

Building a Winning Self-Driving Car in Six Months

Keenan Burnett, Andreas Schimpe, Sepehr Samavi, Mona Gridseth, Chengzhi Winston Liu, Qiyang Li,
 Zachary Kroeze, and Angela P. Schoellig

Abstract—The SAE AutoDrive Challenge is a three-year competition to develop a Level 4 autonomous vehicle by 2020. The first set of challenges were held in April of 2018 in Yuma, Arizona. Our team (aUToronto/Zeus) placed first. In this paper, we describe Zeus’ complete system architecture and specialized algorithms that enabled us to win. We show that it is possible to develop a vehicle with basic autonomy features in just six months relying on simple, robust algorithms. We do not make use of a prior map. Instead, we have developed a multi-sensor visual localization solution. All the algorithms in the paper run in real-time using CPUs only. We also highlight the closed-loop performance of the system in detail in several experiments.

I. INTRODUCTION

The SAE AutoDrive Challenge [1] is a three-year competition to develop a Level 4 autonomous vehicle by 2020. The first year consisted of three individual challenges: lane-keeping through a series of tight turns, stopping at stop signs, and avoiding static objects with lane change maneuvers. These challenges were held in April of 2018 in Yuma, Arizona. Our team (aUToronto/Zeus) placed first.

In this work, we describe Zeus’ system architecture and the specialized algorithms employed. We show that it is possible to develop a vehicle with basic autonomy features in just six months relying on simple, robust algorithms. Such an approach to address the challenges outlined above has not been shown in literature before. This work may serve as a useful reference for researchers looking to rapidly develop their own platform to tackle open problems in self-driving.

The majority of self-driving systems require high precision maps to navigate safely. However, mapping of the competition track prior to a scored run was prohibited and the courses did not feature multiple laps. Without a map, lane detection becomes critical for safe navigation. As a result, we developed a robust multi-sensor visual localization solution and demonstrate its closed-loop performance in significantly varying conditions.

The Intel computing platform prescribed by the competition required us to develop algorithms that would run on CPUs only. For visual perception tasks this presents a considerable challenge as most modern systems rely on GPUs to run deep neural networks [2], [3]. By adhering to this constraint, this system described here may prove useful to researchers looking to develop low-power autonomy.

Experimental results demonstrating closed-loop performance on self-driving cars is lacking in the literature. Hence, as a further contribution, we demonstrate the closed-loop performance on each aforementioned challenge.

The authors are with the Dynamic Systems Lab (www.dynsyslab.org) at the University of Toronto Institute for Aerospace Studies (UTIAS), Canada.



Fig. 1. Zeus at the Year 1 AutoDrive Challenge in Yuma, Arizona. Video at: <http://tiny.cc/zeus-y1>.

II. RELATED WORK

The DARPA Grand Challenges were the first to demonstrate the feasibility of autonomous urban driving [4], [5]. At a decade old, these systems are now outdated. Advances in parallel computing and computer vision are reflected in more recent systems, see [6], [7].

In 2014, the Mercedes/Bertha team navigated 100 km autonomously using only radar and vision [8]. An update to this system for cooperative driving was presented in [9]. Recent effort has also been focused on open-source self-driving. Most notable are Autoware [2], [10] and Apollo [3]. These repositories demonstrate modern self-driving systems but rely heavily on the use of a GPU. In contrast, Zeus’ software runs in real-time using CPUs only.

Popular self-driving datasets [11]–[13] are a useful reference of sensor suites but do not provide a complete self-driving architecture as is done in this work.

Other related works include the design of an autonomous racecar [14] and autonomously navigating rural environments [15]. In comparison to [14], we were prohibited from mapping the competition course and did not get multiple laps. Instead, we rely on a custom visual localization solution. While [15] estimates road boundaries for autonomous navigation, the competition tracks did not allow this solution. And although [15] does not require a detailed prior map, it still requires a topological map with GPS waypoints. This is not a requirement here. None of the above works present their closed-loop driving performance as is done here.

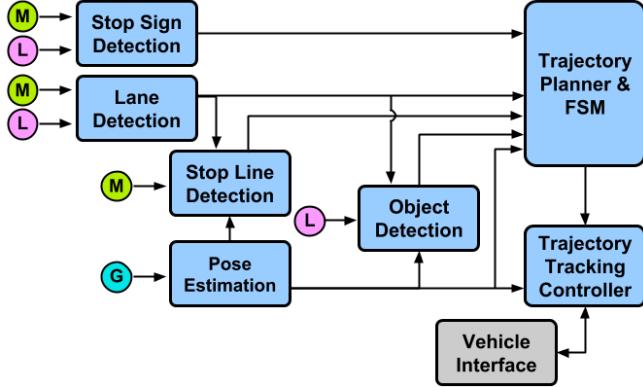


Fig. 2. *Zeus'* software architecture. M: Monocular camera, L: 3D LiDAR, G: GPS/IMU, S: Stereo camera. The Finite State Machine (FSM) determines the current state (lane-keeping | stopping | lane changing).

