# Autonomous Design Report

ASU Racing Team (ASURT)

## I. Introduction

The research in autonomous systems has shown promising results in substituting redundant efforts done by human beings, whether through providing personal assistants in homes or in factories. One of the fields that exploited the progression of Artificial intelligence and autonomy is the self-driving vehicles, where multiple marvelous results has manifested such as predicting the intent of pedestrian or even other vehicles surround the operating vehicle. Other efforts have been shown in providing high resolution images using simple segmentation maps or even producing hard maneuver through obstacles at very high speeds using driving strategies and policies learnt from data with high versatility.

The potential benefits of AVs include, but are not limited to (I) increased highway safety due to the elimination of human error in driving, assuming that AVs will not be subject to system failures and abuse, (II) better use of traveler travel time for productive work or leisure, (III) independent mobility for older adults, the disabled, and other mobility-constrained population segments, (IV) reduction in fuel consumption and emissions due to smoother acceleration/deceleration characteristics and improved traffic flow characteristics, and (V) increased road capacity and reduced congestion. Moreover, the implementation of AVs will represent a step change in how vehicles operate on the transportation network [1].

The aim of this work is to provide state of the art approaches in solving the problems facing autonomous driving starting from the generalized perception in most of weather conditions and robust to the scale of the image semantic content moving through providing the car with conscious strategic maneuver when required. The work would be tested against standards benchmarks in measuring the performance of self-driving vehicles and partnered with the industrial pioneers in the field of autonomous vehicles.

The paper is structured as follows: section II presents the overall system architecture. Section III presents our method for 2D object detection and 3D localization. Section IV covers both the sensor fusion and SLAM algorithms to get the vehicle state and generating an occupancy grid map. Section V presents the developed algorithms to solve the path tracking problem Finally, section VI covers the software architecture design and how different modules are connected.

## II. High Level Architecture

This section presents the high level architecture of the autonomous system starting from the used sensors ending with the Velocity and the steering angle commands provided to the Vehicle control unit (VCU).

The autonomous system (ADS) consist of 4 sub-systems as shown in the figure 1, *1)Perception* which analyzes the environment using various types of sensors LIDAR (VLP-16) and Mono-camera (FLIR 1.6 MP) as it detect and localize the on-road cones for drivable space estimation, all the detected features and visual cues are arranged in an array of proposals so it can be used by other sub-systems, *2)State Estimation* which integrates the LIDAR and Camera measurements processed by the perception team with the fused output (odometry) of the proprioceptive sensors (Garmine GPS - Vectornav IMU - Wheel Encoders), to achieve a successful solution for the state estimation and SLAM problem, *3)Navigation* which uses the previously developed outputs, the vehicle state and the generated occupancy grid map augmented with the important visual cues in order to generate a free collision path for the vehicle and to calculate the suitable steering angle and vehicle speed to follow the path using our developed lateral controller, and finally *4)Software* team's efforts were centered around designing a computational paradigm satisfying the requirements from real-time response and robust handling for the team's core modules.

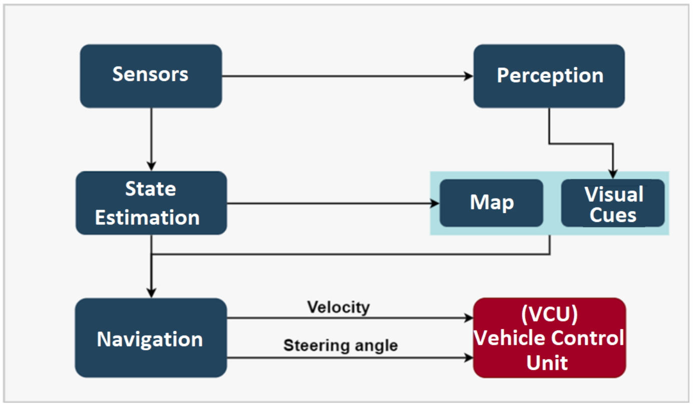

Fig. 1: Overall System Architecture

## III. Perception

Object Detection and Localization can be tackled from many perspectives depending on the inputs. Using RGB images is great for classifying objects but not great for 3D localization. Lidar allows for accurate 3D localization but can not be used to classify objects, especially if the only difference between them is colors. To combine both RGB and Lidar data has many advantages but can be tedious and can result in slow architectures. Our method has the advantage of being simple, faster than real-time, and accurate.

