2018 IEEE Intelligent Vehicles Symposium (IV)
Changshu, Suzhou, China, June 26-30, 2018

A Modular Software Framework for Autonomous Vehicles

Kai Li Lim, Thomas Drage, Roman Podolski, Gabriel Meyer-Lee,
Samuel Evans-Thompson, Jason Yao-Tsu Lin, Geoffrey Channon, Mitchell Poole and Thomas Bräunl

Abstract— **Software frameworks for autonomous vehicles are required to interface and process data from several different sensors on board the vehicle, in addition to performing navigational processes such as path planning and lane keeping. These can include a combination of cameras, LIDARs, GPS, IMU, and odometric sensors to achieve positioning and localisation for the vehicle and can be challenging to integrate. In this paper, we present a unified software framework that combines sensor and navigational processing for autonomous driving. Our framework is modular and scalable whereby the use of protocol buffers enables segregating each sensor and navigation subroutine individual classes, which can then be independently modified or tested. It is redesigned to replace the existing software on our Formula SAE vehicle, which we use for testing autonomous driving. Our testing results verify the suitability of our framework to be used for fully autonomous drives.**

I. INTRODUCTION

UWA's Renewable Energy Vehicle Project (REV) operates two autonomous vehicles, a BMW X5 and a student-built Formula SAE-Electric car. Formula SAE [1] is a long-running annual competition organised by the Society of Automotive Engineers with competition events worldwide. The design competition includes petrol cars, as well as the SAE-Electric class which includes ours with two electric motors driving each of the two rear wheels via independent controllers. We have incorporated full drive-by-wire control of the throttle, steering and the hydraulic braking system. The use of a Formula-SAE car provides several advantages for such a project; the car is mechanically simple, Formula-SAE cars with similar designs are common at universities worldwide and the size of the car makes testing accessible without forgoing race car vehicle dynamics. Furthermore, the use of an electric vehicle makes the project significantly more practical for student work as the risks and environmental issues associated with petrol-engine cars are eliminated and the systems installed in this project can take advantage of the large amount of electrical energy already available on the vehicle.

For the driverless FSAE project, our goal was to be able to autonomously drive a vehicle around a race track. This is being achieved by placing waypoints along the ideal driving line, as well as "fence points" to lock out non-driving areas. Maps can either be recorded by human or remote-controlled driving or specified through a Google Maps driven web-interface. During autonomous driving, a laser scanner and camera are used for detection of road edges as well as any obstacles on the track. Safety systems are essential for a driverless system, as the car weighs more than 250 kg and is capable of driving at a speed of 80 km/h. Both the low-level drive-by-wire, as well as high-level navigation system have independent safety systems built in. These include remote intervention, remote heartbeat and remote emergency stopping, which are implemented through a fail-safe wireless link to a base station as well as through hard-wired buttons on the car itself.

Our motivation for designing and presenting this framework is to improve upon the existing autonomous drive software on our SAE vehicle that is documented in [2]. This software utilises a web-based user interface (UI) that displays via a touch screen mounted in the vehicles cockpit. Our proposed framework utilises this existing UI with a revamped backend as described in Section II. It was noted that the existing software had a heavy reliance on a central Control module, which required all the sensors and their submodules to run to function. These submodules were programmed throughout the years with different programming languages and redundancies, which made integration difficult. Our proposed framework presents a streamlined approach whereby each module will be programmed with a C++ interface that communicates with either a path planner or a drive control system. This system also benefits from additional features including visualisation data playback (online or offline) and a web-based user interface. As a customised framework for our project, this also alleviates the reliance on node-based solutions such as ROS, which usually requires a perpetually running Master node to function and allows higher reliability when the individual components are integrated.

Additionally, this approach presents a long-term advantage whereby our framework is made fully open and contributable by students and enthusiasts looking to implement our framework onto their custom-fabricated vehicles. When compared against other autonomous driving frameworks such as Apollo [3] and Autoware [4] that mostly target commercial vehicles and requiring expensive hardware, our approach leverages on hardware and fabrication methods that are more accessible in cost.

