestrian Challenge required vehicles to sequence several intersections while abiding by traffic lights and reacting appropriately to pedestrians attempting to cross the road. This challenge was divided into two separate courses. Figure 25 depicts the path Zeus took during the first course. The path is colored by the lateral error at each point along the path. Note that Zeus stays quite close to the centerline. Also shown in Figure 25 are the detected pedestrian positions in red. This minimal number of detections outside the crosswalk regions shows that our pedestrian detection has a very low false positive rate. On the bottom-right and top-left of Figure 25, extraneous detections correspond to by-standers and test crew. There were only a handful of false positives in the two courses.

The first pedestrian encountered was a child-sized dummy. Although we did not explicitly train on this target, it was visually quite similar to the adult-sized dummy that we had trained on. Hence, our 2D DNN was still able to detect it. However, due to its size and position on the road, the child-sized dummy was not detected until it was within 15 m of the vehicle. In order to give Zeus ample time to detect and react to these pedestrians, our desired speed for this challenge was lowered to 3 m/s.

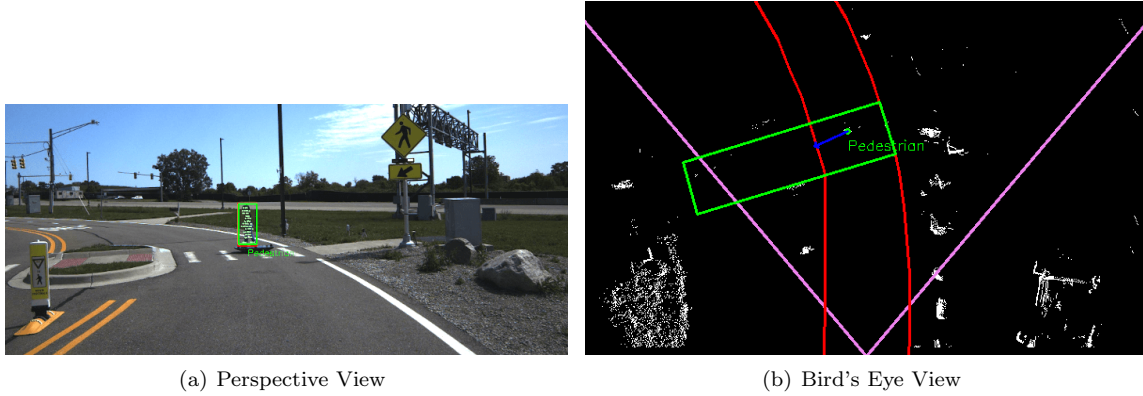(a) Perspective View   (b) Bird's Eye View

Figure 26: This figure depicts a pedestrian detection from both the perspective and bird's eye view.

Zeus' pedestrian detection and tracking worked largely as expected during this challenge. Figure 26 depicts the child-sized dummy being tracked in the perspective and bird's eye view. In the perspective view, the tracked bounding box is green and the green dots are the LIDAR points associated with the pedestrian. In the bird's eye view, the position of the pedestrian relative to the vehicle is shown as a green dot. The red lines are the ego-lane, the crosswalk is highlighted in green, and the field of view of the main camera is shown in purple. The blue line attached to the pedestrian indicates its velocity vector.

The child-sized dummy simply walked from one side of the road to the other in front of the vehicle. The second pedestrian involved a more complex scenario. In this case, the pedestrian was set up to walk towards Zeus on the left-hand crosswalk while Zeus was expected to turn left at the intersection. In this case, more than one crosswalk needed to be scanned and a greater detection range was required. The second pedestrian was initially detected over 40 m away.

As mentioned in section 8, our traffic light detector had undefined behavior for traffic lights in the 'off' state. In order to handle flashing red lights, we simply treated them as solid red lights with a short five-second timeout. This worked quite well during the pedestrian challenge but caused us to perform a rolling stop at the first intersection and lose five points accordingly.

After passing the second pedestrian, a pedestrian detection was reacquired by a corner camera halfway through the intersection. Due to a bug in our planner, this caused us to stop unexpectedly before the next crosswalk. This error caused us to violate our jerk constraints and lose a point.

