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 our 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 of our algorithms run in real-time using CPUs only. We also highlight the closed-loop performance of our 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 our system architecture and the specialized algorithms we developed. 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. Our 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, our system 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 of our system 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.

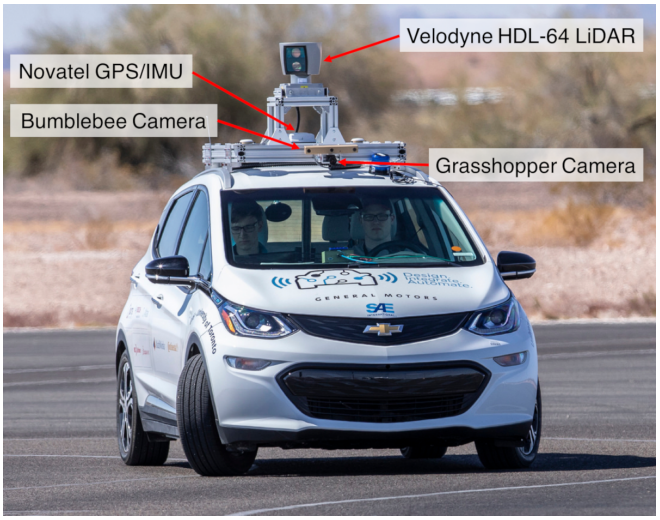


Fig. 1. Our car *Zeus* at the Year 1 AutoDrive Challenge in Yuma, Arizona. Video at: http://tiny.cc/zeus-y1.

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, our 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, our competition track 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 of our system. None of the above works present their closed-loop driving performance as is done here.

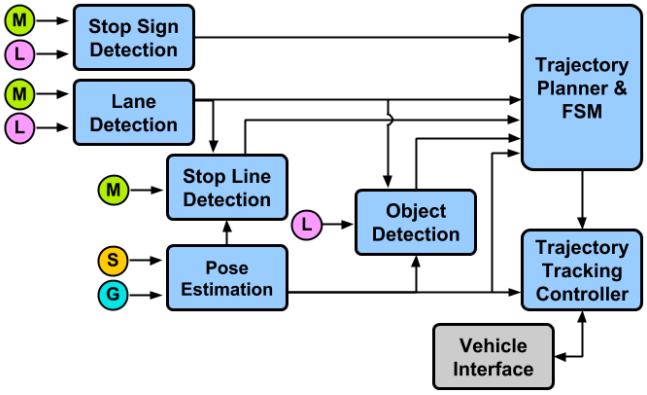


Fig. 2. Our car's software architecture. M: Monocular camera, L: 3D LiDAR, G: GPS/IMU, S: Stereo camera. The Finite State Machine (FSM) determines our current state (lane-keeping | stopping | lane changing).

III. System Architecture

Our car, *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*. Our computing platform features two Intel Xeon E5-2699R v4 CPUs. This equates to 44 physical cores running at 2.20 GHz. Our software was written in C++ using the Robot Operating System (ROS) [16]. Communication between the vehicle and our server is facilitated by a custom CAN interface developed by our team. 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 is critical for safe navigation and forms the basis of our visual localization. We describe this component and its performance in detail below. This component was the most challenging to develop given our computational constraints, low-latency requirements, and unavoidable environmental variations (Yuma desert vs. Toronto winter). A diagram of our lane detection pipeline is shown in Figure 3. First, we convert incoming images into a Bird's Eye View perspective. Second, we filter the images to obtain lane marking pixels using three independent methods, two based on vision and one based on LiDAR. We then fit a centerline to each of the resulting pixel masks. This results in independent measurements of the centerline which are combined in a single Kalman filter. We model the centerline of the vehicle's ego-lane as a quadratic.