II. AUTONOMOUS DRIVING FRAMEWORK

This section introduces our proposed software framework for the SAE vehicle, with the software architecture diagram as illustrated in Fig. 1. The software architecture of the

[1]The authors are with The REV Project at the School of Electrical, Electronic and Computer Engineering, The University of Western Australia, Perth WA 6009 Australia. {kaili.lim,thomas.braunl}@uwa.edu.au, thomas.drage@research.uwa.edu.au, roman.podolski@tum.de, gmeyerl1@swarthmore.edu, {20490234, 21680206, 21317528, 21212271}@student.uwa.edu.au

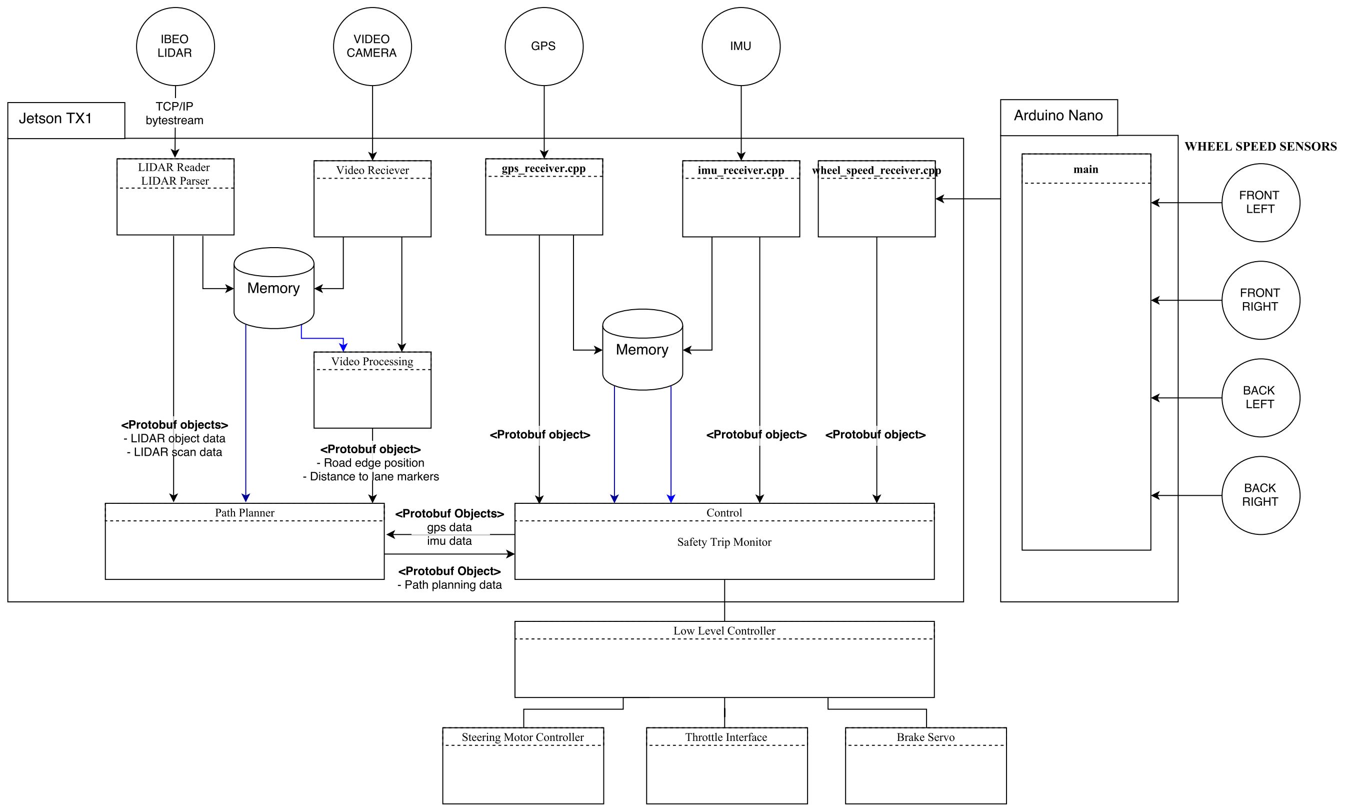

Fig. 1. The software architecture diagram of our proposed framework

vehicle utilises a modular design to allow for continuous parallel development on each of the sensors, the path planning algorithm, and the control policy. Input sensors are comprised of the LIDAR, camera, GPS, IMU and four wheel speed sensors, which are required to function simultaneously in order for the car to drive autonomously. The LIDAR, camera, GPS and IMU each have their own receiver class on the Jetson TX1, which takes the input from the sensors and processes the data. Additionally, rotary-encoder odometry is performed using an Arduino microcontroller connected to each of the wheels' encoder. This information is received in the high-level software by another receiver class which processes the data to produce wheel speed measurements. Relevant data for path planning and localisation is encoded in protobuf [5] format and then passed to either the path-planning program or the control program, which performs localisation as well as some communication and logging utilities. The control program also communicates with the web GUI, providing a visualisation of the data and allows the user to start and stop autonomous driving as well as the safety trip monitor and the controller, thereby introducing a high-level interface for driving the car. The control program, once it has communicated with the path-planner to combine localisation data with a set of future way points, will transmit driving instructions to the controller. This in turn transmits them to a low-level microcontroller. Not pictured in the diagram is the fusion of data from sources such as the GPS and IMU, and IMU and LIDAR in order to improve the accuracy of localisation. Detailed explanations of the sensors and classes are covered in Sections II-A to II-H.

A. Path Planner