III. SYSTEM ARCHITECTURE

Zeus is a 2017 Chevrolet Bolt electric vehicle. The sensors consist of a Velodyne HDL-64 3D LiDAR, a PointGrey Grasshopper monocular camera, a PointGrey Bumblebee stereo camera, and a NovAtel PwrPak7 GPS/IMU. Figure 1 illustrates the placement of these sensors on *Zeus*. *Zeus'* computing platform features two Intel Xeon E5-2699R v4 CPUs. This equates to 44 physical cores running at 2.20 GHz. All software was written in C++ using the Robot Operating System (ROS) [16]. Communication between the vehicle and compute server is facilitated by a custom CAN interface developed by aUToronto. This interface allows us to control steering, braking, propulsion, and transmission. *Zeus'* software architecture is shown in Figure 2.

IV. LANE DETECTION

Without a map, detecting lane lines becomes critical for safe navigation. Furthermore, the development of a reliable lane detection becomes even more challenging when computational constraints, low-latency requirements, and environmental variations (Yuma desert vs. Toronto winter) are taken into account. A diagram of the lane detection pipeline is shown in Figure 3. First, monocular camera images are converted into a Bird's Eye View (BEV) perspective. Second, we obtain lane marking pixels using three independent methods, two vision-based and one LiDAR-based. A centerline is fit to each. This results in three independent measurements of the centerline which are combined in a single Kalman filter. The centerline of the ego-lane is modeled as a quadratic.

A. Steerable Filters

Steerable filters [17] are a traditional computer vision approach, slightly more complex than simple edge detection. They can be made to change in orientation along with the lanes in the image, which presents a significant advantage if the road curvature is high. This is in contrast with approaches that use static filters for the entire image [18]. Steerable Filters can also be tuned to detect only bright lines such as lanes as opposed to all possible edges. The output steerable filters can be seen in Figure 4.

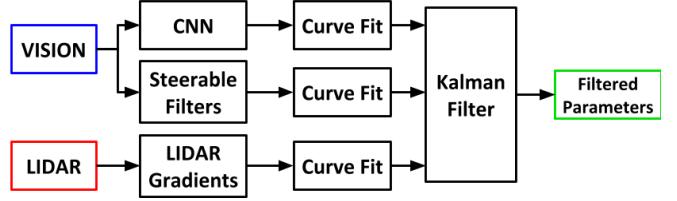


Fig. 3. Lane detection architecture.

B. Convolutional Neural Network

A CNN was developed for the purpose of lane marking segmentation. The network uses an encoder-decoder structure in which the encoder consists of convolution and max pooling layers and the decoder consists of deconvolution and upsampling layers. Skip connections from the encoder layers to the decoder layers allow the network to perform high-resolution upsampling as is done in [19]. This architecture is depicted in Figure 5. Given the recent success of deep learning in semantic segmentation, a deep neural network is a clear choice for this problem. However, the competition's requirement of using CPUs only represented a significant hurdle to this approach. Thus, we opted to design a custom CNN with significantly less layers and filters (see Figure 5) compared to the state-of-the-art. We trained the network in Tensorflow [20] on a custom hand-labeled dataset. Random rotation, flipping, contrast, and brightness was applied to augment the custom dataset. The resulting network is capable of processing frames in real-time (50 Hz) on CPUs.

C. LiDAR-Based Lane Detection

Our LiDAR-based lane detection is based on the approach in [21]. This method takes advantage of the difference in infrared reflectivity between lane marking paint and road asphalt. Each laser scan is searched for regions with high-intensity gradients. Unfortunately, the points resulting from a single scan are too sparse to extract smooth and consistent

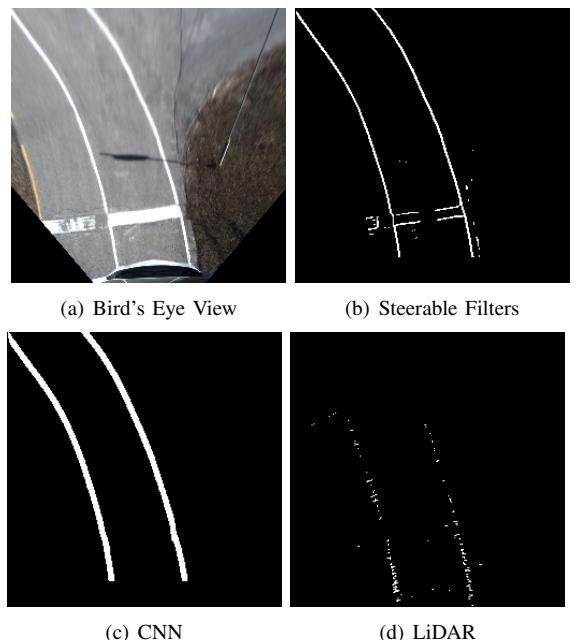


Fig. 4. Comparison between lane marking pixel extraction methods.