The second course used the same intersections and pedestrian locations but in the reverse order. Zeus tracked both pedestrians without error and experienced only a single false positive. Zeus received the maximum possible points for the second course.

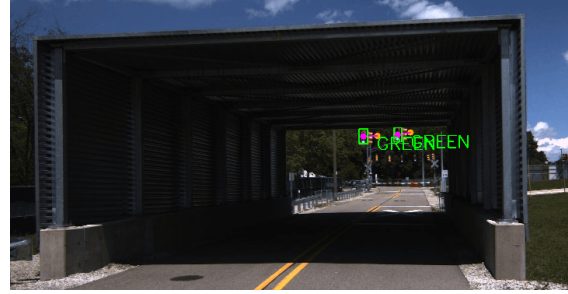### 14.4   Intersection Challenge

During the Intersection Challenge, vehicles were required to sequence 13 intersections while abiding by traffic lights. Zeus' traffic light detection performed largely as expected during this challenge. We believe the success of this system is due to the massive custom dataset that was collected as well as the large amount of testing leading up to the competition. Despite this success, there was an interesting failure case that is worth discussion.

At the second intersection, Zeus became stuck for two minutes before proceeding. The two factors that contributed to this error were the non-maximum suppression (NMS) threshold for SqueezeDet and an offset in our GPS-based positioning. Figure 27 depicts the traffic light detections at this intersection once the lights turned green. By setting the NMS threshold too high, detections that would have otherwise been returned were suppressed. Further, it can be seen in Figure 27(a) that the there is a significant offset between the

(a) Traffic Light Detection Error

(b) Before Tunnel - Traffic Light Detection

(c) Associated Raw Detections

(d) Inside Tunnel - Traffic Light Detection

Figure 27: Left: Zeus was stuck at this intersection for two minutes. (a) There are three traffic lights the upcoming intersection. Only one has been properly detected but it has been associated with the wrong expected light position. (c) This image shows the root of the problem: the non-maximum suppression threshold has been set too high. This high threshold, coupled with the large number of traffic lights in the frame, caused some potential detections to be discarded. Right: Traffic Light Detection experiences an error when entering a tunnel. (b) Before entering the tunnel, the light is correctly classified as green. (d) The area outside the tunnel becomes overexposed and causes the traffic light to be incorrectly classified as red.

expected traffic light position (orange dot) and the detected position (green box). Both of these factors contributed to only one of the traffic lights being detected as green.

Ordinarily, we require all traffic lights at an upcoming intersection to be detected as green before proceeding. Figure 27(a) shows that only one of the traffic lights has been detected and classified as green. Because of this, the other traffic light is assumed to be red, which prevents the vehicle from proceeding. Normally, a 60-second timeout would prevent Zeus from becoming stuck at an intersection. In this case, SqueezeDet occasionally output the correct detections, allowing Zeus to start moving forward. However, the pitching of the vehicle caused the image to change sufficiently for our DNN to incorrectly classify the lights again and cause Zeus to hit the brakes. This process repeated until the vehicle crossed the stop-line, after which traffic light detections were ignored and the vehicle proceeded through the intersection.

The lesson here is that although increasing an NMS threshold can reduce noise, it leads to more false negatives. For several perception tasks in self-driving such as object detection, false negatives can be more detrimental than false positives. As such, a good approach should bias towards more false positives and filter these out using tracking and semantic information. Another lesson is that requiring all traffic lights to be detected as green may be too cautious for real-world driving. There are many scenarios in which several redundant traffic lights are present. In these cases, it is sufficient to detect a subset of the lights as green.

Zeus lost two points for exceeding jerk constraints and another eight points for not stopping behind the stop-line at four different intersections. In one of these cases, Zeus stopped after the stop-line due to an error in our semantic map. Zeus received the most points during this challenge by a significant margin.