We use a real-time capable path-planning algorithm based on [6]. Given a set of pre-recorded or pre-defined waypoints along a track, the planner will generate a optimised path through all way points, which serves as a baseframe for trajectory generation. During operation, the algorithm evaluates a variety of possible trajectories in the configurations space of the vehicle using RRTs [7]. Those intermediate trajectories are generated along a curvilinear coordinate system, along with the baseframe. Each possible trajectory is checked for collisions with obstacles. A collision free path is then picked, utilizing a cost-function, that enables us to influence the driving style of the vehicle. The algorithms are implemented in C++14 and rely heavily on the C++ source libraries Boost and Eigen3.

We use an arc-length parametrised cubic B-spline $P_b(s)$ [8] to generate a baseframe for the curvilinear coordinate system, which can be described as a non-linear transformation of the parameter domain on the four waypoints a to d, parametrised by s.

$$P_b(s) = \begin{cases} x_b(s) & = a_{x,i}(s-s_i)^3 + b_{x,i}(s-s_i)^2 \\ & +c_{x,i}(s-s_i) + d_{x,i} \\ y_b(s) & = a_{y,i}(s-s_i)^3 + b_{y,i}(s-s_i)^2 \\ & +c_{y,i}(s-s_i) + d_{y,i} \end{cases} \tag{1}$$

The curvature κ_b can be calculated from the first and second derivatives of $P_b(s)$

$$\frac{dx_b}{ds} = x_b' = \cos\theta_b \qquad \frac{dy_b}{ds} = y_b' = \sin\theta_b \tag{2}$$

$$\kappa_b = \frac{x_b' y_b'' - x_b'' y_b'}{(x_b'^2 - y_b'^2)^{\frac{3}{2}}} \tag{3}$$

The curvature of a cubic spine is continuous in all sections. For this reason, a parametric cubic B-spline is adopted for the baseframe. Since the position of the vehicle is provided in Cartesian-space, we need to find a transform those coordinates to a curvilinear representation, of which the B-spline provides the base. This is equal to finding the closest point to the vehicle on the baseframe, by minimising the Euclidean distance between the position of the vehicle and the cubic B-spline.

$$\min_s d(s) = \sqrt{(x_v - x_b(s))^2 + (y_v - y_b(s))^2} \tag{4}$$

We choose Brent's method [9] find the minimum. Another suitable and stable algorithm is provided in [10]. From the lateral distance to the baseframe and the baseframe we construct the curvilinear coordinate system in which we generate a number n of paths as cubic polynomials. Each polynomial is defined by a lateral offset $q(s)$ and a curvature κ, to cover a broad section of the configuration space of the vehicle. n is a design parameter and can be chosen to adjust the computational load of the algorithms. We now design a vehicle model of a set of differential equations in Cartesian space.