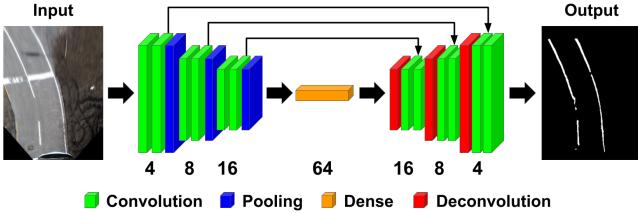


Fig. 5. Convolutional Neural Network architecture. Numbers correspond to the filters. The first two convolutional layers have 4 filters each.

lane parameters. Thus, we accumulate the high-intensity points from several consecutive laser scans. The scans are registered to each other using Iterative Closest Point [22]. Using a minimum of two prior scans is sufficient to alleviate the point sparsity problem. As a pre-processing step, we reject points outside a ground plane prior: 30 m longitudinal, 12 m lateral and 1 m vertical. This step is critical to achieve real-time performance. We are able to run this approach at 26 Hz. An example output is shown in Figure 4.

D. Quadratic Curve Fitting

The curve fitting procedure first runs RANSAC due to its capability of rejecting outliers. Linear Least Squares is then run on the inliers for a more accurate model, yielding the instantaneous parameters for quadratic curves.

E. Tracking

This instantaneous curve fitting is not reliable enough for closed-loop driving. Due to visual distractors and noise, misdetections are not uncommon. In order to smoothly and robustly track estimated lane parameters over time, we employ a linear Kalman filter to the lane parameters. The motion model for the lane parameters takes the following form:

$$x_{k+1} = x_k + \xi_k, \quad (1)$$

where $x_k = [a, b, c]^T$ is the vector of parameters corresponding to the quadratic centerline, and $\xi_k \sim \mathcal{N}(0, R)$ is zero-mean Gaussian system noise. We assume that lane parameters stay approximately the same from one frame to the next within some variation captured by ξ_k . We argue that this assumption is reasonable given the relatively low speeds *Zeus* operates at ($\leq 30 \text{ km/h}$), and the high frame rate of our pipeline. We determined R through experimentation in closed-loop driving tests. Our measurement model for each measurement is represented as:

$$y_k = x_k + \delta_k, \quad (2)$$

where y_k represents the measurements from a given detection method, $\delta_k \sim \mathcal{N}(0, \Omega)$ is the measurement noise associated with that method, and

$$\Omega = \begin{bmatrix} \sigma_a^2 & 0 & 0 \\ 0 & \sigma_b^2 & 0 \\ 0 & 0 & \sigma_c^2 \end{bmatrix}. \quad (3)$$

Of course, each method has a different measurement noise. Since each detection method runs at a different frequency, we apply the measurement updates asynchronously.

We first determined Ω for steerable filter measurements. Other measurement covariances were then chosen to be ratios of Ω . We also tune the relative magnitude of system and measurement covariances, which allows a trade-off between responsiveness to measurements or smooth state estimates. Tuning this parameter was important for performance on tight corners. The entire pipeline runs in real-time (26 Hz).

F. Detection of Adjacent Lanes

For obstacle avoidance, the vehicle needed to have knowledge about the existence of adjacent lanes. To this end, we use the ego-lane width to create two regions of interest (ROIs) where we expect adjacent lane lines to be. We fit curves to the points segmented in the ROIs. We construct a Gaussian with the parameters of the ego-lane as the mean. We leave the covariance as a tuning parameter. We use a two-tailed test to calculate a similarity score between the adjacent lane lines and the ego-lane lines. We use a similarity threshold for determining the existence of adjacent lanes.

V. STOP SIGN DETECTION

Our Stop Sign Detector consists of a Haar Cascade classifier [23]. Compared to an alternative deep learning object detector such as YOLO [24], Haar Cascades run very fast on CPUs. They are also very simple to train and tune. We trained our classifier using OpenCV's Haar Cascade trainer on a custom hand-labeled dataset [25]. We obtained the best results when images were first pre-processed using adaptive histogram equalization. We achieve 97% precision and 90% recall on our own challenging dataset. Our classifier runs at 26 Hz and achieves a detection range in excess of 30 m.