### A. *Method*

As shown in the figure 2 inputs: RGB image and lidar point cloud projected to the front view as 6 channels (3D point location, range, flag whether lidar hit an object, angle). Our method is made of 2 main parts: Segmentation and 3D cone localization.

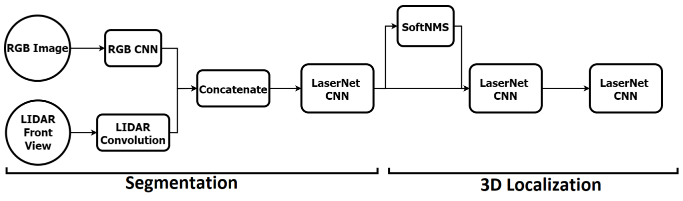

Fig. 2: Block Diagram.

#### 1) *Segmentation*

This section was partly inspired by [2]. We use the CNN part of the architecture but only to do segmentation of the lidar point cloud.

#### 2) *3D cone localization*

For each segmented class, we perform 3D cone localization which takes as input the lidar points that have been classified to such class. To determine the centers of objects we perform 2 operations. The first is an NMS-like operation that chooses 1 point for each object using the object's radius, known previously, similar to using IoU. The second operation is K-means to shift the point towards each object's center. This is efficient because we know K from the previous operation, and we also know an initial estimate for the cluster centers.

### B. *Training Date*

We focus our testing on the task of classifying and localizing traffic cones. We collected our training dataset (1500 frames) using a custom made simulation in Unreal Engine 4 using the environment created by [3]. In addition to collecting the segmentation targets, we also collected the cone ground truths for evaluation purposes.

### C. *Quantitative Results*

The following table I shows the average distance to true cone and the inference time which is measured on Nvidia K80.

| **Methods** | **Average Distance To True Cone (m)** | **Inference Time (ms)** |
|---|---|---|
| CNN | - | 13.0 |
| NMS only | 0.32 | 24.6 |
| NMS + K-means | 0.25 | 26.7 |

TABLE I: Training Results.

The following table II shows the segmentation measurements, the background class was not counted when calculating segmentation values in order not to show over-optimistic results.

| | **mIoU** | **Mean pixel accuracy** |
|---|---|---|
| CNN | 90.9 | 71.7 |

TABLE II: Segmentation Measurements.

### D. *Qualitative Results*

The following figure 3 shows the segmentation results and the predicted cones location.

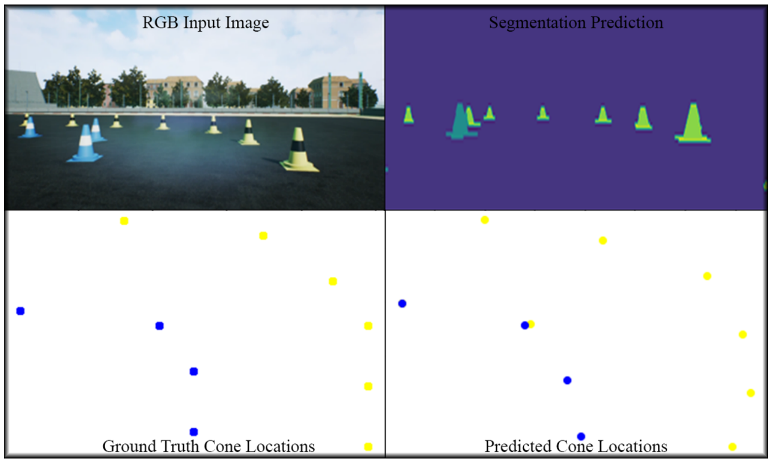

Fig. 3: Qualitative Results.