(a) Railroad Crossing

(b) Cyclist

(c) Deer Crash

Figure 28: MCity Challenge Obstacles included a tunnel, a railroad crossing, a static cyclist, and a dynamic deer. (c) depicts the moment right before the dummy deer collided with the side of Zeus.

14.5 MCity Challenge

The MCity Challenge required vehicles to sequence several intersections while handling various obstacles along the way. These obstacles included a tunnel, railroad crossing, static cyclist, and dynamic animal. Figure 28 depicts the railroad crossing and static cyclist.

Zeus drove through the 10 m long and 6 m wide tunnel mostly without issues. The Novatel GPS/IMU positioning did not jump in the tunnel as originally expected and Zeus stayed close to the centerline throughout. However, the traffic light detector did experience a noteworthy error. This error is depicted in Figure 27 (b,d). Before entering the tunnel, we detected the oncoming traffic lights as green. However, once we entered the tunnel, the auto-exposure of the camera caused the region outside the tunnel to become over-exposed. This caused the traffic light detector to erroneously report the traffic lights as red. This in turn caused Zeus to slow down unnecessarily in the tunnel.

The lesson here is that handling changing illumination conditions while going through tunnels is a challenging problem. One potential solution to this problem could include having cameras with better dynamic range or using multiple cameras with different exposure settings. Another solution could take advantage of semantic information to schedule camera exposure values to handle tunnels better.

The second obstacle encountered was a railroad crossing. MCity has a fully-functional railroad crossing with moving arms and flashing lights. aUToronto ran out of time before the competition to develop perception software to handle railroads. It was not possible to test railroad handling during the 1-hour practice period. Thus, we did not know whether our secondary obstacle detection would be sufficient to detect the railroad arm. As such, we made a last-minute decision to circumvent railroad detection completely by relying on a simple timed stop instead. This strategy allowed us to stop at the railroad but caused us to wait much longer than necessary after the arms lifted. None of the other AutoDrive teams made it past the railroad during the MCity Challenge.

The third obstacle was a static cyclist in a bike lane. Since there are only two bike lanes at MCity, we simply shifted our centerline over to the left for those regions. This centerline shift made it so that at least 1 m of clearance would be provided to a cyclist in the bike lane, thus satisfying competition requirements.

The fourth and final obstacle was a dynamic deer. The deer was positioned on the grass at the side of the road. The deer started moving when it was within 10 m of Zeus. When the deer entered the road, Zeus was travelling 4 m/s and the deer was 6 m away laterally and 5 m away longitudinally. The deer then collided with the side of Zeus, forcing the safety driver to perform a manual takeover. Figure 28 (c) depicts the moment immediately before the collision.

Inspection of recorded data showed that our secondary obstacle detector did not detect the deer at all. There are several reasons for this. Early on in development, we made the incorrect assumption that the dynamic obstacles we would be required to handle would be in front of Zeus and in the ego-lane. Because of this assumption, we ignored LIDAR points outside the ego-lane or points behind the bumper longitudinally. The deer's trajectory was such that it was outside the ego-lane until the last moment, where it was already behind the front bumper longitudinally.

There are several lessons that can be drawn from this collision. The first is that our secondary obstacle detection had too many assumptions built into it. In the future, it will be important to track objects over a much larger region around the vehicle even if the competition does not explicitly require it. The second lesson revolves around prediction. Even if we had detected the deer, the collision would likely still have occurred. This is because our perception system currently does not predict the future motion of objects. Thus, a key component that we will develop in the future will enable Zeus to extrapolate the future positions of objects in order to predict potential collisions and react accordingly.

14.6 Discussion

Zeus completed three out of the four challenges without requiring a manual takeover. Even though the perception components such as object detection and traffic light detection experienced some faults, they operated as expected during the competition. False positives in object detection were filtered out using the semantic information of lane and crosswalk locations. Errors in traffic light classification were overcome using timeouts, ensuring Zeus would complete each challenge to maximize points.

A significant portion of the dynamic challenge points was allocated to remaining within lane lines, stopping accurately, and staying within a kinematic envelope. Our MPC controller kept us close to the reference trajectory and within these kinematic constraints for most of the competition. Even driving at higher speeds (40 km/h), the vehicle drove smoothly with no sudden movements.