Once a Stop Sign is localized in a camera frame, we determine its relative depth using LiDAR. Points are first transformed into the camera frame and projected onto the image plane. We retain points that lie within the stop sign bounding box, fit a plane, and then output the distance to the plane. In order to fuse LiDAR and camera data in this manner, an accurate extrinsic calibration is required. We used an existing library for this purpose [26].

VI. TRAJECTORY PLANNING

The trajectory planner is comprised of a centerline planner and a lane change planner. A constant velocity profile is used if the ego-lane is unobstructed, otherwise, we generate a linearly decelerating velocity profile. The latter behaviour is also used to stop at stop lines. To change lanes, the lane change planner creates a path starting at the center of the ego-lane and ending at the center of an adjacent lane. This path is modeled as a quintic spline $f(s) = \sum_{i=0}^5 m_i s^i$. To adhere to constraints on maximum lateral acceleration of the vehicle, we desire to minimize the curvature of the planned path. For this, we approximate the curvature with the concavity $\kappa(s) \approx f''(s)$ and solve the following optimization problem for the vector of coefficients $m = [m_0, m_1, \dots, m_5]^T$:

$$\min_m \int_{s_0}^{s_F} (f''(s))^2 ds \quad (4a)$$

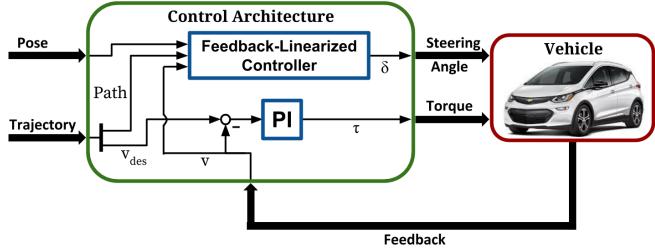


Fig. 6. Control Architecture

subject to

$$f(s_0) = y_0 \quad (4b)$$

$$f'(s_0) = \dot{y}_0 \quad (4c)$$

$$f''(s_0) = \ddot{y}_0 \quad (4d)$$

where (s_0, y_0) and (s_F, y_F) are the starting position and desired final position respectively, and \dot{y}_0, \ddot{y}_0 are the starting velocity and starting acceleration respectively.

VII. MOTION CONTROL

Motion control is carried out using a decoupled control architecture as shown in Figure 6. The longitudinal controller computes a torque that is commanded to the vehicle to track the desired velocity. This is done using a Proportional-Integral (PI) controller for which the gains are estimated through open loop tests and fine-tuned during experiments. The lateral controller is responsible for determining the steering angle to track the desired path. For this, a feedback-linearized controller (FBL) [27] is used.

Deviation with respect to a tracking point on the desired path is quantified by lateral error e_k^L and heading error e_k^H . Using the kinematic bicycle model in the rear-axle frame, their dynamics, discretized with Forward Euler, are given by

$$e_{k+1} = e_k + t_s \dot{e}_k = \begin{bmatrix} e_k^L \\ e_k^H \end{bmatrix} + t_s \begin{bmatrix} v \sin(e_k^H) \\ \frac{v}{L} \tan(\delta_k) \end{bmatrix}, \quad (5)$$

with the steering angle δ_k , the vehicle wheelbase L and the constant forward velocity v . Defining the new system states $p_k = [e_k^L \ v \sin(e_k^H)]^T$, an equivalent linear system can be derived as

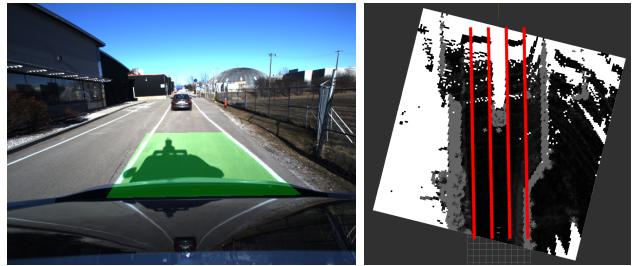
$$p_{k+1} = p_k + t_s \dot{p}_k = \begin{bmatrix} 1 & t_s \\ 0 & 1 \end{bmatrix} p_k + t_s \begin{bmatrix} 0 \\ 1 \end{bmatrix} \eta_k, \quad (6)$$

with the new control input $\eta_k = \frac{v^2}{L} \cos(e_k^H) \tan(\delta_k)$. Choosing a proportional controller $\eta_k = -\gamma^T p_k$ with the gains $\gamma = [\gamma_1 \ \gamma_2]^T$, the stable closed-loop system, for $\gamma_1, \gamma_2 > 0$, is given by

$$p_{k+1} = \begin{bmatrix} 1 & t_s \\ -t_s \gamma_1 & 1 - t_s \gamma_2 \end{bmatrix} p_k. \quad (7)$$

Finally, with the two definitions of the new control input, the feedback-linearized control law for steering is derived as

$$\delta_k = \arctan \left(\frac{-\gamma_1 e_k^L - \gamma_2 v \sin(e_k^H)}{v^2 \cos(e_k^H)} \right). \quad (8)$$



(a) Lane detection

(b) Occupancy grid

Fig. 7. Obstacle detection projects detected lane lines into an occupancy grid and looks for clusters of occupied cells (light grey).