## IV. State Estimation

This section describes the integration between the LIDAR and Camera measurements with the fused output (odometry) of the proprioceptive sensors available in our car, to achieve a successful solution for the state estimation and SLAM problem. To reduce the computational processing allowing faster navigation, our SLAM algorithm is applied during the first lap or till a successful creation of the map, then shifting to Localization only mode as in [5].

### A. *Odometry Estimation*

In Odometry Estimation section, we will discuss the measurements we used in the fusion and the method of fusion we applied.

#### 1) *Wheel odometry:*

it's computed using the kinematic bicycle model which incorporates the following equations 1 and 2

$$\dot{X}_C = V\cos(\theta+\beta) \quad , \quad \dot{Y}_C = V\sin(\theta+\beta) \qquad (1)$$

$$\dot{\theta} = \frac{V \cos\beta \tan\delta}{L} \quad , \quad \beta = \arctan(\frac{L_r \tan(\delta)}{L}) \qquad (2)$$

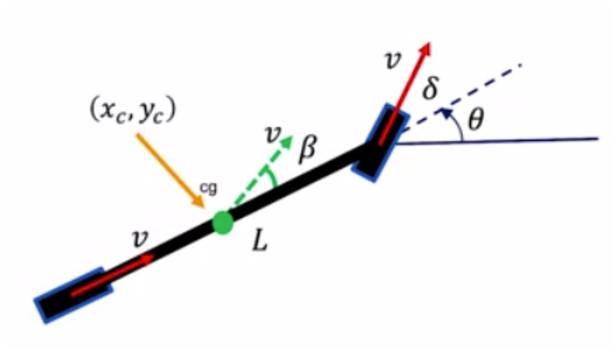

Fig. 4: Bicycle Model.

*2) Sensor Fusion using Extended Kalman Filter:* The EKF algorithm implemented in Robot Localization (ROS package) is used as it fuses the output of the proprioceptive sensors (GPS, IMU, LIDAR odometry, and Encoders) to be fed as an input to the SLAM algorithm.

## *B. Localization and Mapping*

In this section, the algorithms used to solve the SLAM problem will be discussed accurately with adequate redundancy.

Google Cartographer SLAM Algorithm [4] is used integrated with ROS and tested on both EUFS open-source Simulation car on Gazebo and EUFS open-source recorded ROS bags, and it shown an advantage over Gmapping due to the higher accuracy of landmarks detection, and the faster update rate of the map as shown in the table III.

| Methods | Map update (Hz) | landmark position accuracy |
|---|---|---|
| Cartographer | 4.2 Hz | ± 4.6 cm |
| GMapping | 0.6 Hz | ± 7.1 cm |

TABLE III: Cartographer Vs GMapping.

Google Cartographer subscribes on 1) the LIDAR scan message which is processed and filtered from any points beyond the cones. In addition, 2) the accurate odometry estimation resulted from the Sensor Fusion algorithm as shown in the figure 5.

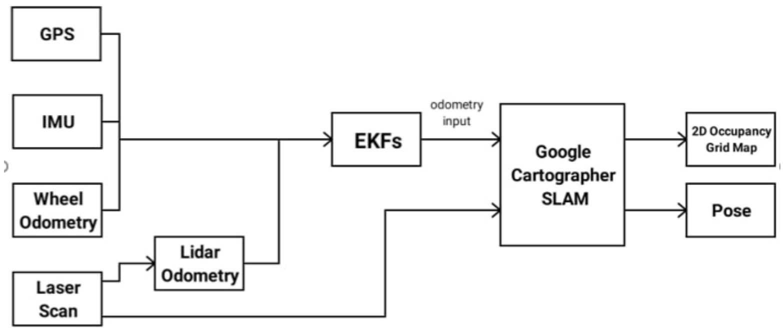

Fig. 5: Block Diagram.

After the occupancy grid map of the track is completed, saved, and drivable space is determined, the system shifts to the Localization Only Mode using Particle-filter based Localization Algorithm, AMCL.