$$\dot{x} = v\cos\theta, \dot{y} = v\sin\theta, \dot{\theta} = v\kappa \tag{5}$$

This vehicle model is transformed onto the curvilinear coordinate system, and the position of the vehicle in Cartesian space can then be determined by forward integration. Paths that violate the non-holonomic constraints of vehicle motions or collide with an obstacle are eliminated. The remaining paths are evaluated by a construction. The cost function itself can be chosen to optimise driving for an arbitrary property, like sportiness or safety. By simulating the path planner using equally weighted costs, the near-optimal path can be obtained as Fig. 2(a), with the path drawn in green and the baseframe in blue. The manoeuvre selection is subsequently presented as Fig. 2(b) Because this planner insoles a simple vehicle model as a set of ordinary differential equations, it can be conveniently integrated into any control algorithm.

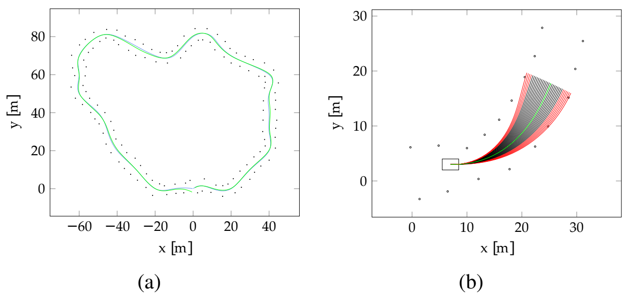

Fig. 2. Simulated near-optimal path (a) and manoeuvre selection (b). Black dots represent road delimiters.

B. Software Communications

The sensors on the vehicle communicate with the path planner using procotol buffers. Protocol Buffers (protobuf) [5] are a formalised data structure developed by Google for use in cross-platform systems. Protobuf allows for the combination of several standard variables into a single packed structure that can be easily serialised and accessed using automatically generated methods. The protobuf library has bindings for many common programming languages, including C++, Python and Java, meaning that a protobuf structure packaged in a C++ application, can be read in by a Java application with no conversion needed.

Protocol buffers are used internally within our software as regularly structured state variables, with easy to use member functions. For example, the GPS software stores its current state within a protobuf object containing data such as latitude, longitude, groundspeed and angle. This GPS state can then be used by internal code regularly, or it can be used serialised and stored. The serialised protobuf data is very compact and space efficient, meaning that a huge amount of logged data can be saved sequentially.

By abstracting the actual data from a specific sensor behind a protobuf object, it allows for the use of Polymorphism within our software, and so dependencies on a specific piece of hardware are loosened. As long as a specific hardware device can be interpreted into the appropriate protobuf form, it can be integrated easily into the system, with only short wrapping code needed to be written. There is also no dependency for this protobuf data to come from a physical sensor, the protobuf data can be read in from a previously serialised log, allowing for the replay of data, or it can be read from an external piece of software, allowing for the use of simulation programs.

C. Localisation

The vehicle achieves localisation through the inertial measurement unit (IMU) and global positioning system (GPS). The IMU used in the project is the Xsens MTi [11] 9 depth-of-field sensor. The sensor contains several advanced internal algorithms in order to provide accurate and noise-free measurements of the current gyroscopic position, the acceleration, and the magnetic field. These values are returned by the IMU readings as the velocity, acceleration, and the three rotations (pitch, roll and yaw) along the x, y and z axis. The sensor is used within the project to determine heading, and assist in the calculation of local positions.

The GPS receiver used is the Columbus V-800 GPS receiver. It is used with the GPSd [12] software to parse the National Marine Electronics Association (NMEA) standard data outputted by the GPS unit, and retransmit internally in an easier to use format. The data used from the GPS unit are the GPS coordinates, the ground speed, and the heading. These GPS coordinates are first converted into a local reference frame, as a distance from a datum point. The acceleration and position data from the IMU and the GPS units respectively are fused together using an extended Kalman filter (EKF), producing a value for positioning that has a higher accuracy than GPS alone. This fusion system outputs the filtered position, velocity, and acceleration data which is then fed into the Path Planner and Control modules along with the bearing to ensure that the vehicle can reliably localise itself and obtain accurate readings for the position, velocity and acceleration.