A. Steerable Filters

The primary advantage of using Steerable Filters [17] is that the filters can be oriented along with the lanes. This is in contrast to approaches that make use of a set of static filters for the entire image [18]. These oriented filters are better at capturing lane markings at the tight corners exhibited by the Lateral Challenge. Steerable Filters can also be tuned to detect only bright lines such as lanes as opposed to all

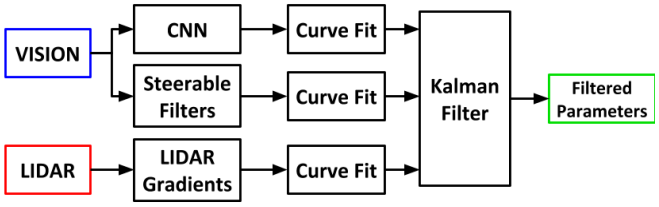


Fig. 3. Lane detection architecture.

possible edges, which is important as distractors such as curbs and cracks are unavoidable in a system based on pure edge detection. The output of our steerable filters can be seen in Figure 5.

B. Convolutional Neural Network

Our team designed a Convolutional Neural Network (CNN) for the purpose of lane marking segmentation. Our 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 4. 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 our own CNN with significantly less layers and filters (see Figure 4) compared to the state-of-the-art. We trained our network in Tensorflow [20] on a custom hand-labeled dataset. Random rotation, flipping, contrast, and brightness was applied to augment our 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. The resulting set of points are then accumulated over several laser scans in order to get a denser result. Scans are accumulated over a sliding window and aligned using Iterative Closest Point [22]. The resulting set of points are then projected into a Bird's Eye View image as seen in Figure 5.

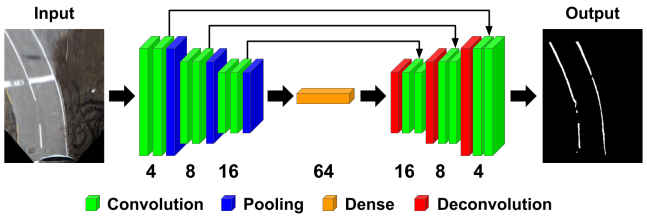
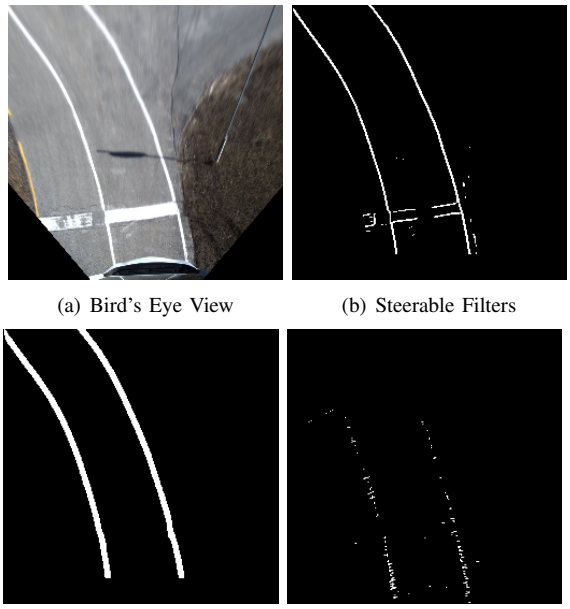


Fig. 4. Convolutional Neural Network architecture. Numbers correspond to the filters. The first two convolutional layers have 4 filters each.



(a) Bird's Eye View (b) Steerable Filters

(c) CNN (d) LiDAR

Fig. 5. Comparison between lane marking pixel extraction methods.

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 our quadratic curve.

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 our 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:

$$x_{k+1} = x_k + \xi_k, \tag{1}$$

where $x_k = [a, b, c]^T$ is the vector of parameters corresponding to our quadratic centerline, and $\xi_k \sim \mathcal{N}(0, R)$ is zero-mean Gaussian system noise. We assume that our 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, 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:

$$y_k = x_k + \delta_k, \tag{2}$$

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

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

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).

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 $m = \begin{bmatrix} m_0, m_1, \ldots, m_5 \end{bmatrix}^T$:

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

subject to

$$f(s_0) = y_0 \qquad f(s_F) = y_F \tag{4b}$$

$$f'(s_0) = \dot{y}_0 \qquad f'(s_F) = 0 \tag{4c}$$

$$f''(s_0) = \ddot{y}_0 \qquad f''(s_F) = 0\,, \tag{4d}$$

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.