This control law contains the two controller gains γ_1 and γ_2 . Here, the corrections for lateral and heading errors can be prioritized. A third parameter is the look-ahead distance along the desired path at which the tracking point is selected. This was found to have an effect on the smoothness of the steering inputs. As the Bird's Eye View image starts several meters in front of the vehicle, the centerline estimate, interpolated back to the rear-axle, was less reliable and stable. Larger look-aheads oppress some of the noise, although this results in the vehicle potentially cutting corners.

A Model Predictive Control (MPC) approach was also developed as an alternative. The advantage of MPC is that hard constraints on lateral and longitudinal acceleration can be imposed as per competition requirements. Under the time constraints, we were unable to adequately tune the MPC to achieve better performance than the FBL controller. This is likely due to the desired trajectory being updated at a rate of 26 Hz. For a static sequence of GPS waypoints, MPC would have likely been a superior choice. A comparison of MPC with FBL can be found in Section IX.

VIII. STATIC OBSTACLE DETECTION

Figure 7 depicts two components of obstacle detection. Figure 7 (a) shows the currently detected lanes in the perspective view. Figure 7 (b) shows the top-down occupancy grid with the extrapolated lane positions projected onto it in red. Note that white corresponds to no data, black to free space, and grey to obstacles. In the "middle" lane, it can be seen that a grey cluster is centered in the lane.

The Elevation Mapping [28] library is used to create a 2.5D gravity-aligned elevation map around the vehicle. The map covers a $35 \text{ m} \times 35 \text{ m}$ area in front of the car at a resolution of 0.25 m. A naive approach would simply threshold on elevation to detect obstacles. However, we observed that elevation alone is unreliable due to its sensitivity to pose drift (our GPS/IMU's error was observed to be $< 30\text{cm}$ without TerraStar corrections). Instead, we calculate a traversability score [29] which is a function of map slope and roughness. We then create an occupancy grid using traversability and apply smoothing to reduce the occurrence spurious detections. To run occupancy mapping in real-time, we downsample incoming point clouds, and limit the horizontal field of view to 120° . Map size and resolution were chosen to minimize computation without impacting the ability to detect small objects.

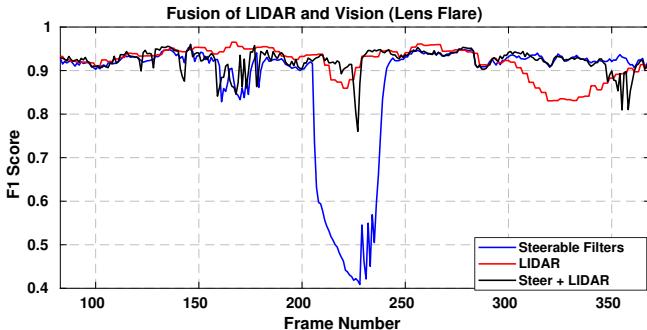


Fig. 8. Fusion of Steerable Filters and LiDAR (Lens Flare Scenario).

To detect objects, we project detected lanes into the occupancy grid using an extrinsic transformation between the monocular camera and LiDAR acquired using the calibration process described in [26]. We then iterate through the grid and check for obstacles within the lanes. To further minimize the occurrence of spurious detections, we threshold on traversability and look for clusters of occupied cells. A visualization of this process is shown in Figure 7 (b).

IX. EXPERIMENTS

In this section, we describe several experiments that demonstrate the closed-loop performance of our system. All experiments were performed on private roads around UTIAS.

For benchmarking lane detection, images are first converted to a BEV image and the lane markings are hand-labeled. We then fit a polygon to delimit the area within the lane markings. We compare the polygons based on detections and ground truth to calculate an F1 score for per-pixel classification on each frame [30]. We calculate RMS error based on the lateral difference between hand-labeled and detected centerline. The results reported in Table I.

Sunny: All approaches have comparable performance with vision slightly outperforming LiDAR. **Shadows:** Vision-based approaches perform worse due to shadows. The LiDAR-based approach performed worse due to worse road quality on this course. **Overcast:** Vision-based algorithms improve in overcast conditions due to diffuse lighting. The LiDAR-based approach dropped in performance possibly due to a wet road. **Lens Flare:** Vision-based algorithms suffer in the event of a lens flare, but the LiDAR-based approach performed similarly to Sunny. **Tight Corners:** High-curvature sections are difficult for all approaches.