D. *Odometry*

The SAE car has been fitted with Hall Effect sensors that send its data through a comparator and an OR gate, this makes a pulse train where we use an Arduino UNO [13] to count the time between pulses to give angular velocity, which can be translated to meters per second. This gives the car Odometry, in which software will use an EKF to fuse the measurements together to improve their individual measurements and the vehicles localization capabilities. As the Arduino UNO is too slow to control the steering as well as breaking for the SAE Car, we added a second Arduino to do the odometry which then sends the data through serial communications to the main Arduino Nano. This low-level communicates steering and wheel velocity to the Nvidia Jetson TX1 for processing the data through the EK Filter, using a simple car kinematic model. The goal is to achieve the localization with reduced reliance on the GPS.

E. *LIDAR*

The Autonomous Formula SAE vehicle uses an ibeo LUX [14] Light Distance and Ranging system (LIDAR) to sense distance information about the world around it. The LIDAR records the time interval between emission and recapture of thousands of infra-red light pulses to record a stream of 3-Dimensional points. The LIDAR is specifically designed for automotive purposes and is capable of internal data analysis; detecting and classifying objects in its field of vision. The LIDAR connects via an Ethernet switch to the Nvidia Jetson TX1 [15]. A LIDAR reader class receives the serial bytestream which is continuously being transmitted. The data parsing is handled by the parser class which converts the bytestream into Protobuf objects. This format facilitates the storage and sharing of the information to the LIDAR visualisation.

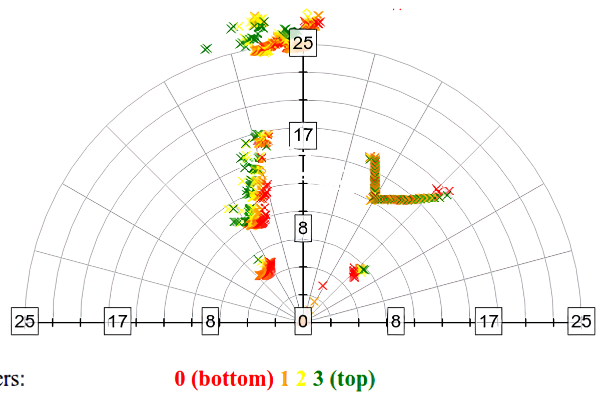

Fig. 3. LIDAR plot showing the detected parked vehicles at the position where Fig. 4 was captured. The graph axes measure distances in metres.

Road-edge detection is achieved through the use of the LIDAR data. The LIDAR, aimed at an angle below horizontal beyond the front of the car, provides four layers of depth information with a horizontal angle of 85 degrees. Its output is visualised as Fig. 3. Road edge detection is achieved by analysing the depth information in one of the layers and checking it for both smoothness and slope. The central data points and those near them are considered and checked to confirm that they meet the slope criteria (the road should be relatively flat so no great changes in depth should be noted in a line) iteratively further and further points are considered in a stepwise process where the correlation coefficient is considered at each point. The road edge is the point at which the correlation coefficient is the highest whilst the slope condition is still being met. This approach was improved with the implementation of a Kalman filter which creates a time-averaged estimate of the road edge-position assisting in the prediction of the current road edge.

F. *Vision*

The SAE vehicle uses visual information as one of its references for driving decisions. It mainly uses an off-the-shelf monocular camera to collect images then applied through OpenCV and SegNet [16] for road edges detection. OpenCV provides many modules, such as image processing, video, and video I/O, that is useful for road edges detection. However, using OpenCV alone for image recognition is limited by variations in image quality, brightness, and contrast. SegNet is an image semantic segmentation approach. It can classify road scene objects, such as the pedestrian, lane marking, traffic light, vehicles etc., that complement the insufficient of single image processing scheme. SegNet is a pixel-wise semantic segmentation in deep learning framework. Semantic segmentation is used to apply for understanding the visual scene and object. This has been widely adopted in autonomous driving. The architecture of SegNet is a convolution encoder and decoder which is a pixel-wise classifier. The objects classify from SegNet is including following classes, sky, Building, column-pole, road-marking, road, pavement, tree, sign-symbol, fence, vehicle, pedestrian, and bicyclist. The accuracy of classify result is 65.9% for classes average [16]. The input images utilise SegNet to perform visual scene classification. This will produce results whereby road, road-marking, and pavement are classified (see Fig. 4), which is useful for road edges detection. OpenCV is simultaneously used to perform image processing. The first step for image processing is camera calibration to get an undistorted image. This is achieved using a chess board image and finding chess board corners to get two accumulated list – 3D point in real world space and 2D points in an image plane. Then we use the camera calibration function in the OpenCV library to obtain the camera calibration and distortion coefficients. This scheme will remove camera distortions.