In the previous sections, we identified some of the faults encountered by Zeus during each challenge. In the Traffic Control Sign Challenge, our parking occupancy software took too long to determine the occupancy of each spot, which almost prevented a parking maneuver. During the Pedestrian Challenge, Zeus committed a rolling stop and stopped unexpectedly during the first course. During the Intersection Challenge, Zeus became temporarily stuck at the second intersection due to a localization offset and a non-maximum suppression threshold. In the MCity Challenge, Zeus slowed down in the tunnel, relied on a hard-coded timeout to pass the railroad, and was hit by the dynamic deer.

By highlighting these faults in our system, our goal was to outline some of the weaknesses of the approaches that we employed. Although Zeus performed the best out of all the AutoDrive Challenge teams, these faults demonstrate that there is still a significant amount of work left in order to bring Zeus to Level 4 autonomy.

It is also important to note that Zeus required several last-minute bug fixes in order to perform well during the competition runs. During our 1-hour practice time, several bugs in the planner and LIDAR localization were uncovered. The planner bugs included issues with runtime performance, red light timeouts, railroad crossings,

and pedestrian handling. Most notably, Zeus did not autonomously park during our 1-hour practice, which prevented us from tuning parameters before the competition run. Possibly the most significant bug uncovered during this 1-hour practice was that LIDAR localization did not work as expected. In some regions of MCity, the localization output experienced jumps.

As mentioned in section 11, we had initially planned to use Applanix's LIDAR localization at the competition. Since access to MCity prior to the competition was prohibited, we acquired a LIDAR map from Carmera. This LIDAR map was in the form of a large aligned pointcloud for all of MCity. This LIDAR map needed to be converted to Applanix's map format in order to run their localization software.

Our assessment is that our process of converting Carmera's map into Applanix's format resulted in the errors that were observed during the practice run. Ideally, LIDAR maps are custom-built using Applanix's own software. This is what we do at the University of Toronto and the resulting position estimates are exceptionally reliable. Unfortunately, there was not adequate time during the one hour of practice to build a LIDAR map and test autonomous functions.

Due to this issue with LIDAR localization, we were forced to use Novatel's GPS/IMU positioning instead. Our primary concern was that the Novatel positioning was biased with respect to the semantic map provided by Carmera. In order to calibrate for this bias, we compared our GPS/IMU positioning against LIDAR localization during the 1-hour practice run. It is important to note that LIDAR localization returns a relative measurement of one's position in a LIDAR map, whereas GPS/IMU positioning returns a global measurement of one's position on the Earth. For this reason, it was possible to treat LIDAR localization as a 'ground truth' to calculate the bias in our GPS/IMU positioning. The bias we calculated was quite large: 70 cm in Easting, 90 cm in Northing, and 120 cm in altitude. Without this bias correction, we may not have been able to complete any of the challenges.

As a team, aUToronto benefited from being well-organized and performing frequent testing on Zeus. Frequent testing was enabled by having access to private roads at U of T's Institute for Aerospace Studies (UTIAS) where Zeus' garage is located. The timeline leading up to the competition was broken up into milestones which each culminated in a series of real-world tests. Two of these milestones tests were held at new locations: first at U of T's Mississauga campus and second at the Clearpath Robotics office in Waterloo. By testing Zeus in new locations, we were forced to encounter the shortcomings in our system that might otherwise have gone unnoticed. Lastly, for the six weeks leading up to the competition, a small subset of aUToronto was dedicated to testing Zeus and fixing bugs on a daily basis. We believe that this testing prior to competition was one of the major elements that set Zeus apart in terms of reliability.