## *C. Results*

In this Section, we introduce our tested algorithms, and how they performed.The figure below 6 shows the output of AMCL Versus the ground truth.

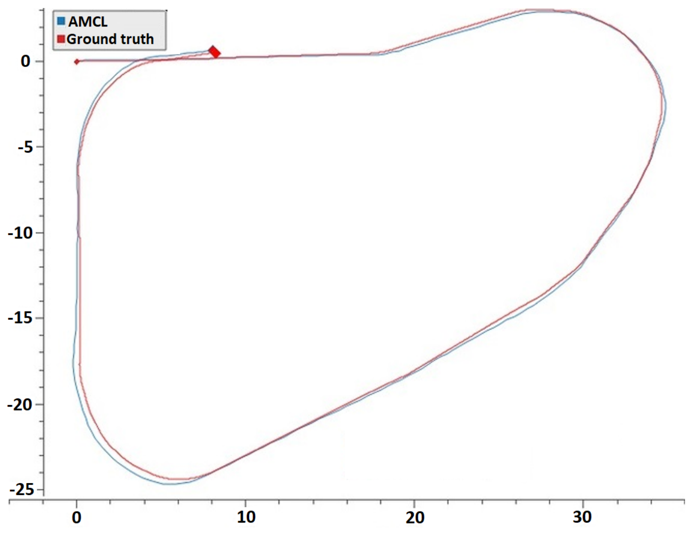

Fig. 6: AMCL VS Ground truth.

## V. Navigation

The use of model-based Reinforcement Learning methods has proven the efficiency in finding optimal policy for endowing complex systems with a certain behavior, however, model-based Reinforcement learning is too sensitive to the model used into finding the optimal policy. Systems are exposed to unpredictable disturbances and modelling errors that might cause the model to drift from the one formulated for finding the optimal policy, which in turns might cause the system to exhibit erroneous behaviour. To address these problems, system identification for modelling purposes but needed to be done per each system. The existence of an oracle that can generically predict the behavior of some class of systems given a sub-sequence for the behavior of a certain system within that class can leverage the consistency exhibited within that class and ease the system identification process, given the oracle, few shot system identification can be realized. This approach is applied on the case of learning generic 4-wheeled vehicle model and using a sub-sequence of observation from a certain vehicle to get the local model.

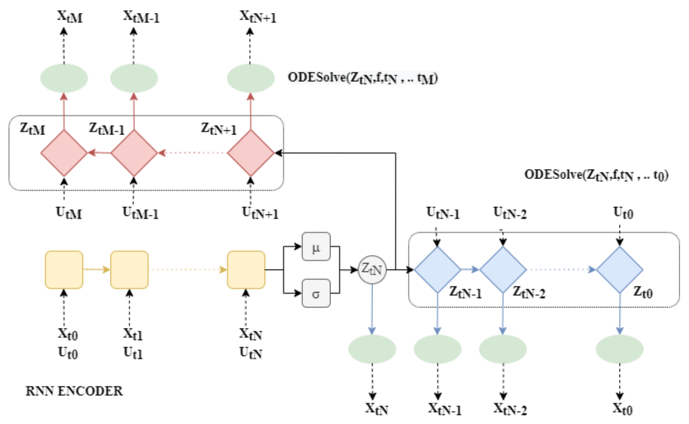

Fig. 7: Navigation Module Architecture

## A. CNODE-VAE: One shot system identification

One of the main success of deep learning is in encoding complex feature and embeddings given the data, this kind of problem is mainly used in text processing specially in sequence-to-sequence text translation where the network encodes the features and the semantics of the sentence in one language in order to translate to another trained on large text corpus. Our model uses the same approach in finding embedding the actual dynamics of the system given few trajectory data which is used then in the predictive model in order to predict the behavior of the system thereafter.
We propose a novel algorithm which formulates such predictive model towards the goal of generic model based reinforcement learning inspired by [6]. The model is split into two main parts system encoder and dynamics model. The first part acts as a generic RNN encoder for the general class and given a sub-trajectory emits an embedding vector that encodes the local dynamics. The embedding vector is as an initial state in a predictive model that is used to fit the differential equation itself called controlled neural ordinary differential equations (CNODE) to predict the response of the actual system.