The road edges detection scheme detects lane-marking at two sides of autonomous SAE vehicle. The lane-marking detection can also be performed solely by the OpenCV library. Finding the edges of the whole image will reduce the

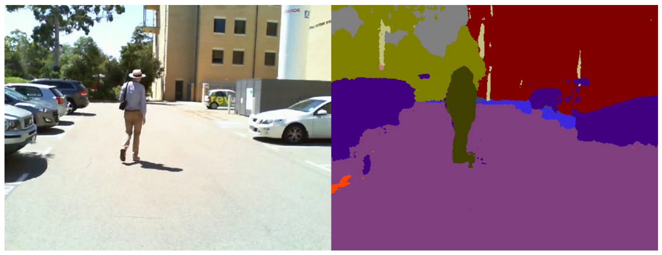

Fig. 4. Segmentation results from a parking area on campus grounds.

image complexity because numbers of colour and gradient of the image would make image processing more difficult. Canny edge detection [17] is a convenient approach in found in the OpenCV library that can be applied for this purpose. Then, Hough transform [18] can be applied onto the image to detect the lanes on both sides of autonomous SAE vehicle. The marking of lanes is detected then using perspective transforms to get a birds eye view-like image. It can easily find the four points to represent the lane marking pair, where a second order polynomial method can be applied to fit the points. The lane distances are obtained using pixel values that are converted into metres. The scaling factors are according to Australia road width standard 3.3 to 3.5 meters. However, the image processing approach might fail because the lane markings are not clear or no lane markings. Therefore, using SegNet's results can effectively circumvent this drawback due to its ability to robustly detect and classify road and lane markings, whereby the same road distance calculation can be applied to find the vehicle's distances to the road edges.

G. Safety Trip Monitor

The safety trip monitor was designed with an observer-notifier structure. Any of the objects responsible for performing a safety-crucial function in the software can call a trip on the trip state monitor. This includes the low-level safety software, the controller, the GPS software, the web interface, the heart beat and the car network. The trip state is stored by the trip state monitor and pushed to any object which implements the trip state observer class and has registered itself with the monitor. The observer class ensures that the trip state does not produce any irregular operations while driving. The observers which receive the trip state upon each change are attached to the monitor after its instantiation, which means that the set of objects in the software which can change the trip state and which need to track to trip state can be completely reconfigured without needing to make changes to the trip state monitor or observer classes. This is an ideal structure for the software of a research autonomous vehicle, as the continuous development of ongoing research will frequently modify the structure of functions of portions of the software while seeking to maintain the integrity of safety features, like the safety trip.

H. Controller

The controller class is the high-level interface for the drive-by-wire functionality of the vehicle. The actual drive-by-wire controlling of the vehicle is done by separate software on an Arduino micro controller. The controller class is the only high-level software which communicates with the low-level controller. The program utilizes three PID controllers to set the throttle, brake, and steering values. The controller class provides a high-level interface with methods to begin to stop autonomous control of the throttle, brake, and steering, as well as methods to set the bearing or speed of the vehicle with desired values. This interface is utilized by the Control program, which handles path planning, and the Fusion program, which provides fused IMU-GPS data in order to facilitate waypoint-based driving. Fig. 5(a) and 5(b) illustrates the baseframe and curvature output by the path planner using the Control program based on our evaluation path in Section III. A large change in curvature is present where the U-turn was made.