These results demonstrate that the fusion of multiple approaches yields comparable, if not better performance as shown in Figure 8. At the competition, we opted to use Steerable filters due to its simplicity and limited development time. Qualitative performance is shown in Figure 9.

A. Closed-Loop Lane Keeping

To benchmark closed-loop lane-keeping performance, we compare against ground truth generated from a human driver. To increase accuracy, we use the average of several traverses. GPS waypoints are collected at 5 cm intervals using a Novatel PwrPak7-E1 GNSS Intertial Navigation System (INS). Using TerraStar corrections yields an accuracy of 6 cm. It

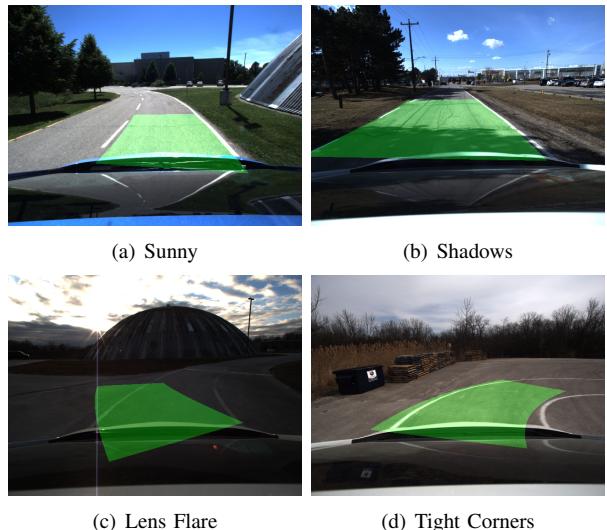


Fig. 9. Qualitative performance of Steerable Filters for lane detection.

is important to note that this accuracy applies for both the ground truth collection as well as the data collected for closed-loop runs. The test course is illustrated in Figure 10. The course is roughly 200 m in length, with four tight turns with a 3.0 m radius. The lanes are 3.0 m in width. We tracked a constant velocity of 2.5 m/s for this experiment. Note the lack of curbs and presence of vehicles.

In Figure 11, lateral tracking error for closed-loop runs using both the FBL controller and MPC is plotted as a function of the closest waypoint index. As mentioned in Section VII, it is expected that MPC should have better performance. However, the reference path is constantly being updated by Lane Detection at 26 Hz, and MPC has significantly more parameters to tune compared to FBL. Due to these reasons



Fig. 10. Closed-loop lane-keeping performance around our track at UTIAS.

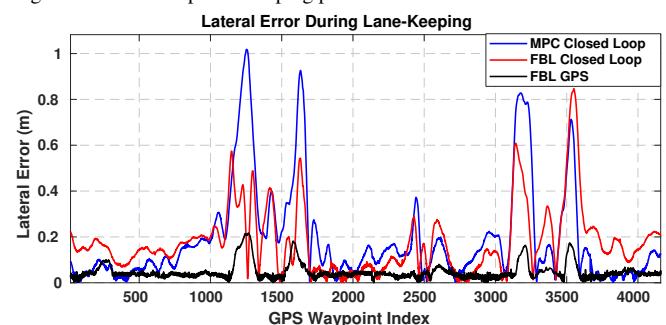


Fig. 11. Lateral tracking error during lane-keeping.

	Sunny		Shadows		Overcast		Lens Flare		Tight Corners	
	RMS (m)	F1								
Steerable Filter	0.1365	0.9471	0.2233	0.9433	0.0978	0.9479	0.2741	0.8849	0.3297	0.8869
LiDAR	0.1672	0.9386	0.4172	0.9053	0.3014	0.8996	0.1649	0.9182	0.2994	0.7786
CNN	0.1377	0.9470	0.2130	0.9487	0.0923	0.9487	0.3551	0.8655	0.3421	0.8791
Steerable Filter + CNN	0.1344	0.9479	0.2055	0.9492	0.0976	0.948	0.1832	0.9101	0.3250	0.8871
Steerable Filter + LiDAR	0.1408	0.9466	0.2316	0.9410	0.0826	0.9504	0.1310	0.9215	0.2914	0.9030

TABLE I
COMPARISON OF DIFFERENT LANE DETECTION APPROACHES.

and the time constraints of the competition, we decided to use FBL for path tracking. Figure 11 demonstrates that we are able to achieve an RMS tracking error less than 20 cm in straight sections and less than 1 m in tight turns. Also note: tracking GPS waypoints results in a significant improvement: an RMS error of less than 10 cm.

B. Stopping at stop signs

For this experiment, we recorded the position of our stop sign and stop line in constant UTM coordinates. In this way, we are able to extrapolate a ground truth distance from the stop sign to our current position by using the INS. In this experiment, we start from rest, accelerate to 2.5 m/s and then decelerate to a stop at the stop sign. Figure 12 demonstrates the performance of our velocity controller as we approach a stop sign. This experiment was repeated 10 times and achieved an average stop distance of 14 cm, maximum stop distance of 29 cm, and a minimum stop distance of 1 cm to the stop line.