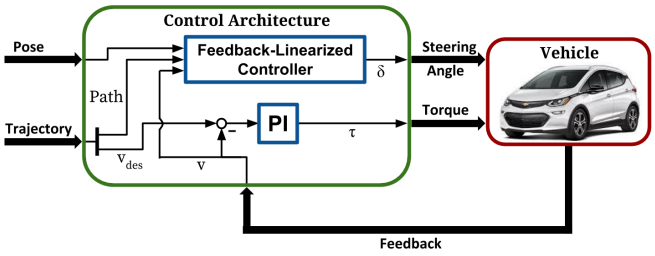
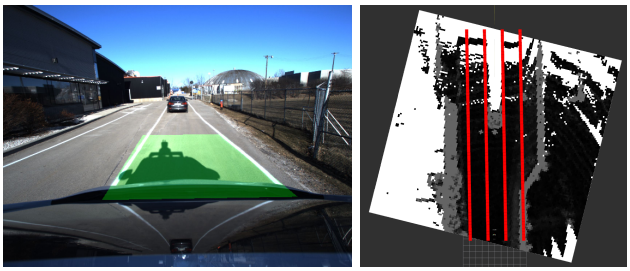


Fig. 6. Control Architecture



(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).

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. 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

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

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

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

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 $\gamma = \begin{bmatrix} \gamma_1 & \gamma_2 \end{bmatrix}^T$, the stable closed-loop system, for $\gamma_1, \gamma_2 > 0$, is given by

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

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

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

This control law contains the two controller gains γ_1 and γ_2. Herewith, 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 of the competition, we were unable to adequately tune the MPC to achieve better performance than the FBL controller. This is likely due to the desired reference trajectory being updated by the lane keeping module at 26 Hz. If the reference trajectory was given by a static sequence of GPS waypoints, which was not allowed by the competition, MPC would have likely been a superior choice. A comparison of MPC with FBL can be found in Section IX.

VIII. Static Obstacle Detection

We use the Grid Map [28] and Elevation Mapping [29] libraries to create a 2.5D gravity-aligned elevation map around the vehicle. The map covers a 35 m $\times$ 35 m area in front of the car at a resolution of 0.25 m.

Elevation alone is unreliable for detecting obstacles due to its sensitivity to pose drift. Instead, we calculate a traversability score [28] which is a function of map slope and roughness. We create an occupancy grid using traversability and apply smoothing to reduce spurious detections.

Given the competition's restriction to use CPUs only, it was challenging to run static occupancy mapping in real-time (10 Hz). To achieve this, we downsample incoming point clouds, and limit the horizontal field of view to 120°. Our map size and resolution was chosen to minimize computation without impacting our ability to detect small objects.

To detect objects, we first project detected lanes into the occupancy grid using an extrinsic transformation between our camera and LiDAR acquired using [26]. We then iterate through parts of the grid that fall between lanes lines to check for obstacles. To avoid spurious detections, we threshold on traversability and look for clusters of occupied cells. A visualization of this pipeline is shown in Figure 7.

IX. Experiments

In this section, we describe several experiments that demonstrate the closed-loop performance of our system on tasks required by the AutoDrive Challenge. All data-collection and experiments were performed on private roads around our institute's facility.

	Sunny		Shadows		Overcast		Lens Flare		Tight Corners	
	RMS (m)	F1	RMS (m)	F1	RMS (m)	F1	RMS (m)	F1	RMS (m)	F1
Steer	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
Steer + CNN	**0.1344**	**0.9479**	**0.2055**	**0.9492**	0.0976	0.948	0.1832	0.9101	0.3250	0.8871
Steer + 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.