There were also several design choices that may have given aUToronto an edge. First, the decision was made early on to avoid active lane detection and to simply focus on driving using a semantic map. We credit some of the dynamic challenge points to the performance of our MPC controller. Due to the competition restriction to use CPUs and FPGAs, significant effort was invested into designing perception components that would run in realtime on CPUs. By keeping this restriction in mind, aUToronto was able to develop perception components with low latency. Using multiple cameras was also a critical component to completing several challenges. Our narrow-field-of-view camera boosted visual detection range, and the 45-degree cameras were essential to tracking a pedestrian from one side of the crosswalk to the other. Finally, we believe that training our DNNs on our own custom dataset allowed us to achieve high precision and recall numbers that we might not have otherwise.

How could this system have been made more general? The reader will note that the system described in this work is very optimized for the AutoDrive competition. Here, we note several aspects of our design that make it generalizable to more complex environments. First, the entire software stack is agnostic to the actual test location. We simply create or acquire a semantic map for a new test location and Zeus is able to drive autonomously. However, the semantic map must be accurate. We believe that our traffic light detection system is generalizable to more complex driving. With a more powerful DNN than SqueezeDet, and a larger training set, it should be possible to detect and classify a greater variety of traffic light states with higher

reliability. Further, our method of projecting the expected traffic light positions onto the image should be generalizable to any new intersection. Our object detection and tracking pipeline, aUToTrack, works well for pedestrians but not as well for cars. This does not necessarily mean that it needs to be replaced entirely for Level 4 autonomy, but rather that there should be several object detectors working in parallel to detect cars, static obstacles, and other traffic participants. In addition, our object tracking pipeline can easily be extended to track a large number of objects and types.

Here we summarize some of the major improvements that should be made before taking Zeus onto public roads. In Section 7, we mentioned that our object detection and tracking pipeline is not immediately extensible to vehicles. This shortcoming is primarily due to our method of clustering LIDAR points to obtain object centroids. In order to detect vehicles and retrieve an accurate 3D bounding box, using a modern 3D object detector is required. These types of detectors work directly on LIDAR data or on a fusion or LIDAR and vision. Examples of these types of detectors include PointPillars (Lang et al., 2019) and AVOD (Ku et al., 2018). Additional components that should be developed inclue a dynamic cluster detection node such as the method described in (Yoon et al., 2019) and an occupancy-grid-style static obstacle detection system to support the primary dynamic object detection.

We believe that our planner needs to be revamped while still maintaining a hierarchical structure. Currently, we stitch together the centerlines of road segments to form a path and perform minor adjustments for smoothness. This approach works well for the constrained environments of the AutoDrive competition but is unlikely to work reliably in real-world driving. For example, nudging around static obstacles that protrude onto the road would be very difficult to achieve with our current planner. Two candidate solutions to replace our current approach include a sampling-based approach as described in (Werling et al., 2010) and a lattice-grid-based approach as described in (Pivtoraiko et al., 2009).

How did the competition environment compare to real-world driving conditions? MCity is a small mock-town built for self-driving testing at the University of Michigan. The test site is a great analog for a small town urban environment. We did not get the impression that this site was designed to facilitate self-driving testing, but rather to be as realistic of a driving analog as possible. Nevertheless, the way in which MCity was used in the competition was far from a realistic driving test. Encounters with pedestrians were limited to four crosswalk situations with familiar pedestrian dummies on remote-control platforms. There were no pedestrians outside of the expected crossing locations and no jaywalkers, both of which are common occurrences in real driving. We believe that the traffic light testing performed at MCity was a realistic test, but simply lacked enough variety to be confident that our system would work on public roads. The most glaring omission was the lack of dynamic vehicles and other traffic participants which a self-driving car must be capable of handling on public roads. Furthermore, the tests were conducted in sunny or overcast conditions so inclement weather and night-time driving were not a factor.

How do these lessons learned translate to real-world driving? First, having a clear understanding of the strengths and weaknesses of a chosen DNN architecture is critical. To reiterate, we chose to use SqueezeDet because it was the best approach that fit within our computational constraint of using CPUs and FPGAs. There are many alternatives to this architecture which can run in real-time on a moderately powerful GPU while achieving significantly greater performance. Second, the quantity and quality of the training data can have a large impact on DNN performance. Great care must be taken to collect a sufficient quantity of data and for it to be representative of the test distribution while being mindful of class imbalances and the long tail of infrequent objects. Creating a dataset, creating a training pipeline, and managing a hyperparameter search is a significant engineering effort. Most of the errors Zeus encountered at the Year 2 competition were the result of insufficient testing or simple software bugs. Clearly, following software engineering best practices and performing exhaustive simulation testing is critical to minimizing these preventable bugs.