The predictive model is trained in a Variational Auto Encoder (VAE) format with using ELBO loss.

The model leverages the ability of solving ODEs in both direction of time to achieve real-time performance by separting the operation for reconstruction and prediction. The model achieves $0.5 - 3\%$ RMS normalized over the range of values on non-linear dynamics system including the vehicle as a bicycle model with a pacejka tire model and cart-pole model with friction through simulation and pre-built model in which Perlin noise as random control with temporal structure for better exploration of the state space rather than regular random number generator that lacks structure in time and performs weakly in terms of exploration. The inference time of such model without the ODE45 solver for the learnt ODE (Encoder and Decoder) is almost negligible compared to the time complexity of the Dormand-Prince ode-solver (ODE45).

| | **Inference time (ms)** |
|---|---|
| RNN Encoder and Decoder | 4 ms |
| ODE Solver | 60 ms |

TABLE IV: Real Time performance

## B. Model Predictive Control with CNODE-VAE

The model developed is utilized in a quadratic programming (QP) context to solve for the optimal policy which can be solved in an model predictive control paradigm. The problem in our task is to develop a policy that can finish the track as fast as possible while avoiding hitting the boundaries of the track or the cones. This corresponds to the non linear optimization problem in equations [12] [13] in [7]. The target of the model is to trade off between trying to be as close as possible to the center of the track and also finish the track as fast as possible. The optimal control state variable is lifted to the latent space of the Controlled Neural ODE for the ease of computation of gradients. $\theta_{\mathcal{P}}$ corresponds to progress along the track, while $e^c$ corresponds to the contouring error for following the center-line of the track. The control actions $u$ and states $x$ are bounding in addition to adding bounds for the track In addition to that, the function $F_dec$ that transforms the latent space states and the state space of the vehicle $X, Y$ is chosen to be a linear to ease the linearization of the cost function and representation of constraints and boundaries.

$$\begin{array}{lll} \min & \sum_{k=1}^{N}\left\{\|e^c_k\left(F_{dec}Z_k,\theta_{\mathcal{P}}\right)\|^2_{q_c}\right\} - \gamma\theta_{\mathcal{P},N} & \\ \text{s.t.} & z_0 = z & \\ & z_{k+1} = f\left(z_k,u_k\right), & k=0,\dots,N-1 \\ & F_k F_{dec} z_k \leq f_k, & k=1,\dots,N \\ & \underline{x} \leq F_{dec} z_k \leq \bar{x}, & k=1,\dots,N \\ & \underline{u} \leq u_k \leq \bar{u}, & k=0,\dots,N-1 \end{array} \quad (3)$$

The training of the network is optimized to estimate up to N = 40 steps which corresponds to the time horizon of the MPC controller solver.

## VI. Software Architecture

The software architecture design follows a modular approach taking into consideration the robustness of the system and fulfilling the speed requirements of a real-time system such as our case.

The software design is shown in the following figure 8 expressing the modular design concept followed as well as the different low-level layers servicing our operational modules together with high-level system management levels.

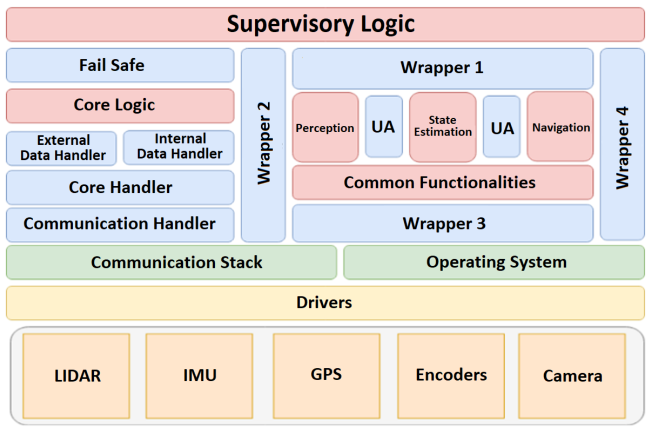

Fig. 8: Software architecture

System modules can be categorized into different classes from core functionalities, supporting functionalities, communication layer, sensor level drivers and sensor devices plugins. The pivotal classes for the system's operation can be illustrated as follows:

### *1) Core functionalities:*

Involve the state-machine supervisory logic affecting the activation of other core modules as well as the main self-driving cars software stack functionalities from perception, state estimation and navigation. Thus, they are responsible for the driving tasks and the decision making scenarios followed.

### *2) Communication stack / Operating system:*

The communication between the drivers of the sensors and the system is accomplished by the communication stack through the mediator pattern. The mediator is a software design pattern designed for the behavioral systems like ours. The pattern encapsulates each part of the system and promotes loose coupling by keeping each part of the system from referring to each other explicitly. It also lets us manage their interactions independently. Some features were added to the pattern to be able to distribute messages to every of specific system modules.

The operating system we have chosen to initiate the nodes of the system and manage the formation of the messages published in the system is ROS version 1. ROS1 was chosen as it matches our demands. It is abstracted from the communication stack and the nodes in it receive the messages from the communication stack via the mediator APIs.

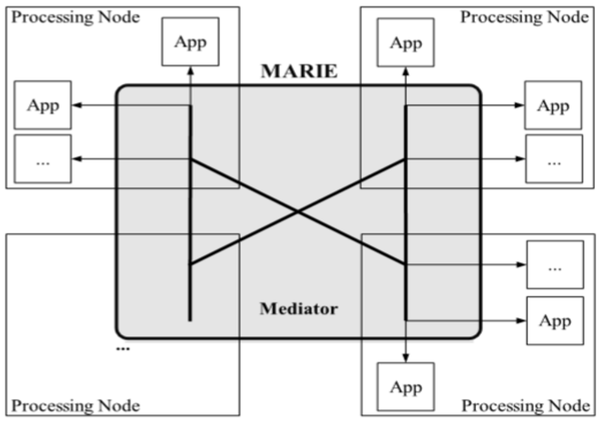

Fig. 9: Mediator extended features

### *3) Communication Handler:*

The communication handler is responsible for the conversion of the messages from the driver format (the message format, the comm stack receives, from the sensor drivers) to ROS message format and vice versa.

### *4) Wrappers:*

The wrappers are a list of APIs to use the functionalities of the communication stack and communicate with ROS1 running threads.

### *5) Drivers:*

The software libraries that enable the robust communication and handling of low-level devices and sensors to provide the system with the raw data needed, as well as the delivery of action messages to be sent to actuators based on navigation module's output.

## Acknowledgment

This research is supported by Autotronics Research Lab (ARL).

## References

[1] Tullio Giuffrè, Antonino Canale, Alessandro Severino, Salvatore Trubia. "Automated Vehicles: a Review of Road Safety Implications as a Driver of Change". 27th CARSP Conference, . Pages: 1–15. 2017.
[2] Gregory P. Meyer, Jake Charland, Darshan Hegde, Ankit Laddha and Carlos Vallespi-Gonzalez. "Sensor Fusion for Joint 3D Object Detection and Semantic Segmentation" In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.
[3] Tom Hirshberg, Dean Zadok and Amir Biran.Formula Student Technion algorithm team. Formula Student Technion Driverless - Based on AirSim.
[4] Hess, Wolfgang, et al. "Real-time loop closure in 2D LIDAR SLAM." 2016 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2016.
[5] Gosala, Nikhil, et al. "Redundant perception and state estimation for reliable autonomous racing." 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019.
[6] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In 32nd Conference on Neural Information Processing Systems, 2018.
[7] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale RC cars," Optim. Control Appl. Methods, vol. 36, no. 5, pp. 628–647, 2015.