C. Obstacle Avoidance

For this experiment, we first recorded the location of the obstacle and obtained the location of the lane centers in UTM coordinates. We use Lane Detection to generate the lane center of the current lane, as well as available adjacent lanes, until a lane change is desired. At this point, we construct a quintic spline between the measured lane centers to obtain ground truth. Figure 13 illustrates the test course as well as the paths that our system is able to generate. Note the smoothness of the lane change trajectory and the high quality lane keeping performance before and after the lane change maneuver. In this experiment, we detect the obstacle 24 m away. We are also able to detect varying sizes of objects, down to a 1-meter tall and 0.75-meter wide pylon.

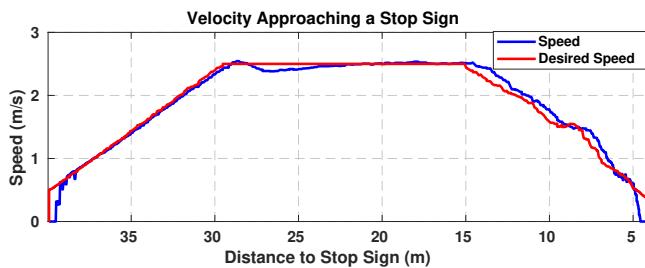


Fig. 12. Velocity profile of *Zeus* approaching a stop sign. While we are able to detect the stop sign at 30 m, we only begin slowing down depending on the desired maximum deceleration.

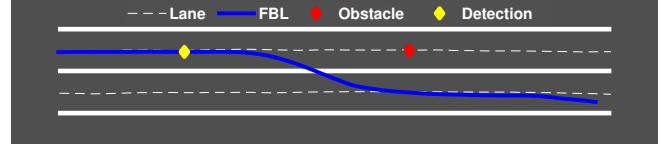


Fig. 13. Closed-loop lane changing in the presence of a static obstruction in the ego-lane. The closed-loop path is shown in blue. We are able to detect obstacles at 30 m and plan an appropriate lane change maneuver that respects the lateral acceleration constraints imposed by the competition.

X. CONCLUSIONS

At our competition, the system performed largely as expected with little tuning required. Parameter tuning included adjusting the gain and bias of steerable filters for the unexpectedly bright Yuma days.

What went well? Lane detection was quite reliable. After extensive testing of tight corners at UTIAS, *Zeus* was well prepared for this challenge. Our system was capable of driving at 3 m/s around tight corners, but we opted to drive at 2.5 m/s in order to maximize our chances of completing the course. Stop sign detection and obstacle detection operated as expected without tuning.

What did not go well? During the Lateral Challenge, faint lane lines and glare off the pavement caused our lane detection to fail during one segment of the lateral challenge. All but two teams failed this segment. During the straight line challenge, dashed lines were very sparse (5-10 m between dashes). Most teams could not handle starting the challenge without any lane lines for 10 m, so the starting point was adjusted by the competition organizers. During the obstacle avoidance challenge, evening glare on the road impacted our multi-lane detection and forced a restart. We also experienced a spurious obstacle detection during our second run which caused the vehicle to come to a complete stop.

By focusing on simple approaches with added redundancy, we were able to build a system sufficiently reliable to win the competition in six months. Only two teams completed all three challenges: aUToronto, and Virginia Tech. Our future work will include incorporating an HD map which will encode the locations of lanes, signs, and traffic lights. We will be looking to augment the current pose estimation system, possibly with a vision- or LiDAR-based localization. The constraint to use CPUs only will be relaxed slightly to allow the use of FPGAs. As such, we will be looking to jump-start aUToronto's development using powerful deep learning tools for all visual perception tasks.