Figure 29: This figure depicts Zeus at the start of the Intersection Challenge, in front of the movable building facades at MCity.

15   Conclusions and Future Work

This article provided a system description of Zeus, aUToronto's winning entry in the SAE AutoDrive Challenge. A final picture of Zeus at the start of the Intersection Challenge is shown in Figure 29. We described the team's organizational structure and the development timeline. In the system overview, we described the layout of sensors on Zeus and the resulting fields of view. We also described Zeus' software architecture at a high level. In the subsequent sections, we described the design of each major component of Zeus' software stack. In the final section, we described Zeus' performance on each dynamic challenge.

In order to achieve high performance from our DNNs for object detection, we collected our own dataset using a mixture of private and public road driving in Toronto. Our object detection and tracking pipeline, dubbed aUToTrack, relied on a 2D DNN and LIDAR clustering to identify pedestrians. We employed a Model Predictive Controller which enabled high tracking performance at the competition. For the majority of each challenge, we stayed within 15 cm of the desired centerline.

Year 3 of the AutoDrive Challenge will be held at the Ohio Transportation Research Center in October 2020. The purpose of the challenge is to simulate an autonomous ride-sharing vehicle. The challenge will include dynamic pedestrians, traffic lights and traffic signs. The most notable additions are the introduction of road closures in the form of construction zones. The routes are intended to be much longer and varied.

Our future work will include adding a prediction component to object detection and tracking. In order to properly handle jaywalkers and deer, it will be critical to predict the future positions of traffic participants. Further, we are developing a LIDAR-based 3D object detector to detect vehicles in future years of the competition. We will research using the output of the perception nodes as a localization correction. For the purpose of safety, we will investigate adding sensors to cover the blind spots of the HDL-64. Our planning software will be updated to dynamically plan paths around obstacles. We will investigate using a dynamic vehicle model with our model predictive controller. We will implement a dynamic cluster detector and an occupancy grid generator. We will be reopening our DNN architecture search in an attempt to find a better architecture that still runs in realtime.

16   Acknowledgements

References

Aeberhard, M., Rauch, S., Bahram, M., Tanzmeister, G., Thomas, J., Pilat, Y., Homm, F., Huber, W., and Kaempchen, N. (2015). Experience, results and lessons learned from automated driving on germany's highways. *IEEE Intelligent Transportation Systems Magazine*, 7(1):42–57.

Apollo (2019). Apollo: An open autonomous driving platform. `https://github.com/ApolloAuto/apollo`.

Barfoot, T. D. (2017). *State Estimation for Robotics*. Cambridge University Press.

Behrendt, K. and Novak, L. (2017). A deep learning approach to traffic lights: Detection, tracking, and classification. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE.

Bewley, A., Ge, Z., Ott, L., Ramos, F., and Upcroft, B. (2016). Simple online and realtime tracking. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 3464–3468. IEEE.

Broggi, A., Cerri, P., Debattisti, S., Laghi, M. C., Medici, P., Molinari, D., Panciroli, M., and Prioletti, A. (2015). Proud—public road urban driverless-car test. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):3508–3519.

Burnett, K., Samavi, S., Waslander, S. L., Barfoot, T. D., and Schoellig, A. P. (2019). aUToTrack: A lightweight object detection and tracking system for the SAE autodrive challenge. *Computer and Robot Vision (CRV)*.

Burnett, K., Schimpe, A., Samavi, S., Gridseth, M., Liu, C. W., Li, Q., Kroeze, Z., and Schoellig, A. P. (2018). Building a winning self-driving car in six months. *arXiv preprint arXiv:1811.01273*.

Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. (2019). nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*.