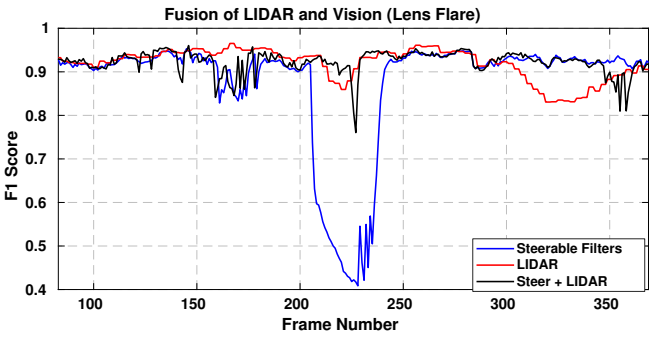
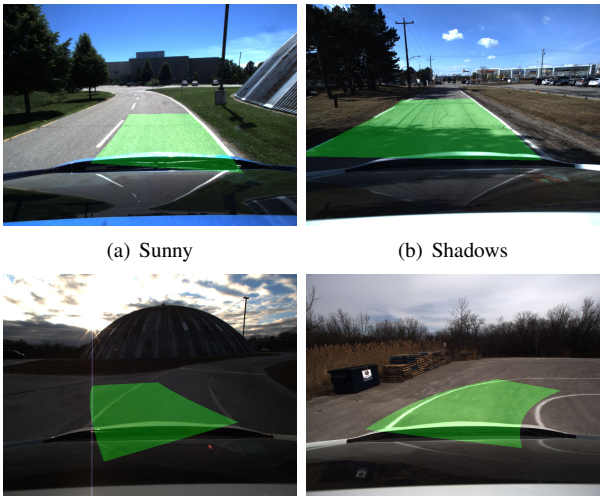


Fig. 8. Fusion of Steerable Filters and LiDAR (Lens Flare Scenario).

Images are first converted to a Bird's Eye View and lane markings are hand-labeled. We then fit a polygon to delimit the ground truth and detected lane markings. We compare these polygons to each other to calculate an F1 score for per-pixel classification on each frame [30]. We also calculate an RMS error based on the lateral difference between our hand-labeled and detected centerline. We play back recorded data and use the output of our filtered lane detection parameters to obtain results. 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 overhanging shadows. 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. Our 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. We have observed that when one method performs poorly for a single frame, the second method will often perform much better. In this way, the combination of multiple methods results in robust performance as shown in Figure 8. In the case of Shadows, a system can benefit from fusing multiple independent vision approaches. In the case of Lens Flares, it is clear that a system can benefit from the fusion of multiple sensing modalities. This benefit is also shown in Figure 8. At the competition, we opted to simply use Steerable filters due to the simplicity of this approach and limited development time. Qualitative performance is shown in Figure 9.



(a) Sunny (b) Shadows

(c) Lens Flare (d) Tight Corners

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

A. *Closed-Loop Lane Keeping*

In order to benchmark our closed-loop lane-keeping performance, we compare against ground truth generated from the average of several traverses by a human driver. GPS waypoints are collected at 5 cm intervals using our Novatel PwrPak7-E1 GNSS Intertial Naviation System (INS). TerraStar corrections sent to the INS yield an RMS position accuracy of 6 cm. It 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 nearby vehicles.

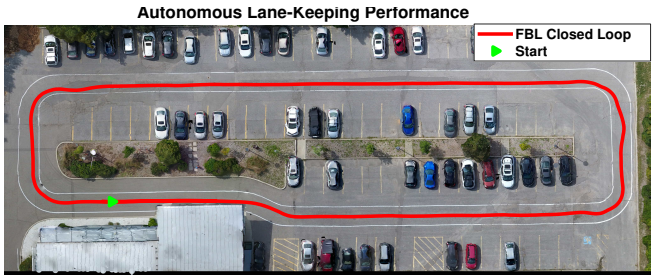


Fig. 10. Closed-loop lane-keeping performance around our track at UTIAS.

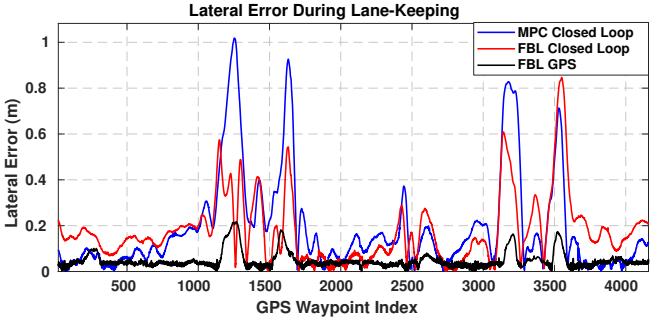


Fig. 11. Lateral tracking error during lane-keeping.