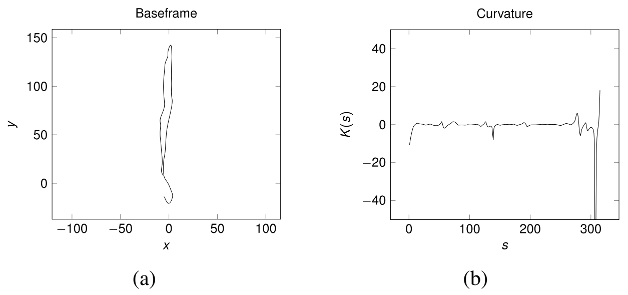

Fig. 5. The trajectory baseframe generated by the path planner (a) and its derived curvature (b).

III. IMPLEMENTATION ON AN SAE VEHICLE

The software for the autonomous driving system is programmed onto an Nvidia Jetson TX1 embedded computer that is mounted on the chassis of the vehicle. The environmental sensors namely the LIDAR, GPS, IMU and camera connects directly to the camera via Ethernet (LIDAR) and USB 2.0 (GPS, IMU, camera) respectively. This software can be implemented onto another vehicle so long as the same sensors are used, as the system outputs drive commands through the Control module, which can be configured according to the vehicle's hardware specifications. To test our system, we collected driving data with the vehicle driving in a parking area at the University of Western Australia (shown in Fig. 6) by recording readings from the GPU, IMU, LIDAR and camera. The GPS and IMU plots waypoints for the car, LIDAR performs obstacle detection, and the camera performs semantic segmentation. The test drive begins from the southern end and then driving towards the northern end before making a U-turn back to the vehicle's station position. The recorded waypoints are passed to the path planner, which generates a trajectory baseframe as shown in Fig. 5(a). Subsequently, the curvature is obtained from the derivative of the baseframe as Fig. 5(b), which can then be used to determine the steering angle for autonomous driving.

IV. RESULTS

Fig. 4 was captured while the car was driving northward as it approaches the end of its path. The input image is displayed on the left and its semantic segmentation result is displayed on the right image. Results from semantic

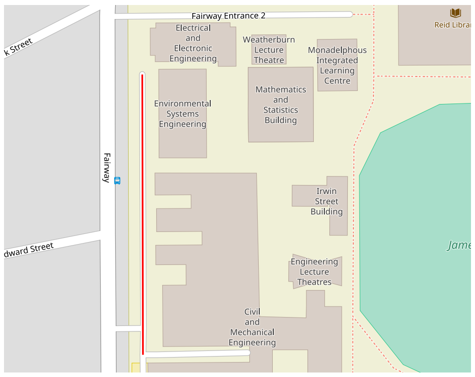
Fairway Entrance 2
Electrical and Electronic Engineering
Weatherburn Lecture Theatre
Monadelphous Integrated Learning Centre
Reid Library
Environmental Systems Engineering
Mathematics and Statistics Building
Irwin Street Building
Engineering Lecture Theatres
Civil and Mechanical Engineering
Fairway
k Street
dward Street

Fig. 6. Map showing the path taken by the vehicle in with a solid red line.

segmentation showed that the road is properly classified, along other elements in the frame. Its LIDAR readings at that position is as illustrated in Fig. 3, whereby the parked vehicles are detected on the left, along with the vegetation in the distance and the wall on the right side of the vehicle. The combination of LIDAR and semantic segmentation enables the vehicle to understand its position on the road, along with the obstacle types and the distances to each obstacle. For further results on road and lane marker detection, we processed a drive recording from Udacity's Self-Driving Car Nanodegree [19]. From Fig. 7, the input image (Origin) is used for both semantic segmentation (SegNet), and bird's eye view (BEV) transformation (Bird view). The system was able to identify objects on the road scene, and the curvature of the road can be calculated using the BEV. The distance between the centre of the vehicle and the left and right lane markers are calculated as shown in the output. This is accompanied with a confidence value whereby a successful detection of the road lane markings will be denoted with a '1'.