Carvalho, A., Gao, Y., Gray, A., Tseng, H. E., and Borrelli, F. (2013). Predictive control of an autonomous ground vehicle using an iterative linearization approach. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 2335–2340.

Furgale, P., Schwesinger, U., Rufli, M., Derendarz, W., Grimmett, H., Mühlfellner, P., Wonneberger, S., Timpner, J., Rottmann, S., Li, B., Schmidt, B., Nguyen, T. N., Cardarelli, E., Cattani, S., Brüning, S., Horstmann, S., Stellmacher, M., Mielenz, H., Köser, K., Beermann, M., Häne, C., Heng, L., Lee, G. H., Fraundorfer, F., Iser, R., Triebel, R., Posner, I., Newman, P., Wolf, L., Pollefeys, M., Brosig, S., Effertz, J., Pradalier, C., and Siegwart, R. (2013). Toward automated driving in cities using close-to-market sensors: An overview of the v-charge project. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 809–816.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1050–1059. JMLR.org.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Girshick, R. B., Donahue, J., Darrell, T., and Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524.

Himmelsbach, M., Müller, A., Luettel, T., and Wuensche, H.-J. (2008). Lidar-based 3d object perception.

Huang, X., Cheng, X., Geng, Q., Cao, B., Zhou, D., Wang, P., Lin, Y., and Yang, R. (2018). The apolloscape dataset for autonomous driving. *CoRR*, abs/1803.06184.

Jo, K., Kim, J., Kim, D., Jang, C., and Sunwoo, M. (2015). Development of autonomous car—part ii: A case study on the implementation of an autonomous driving system based on distributed architecture. *IEEE Transactions on Industrial Electronics*, 62(8):5119–5132.

Kato, S., Takeuchi, E., Ishiguro, Y., Ninomiya, Y., Takeda, K., and Hamada, T. (2015). An open approach to autonomous vehicles. *IEEE Micro*, 35(6):60–68.

Kato, S., Tokunaga, S., Maruyama, Y., Maeda, S., Hirabayashi, M., Kitsukawa, Y., Monrroy, A., Ando, T., Fujii, Y., and Azumi, T. (2018). Autoware on board: Enabling autonomous vehicles with embedded systems. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 287–296.

Ku, J., Mozifian, M., Lee, J., Harakeh, A., and Waslander, S. L. (2018). Joint 3d proposal generation and object detection from view aggregation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE.

Lang, A. H., Vora, S., Caesar, H., Zhou, L., Yang, J., and Beijbom, O. (2019). Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12697–12705.

Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., Kolter, J. Z., Langer, D., Pink, O., Pratt, V., Sokolsky, M., Stanek, G., Stavens, D., Teichman, A., Werling, M., and Thrun, S. (2011). Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 163–168.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer.

Maddern, W., Pascoe, G., Linegar, C., and Newman, P. (2017). 1 Year, 1000km: The Oxford RobotCar Dataset. *The International Journal of Robotics Research (IJRR)*, 36(1):3–15.

Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A., and Thrun, S. (2008). Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 25(9):569–597.

Moosmann, F., Pink, O., and Stiller, C. (2009). Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion. In *2009 IEEE Intelligent Vehicles Symposium*, pages 215–220. IEEE.

Nanfack, G., Elhassouny, A., and Thami, R. O. H. (2018). Squeeze-segnet: a new fast deep convolutional neural network for semantic segmentation. *Tenth International Conference on Machine Vision (ICMV 2017)*.

Novel, B., Campion, G., and Bastin, G. (1995). Control of nonholonomic wheeled mobile robots by state feedback linearization. *I. J. Robotic Res.*, 14:543–559.

OpenStreetMap (2020). Openstreetmap. `https://wiki.openstreetmap.org/w/index.php?title=API&oldid=1929746`. Accessed: March 2020.

Paden, B., Cap, M., Yong, S. Z., Yershov, D., and Frazzoli, E. (2016). A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55.