REFERENCES

- [1] “SAE AutoDrive Challenge,” <https://www.sae.org/attend/student-events/autodrive-challenge/>, accessed: September 2018.
- [2] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monroy, T. Ando, Y. Fujii, and T. Azumi, “Autoware on board: Enabling autonomous vehicles with embedded systems,” in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, 2018, pp. 287–296. [Online]. Available: <https://doi.org/10.1109/ICCPs.2018.00035>
- [3] “Apollo: An open autonomous driving platform,” <https://github.com/ApolloAuto/apollo>, 2018.
- [4] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Etinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke *et al.*, “Junior: The stanford entry in the urban challenge,” *Journal of Field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.
- [5] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, “Autonomous driving in urban environments: Boss and the urban challenge,” *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.
- [6] J. Levinson, J. Askeland, J. Becker, J. Dolsion, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun, “Towards fully autonomous driving: Systems and algorithms,” in *2011 IEEE Intelligent Vehicles Symposium (IV)*, 6 2011, pp. 163–168.
- [7] J. Wei, J. M. Snider, J. Kim, J. M. Dolan, R. Rajkumar, and B. Litkouhi, “Towards a viable autonomous driving research platform,” in *2013 IEEE Intelligent Vehicles Symposium (IV)*, 6 2013, pp. 763–770.
- [8] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, E. Kaus, R. G. Herrtwich, C. Rabe, D. Pfeiffer, F. Lindner, F. Stein, F. Erbs, M. Enzweiler, C. Knoppel, J. Hipp, M. Haueis, M. Trepte, C. Brenk, A. Tamke, M. Ghanaat, M. Braun, A. Joos, H. Fritz, H. Mock, M. Hein, and E. Zeeb, “Making Bertha drive - an autonomous journey on a historic route,” *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 2, pp. 8–20, 2014.
- [9] Ö. Ş. Taş, N. O. Salscheider, F. Poggenhans, S. Wirges, C. Bandera, M. R. Zofka, T. Strauss, J. M. Zöllner, and C. Stiller, “Making Bertha cooperate-team AnnieWAY’s entry to the 2016 grand cooperative driving challenge,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 4, pp. 1262–1276, 4 2018.
- [10] “Autoware: Open-source to self-driving,” <https://github.com/CPFL/Autoware>, 2018.
- [11] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the KITTI vision benchmark suite,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [12] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, “1 Year, 1000km: The Oxford RobotCar Dataset,” *The International Journal of Robotics Research (IJRR)*, vol. 36, no. 1, pp. 3–15, 2017.
- [13] X. Huang, X. Cheng, Q. Geng, B. Cao, D. Zhou, P. Wang, Y. Lin, and R. Yang, “The apollo dataset for autonomous driving,” *CoRR*, vol. abs/1803.06184, 2018. [Online]. Available: <http://arxiv.org/abs/1803.06184>
- [14] M. d. I. I. Valls, H. F. C. Hendrikx, V. Reijgwart, F. V. Meier, I. Sa, R. Dubé, A. R. Gawel, M. Bürgi, and R. Siegwart, “Design of an autonomous racecar: Perception, state estimation and system integration,” in *2018 IEEE Conference on Robotics and Automation (ICRA)*, 2018.
- [15] T. Ort, L. Paull, and D. Rus, “Autonomous vehicle navigation in rural environments without detailed prior maps,” in *2018 IEEE Conference on Robotics and Automation (ICRA)*, 2018.
- [16] “ROS,” <http://www.ros.org/>, accessed: September 2018.
- [17] W. T. Freeman and E. H. Adelson, “The design and use of steerable filters,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 9, pp. 891–906, 9 1991.
- [18] Z. Kim, “Robust lane detection and tracking in challenging scenarios,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 1, pp. 16–26, March 2008.
- [19] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2015, pp. 3431–3440.
- [20] “Tensorflow,” <https://www.tensorflow.org/>, accessed: September 2018.
- [21] S. Kammel and B. Pitzer, “Lidar-based lane marker detection and mapping,” in *2008 IEEE Intelligent Vehicles Symposium*, 2008, pp. 1137–1142.
- [22] A. Segal, D. Haehnel, and S. Thrun, “Generalized-icp,” in *Robotics: science and systems*, vol. 2, no. 4, 2009, p. 435.
- [23] P. Viola and M. Jones, “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*, vol. 1, Dec 2001, pp. I–I.
- [24] J. Redmon and A. Farhadi, “YOLO9000: better, faster, stronger,” *CoRR*, vol. abs/1612.08242, 2016. [Online]. Available: <http://arxiv.org/abs/1612.08242>
- [25] “Opencv,” <https://opencv.org/>, accessed: September 2018.
- [26] R. Unnikrishnan and M. Hebert, “Fast extrinsic calibration of a laser rangefinder to a camera,” *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-05-09*, 2005.
- [27] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot, “Visual teach and repeat, repeat, repeat: Iterative Learning Control to improve mobile robot path tracking in challenging outdoor environments,” *IEEE International Conference on Intelligent Robots and Systems*, pp. 176–181, 2013.
- [28] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, “Robot-centric elevation mapping with uncertainty estimates,” in *International Conference on Climbing and Walking Robots (CLAWAR)*, 2014.
- [29] P. Fankhauser and M. Hutter, “A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation,” in *Robot Operating System (ROS) –The Complete Reference (Volume 1)*. Springer, 2016, ch. 5.
- [30] J. Fritsch, T. Kühl, and A. Geiger, “A new performance measure and evaluation benchmark for road detection algorithms,” in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, Oct 2013, pp. 1693–1700.