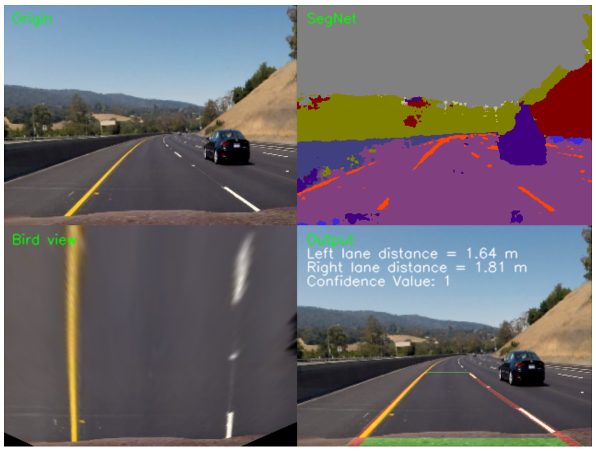

Fig. 7. Original and Jetson TX1 output showing segmentation, birds eye view, vehicle distance to left and right lane marks.

V. Conclusion

In this paper, we have designed and demonstrated the functionality of our software framework that is implemented on our autonomous SAE vehicle. This framework is designed to cohesively interface with the vehicle's camera, LIDAR, GPS, IMU and wheel speed sensors while being capable of performing navigational tasks such as path planning, image processing, odometry, localisation, safety checks, speed and steering control. Each sensor and navigation task is programmed as a separate module to ensure modularity and scalability, allowing for each module to be changed independently. Protocol buffers handle intermodular communications, whereby each process parses its output as a protobuf to be sent to another module. With this framework implemented on the Jetson TX1, our test drives on the autonomous SAE vehicle was able to achieve results that are adequate for fully autonomous driving. Future works will include further testing of the autonomous navigation software and refinements to the control and path planner classes to ensure that the system is capable for road drives using full automation.

Acknowledgment

The authors thank Nvidia Corporation for providing the GPU and the Jetson TX1 under its GPU Grant Program.

References

[1] SAE International, "Student Events - Events - Collegiate Design Series." [Online]. Available: https://www.sae.org/attend/student-events/
[2] T. H. Drage, "Development of a Navigation Control System for an Autonomous Formula SAE-Electric Race Car," Master's thesis, The University of Western Australia, 2013. [Online]. Available: http://robotics.ee.uwa.edu.au/theses/2013-REV-Navigation-Drage.pdf
[3] Baidu, "Apollo." [Online]. Available: http://apollo.auto/
[4] Autoware, "Autoware." [Online]. Available: https://autoware.ai/
[5] Google Developers, "Protocol Buffers." [Online]. Available: https://developers.google.com/protocol-buffers/
[6] K. Chu, M. Lee, and M. Sunwoo, "Local path planning for off-road autonomous driving with avoidance of static obstacles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 4, pp. 1599–1616, Dec 2012.
[7] S. M. Lavalle, "Planning Algorithms," *Journal of Chemical Information and Modeling*, vol. 53, no. 9, pp. 1689–1699, 2013.
[8] H. Wang, J. Kearney, and K. Atkinson, "Arc-length parameterized spline curves for real-time simulation," in *Proc. 5th International Conference on Curves and Surfaces*, 2002, pp. 387–396.
[9] R. P. Brent, *Algorithms for Minimization without Derivatives*. Englewood Cliffs, N.J.: Prentice-Hall, 1973.
[10] J. Xu, W. Liu, H. Bian, and L. Li, "Accurate and efficient algorithm for the closest point on a parametric curve," in *2008 International Conference on Computer Science and Software Engineering*, vol. 2, Dec 2008, pp. 1000–1002.
[11] Xsens, "MTi (legacy product) - Products." [Online]. Available: https://www.xsens.com/products/mti/
[12] E. S. Raymond, "GPSd Put your GPS on the net!" [Online]. Available: http://www.catb.org/gpsd/
[13] Arduino, "Arduino Uno Rev3." [Online]. Available: https://store.arduino.cc/usa/arduino-uno-rev3
[14] AutonomouStuff, "ibeo Standard Four Layer Multi-Echo LUX Sensor | LiDAR | Product." [Online]. Available: https://autonomoustuff.com/product/ibeo-lux-standard/
[15] NVIDIA Corporation, "Embedded systems," 2017. [Online]. Available: http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html
[16] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *arXiv preprint arXiv:1511.00561*, 2015.
[17] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov 1986.
[18] P. Hough, "Method and means for recognizing complex patterns," Dec 1962.
[19] Udacity, "Carnd-lanelines-p1," 2017. [Online]. Available: https://github.com/udacity/CarND-LaneLines-P1