Pendleton, S., Uthaicharoenpong, T., Chong, Z. J., Guo Ming James Fu, Qin, B., Wei Liu, Xiaotong Shen, Zhiyong Weng, Kamin, C., Ang, M. A., Kuwae, L. T., Marczuk, K. A., Andersen, H., Mengdan Feng, Butron, G., Chong, Z. Z., Ang, M. H., Frazzoli, E., and Rus, D. (2015). Autonomous golf cars for public trial of mobility-on-demand service. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1164–1171.

Pivtoraiko, M., Knepper, R. A., and Kelly, A. (2009). Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333.

Qi, C. R., Liu, W., Wu, C., Su, H., and Guibas, L. J. (2018). Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 918–927.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.

RightHook (2020). Righthook. `https://righthook.io/`. Accessed: March 2020.

Roborace (2019). Roborace. `https://roborace.com/`.

ROS (2019). ROS. `http://www.ros.org/`. Accessed: October 2019.

Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China.

Tas, Ö. S., Salscheider, N. O., Poggenhans, F., Wirges, S., Bandera, C., Zofka, M. R., Strauss, T., Zöllner, J. M., and Stiller, C. (2018). Making Bertha cooperate–team AnnieWAY's entry to the 2016 grand cooperative driving challenge. *IEEE Transactions on Intelligent Transportation Systems*, 19(4):1262–1276.

Unnikrishnan, R. and Hebert, M. (2005). Fast extrinsic calibration of a laser rangefinder to a camera. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-05-09*.

Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M. N., Dolan, J., Duggins, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T. M., Kolski, S., Kelly, A., Likhachev, M., McNaughton, M., Miller, N., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Salesky, B., Seo, Y.-W., Singh, S., Snider, J., Stentz, A., Whittaker, W. R., Wolkowicki, Z., Ziglar, J., Bae, H., Brown, T., Demitrish, D., Litkouhi, B., Nickolaou, J., Sadekar, V., Zhang, W., Struble, J., Taylor, M., Darms, M., and Ferguson, D. (2008). Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466.

Valls, M. d. l. I., Hendrikx, H. F. C., Reijgwart, V., Meier, F. V., Sa, I., Dubé, R., Gawel, A. R., Bürki, M., and Siegwart, R. (2018). Design of an autonomous racecar: Perception, state estimation and system integration.

Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*.

Wei, J., Snider, J. M., Kim, J., Dolan, J. M., Rajkumar, R., and Litkouhi, B. (2013). Towards a viable autonomous driving research platform. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 763–770.

Werling, M., Ziegler, J., Kammel, S., and Thrun, S. (2010). Optimal trajectory generation for dynamic street scenarios in a frenet frame. In *2010 IEEE International Conference on Robotics and Automation*, pages 987–993. IEEE.

Wu, B., Iandola, F., Jin, P. H., and Keutzer, K. (2017). Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 129–137.

Yang, B., Luo, W., and Urtasun, R. (2018). Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7652–7660.

Yoon, D., Tang, T., and Barfoot, T. (2019). Mapless online detection of dynamic objects in 3d lidar. In *2019 16th Conference on Computer and Robot Vision (CRV)*, pages 113–120. IEEE.

Yu, F., Xian, W., Chen, Y., Liu, F., Liao, M., Madhavan, V., and Darrell, T. (2018). BDD100K: A diverse driving video database with scalable annotation tooling. *CoRR*, abs/1805.04687.

Ziegler, J., Bender, P., Schreiber, M., Lategahn, H., Strauss, T., Stiller, C., Dang, T., Franke, U., Appenrodt, N., Keller, C. G., Kaus, E., Herrtwich, R. G., Rabe, C., Pfeiffer, D., Lindner, F., Stein, F., Erbs, F., Enzweiler, M., Knoppel, C., Hipp, J., Haueis, M., Trepte, M., Brenk, C., Tamke, A., Ghanaat, M., Braun, M., Joos, A., Fritz, H., Mock, H., Hein, M., and Zeeb, E. (2014). Making Bertha drive - an autonomous journey on a historic route. *IEEE Intelligent Transportation Systems Magazine*, 6(2):8–20.