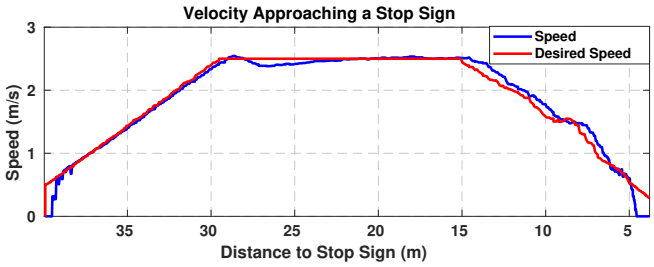


Fig. 12. Velocity profile of *Zeus* approaching a stop sign. While we are able to detect the stop sign at 30 m, we only begin slowing down depending on the desired maximum deceleration.

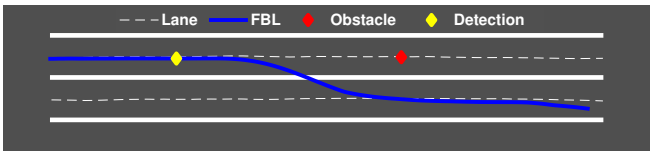


Fig. 13. Closed-loop lane changing in the presence of a static obstruction in the ego-lane. The closed-loop path is shown in blue. We are able to detect obstacles at 30 m and plan an appropriate lane change maneuver that respects the lateral acceleration constraints imposed by the competition.

In Figure 11, lateral tracking error for closed-loop runs using both the FBL controller and MPC is plotted as a function of the GPS waypoint index of the manual run. Of note is that FBL and MPC have similar maximum error, but the RMS of FBL is lower, 23 cm and 29 cm, respectively. As mentioned in Section VII, while it would be expected that MPC would have better performance, 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 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. This figure also shows the significant improvement in lateral error if the ground truth GPS waypoints are used as a reference, achieving an RMS error of less than 10 cm.

Due to the simplicity and the built-in redundancy of our approach, the closed-loop lane-keeping performance is highly reliable. At our competition in the desert in Yuma, Arizona, the system performed as expected with little tuning required despite the drastic difference in environmental conditions compared to winter weather in Toronto, Canada.

B. Stopping at stop signs

For this experiment, we recorded the position of our stop sign and stop line in constant Universal Transverse Mercator (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.

X. Conclusions

In this paper, we described our approach to the Year 1 SAE AutoDrive Challenge. We demonstrated a multi-modal visual localization solution that formed the basis of our autonomy suite. We demonstrated the performance of our system in several closed-loop experiments. By focusing on simple approaches with added redundancy, we were able to build a system sufficiently reliable to win the competition in six months. In fact, very few tuning parameter or code changes were required once at the competition and the system operated largely as expected. Year 2 of the SAE AutoDrive Challenge is to be held at MCity, Michigan in June 2019.

Map-less solutions result in high reliability and are less susceptible to GPS failures. Nevertheless, our future work will include developing a mapping suite to encode the location of lanes, signs, and intersections. In addition, a more comprehensive planning system will be required to handle intersections and traffic lights. We will be looking to augment our pose estimation system, possibly with a vision- or LIDAR-based SLAM solution. Our constraint to use CPUs only will be removed. As such, we will be looking to jump-start our team's development using powerful deep learning tools for all visual perception tasks. We will be looking to accelerate these networks on a combination of CPUs and FPGAs.

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. Monrroy, 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. Ettinger, 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. Dolson, 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 apolloscape dataset for autonomous driving," *CoRR*, vol. abs/1803.06184, 2018. [Online]. Available: http://arxiv.org/abs/1803.06184

[14] M. d. l. I. Valls, H. F. C. Hendrikx, V. Reijgwart, F. V. Meier, I. Sa, R. Dubé, A. R. Gawel, M. Bürki, 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 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.

[29] 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.

[30] J. Fritsch, T. Kühnl, 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.