Autonomous Design Report
Revolve NTNU Driverless, Car 463

Authors:
Revolve NTNU Driverless Team 2018
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway

Contact Person:
Mathias Backsæther
Email: mathias.backsaether@revolve.no
Phone: +47 47879300

Abstract—**This report gives an insight into the autonomous system of the 2018 Formula Student (FS) team Revolve NTNU Driverless. For the last 9 months, the FS Electric Vehicle (EV), ELD, has been modified to be able to participate in the driverless category of the competition. The results of this work show two different approaches to autonomous driving.**

I. Introduction

Our goal is a top 5 overall score at the FS Driverless (FSD) competitions of 2018. Based on last year's competition results, the most important step towards this goal is to complete all the dynamic events. We therefore make reliability and redundancy our top priorities by thoroughly and continuously testing all our software, and developing separate solutions for each sub-system. A minimum viable prototype (MVP) is implemented to get each sub-system functional as quickly as possible, and at least one high-performance version to be competitive. The different solutions can also be run in parallel during an event and work as back-ups and/or sanity checks for one another.

Section (II) gives an overview of the choices we made regarding the structure of the autonomous system. Section (III) explains our Machine Learning (ML) approach to autonomous racing. Section (IV) then describes the full Driving Pipeline. Section (V) depicts our implementation of the autonomous system and how we managed to test early and often.

II. Overview of the Autonomous System

A. System constraints

The autonomous software system utilizes Ubuntu 16.04 and is powered by an NVIDIA Drive PX2 AutoChauffeur ("PX2"). This platform was chosen instead of a normal desktop PC because of the PX2's processing power compared with its power consumption, as the PX2 is optimized for autonomous driving. We also considered using NVIDIA's Jetson TX2 platform. However, after testing an early version of the pipeline on it, we discovered that we would need a more powerful processing unit and that its reduced power consumption would not weigh up for its limited processing power.

For internal communication between software modules, we use the Robot Operating System (ROS), which is perfect for building a highly modular software architecture. Additionally, ROS has practical visualization tools and an active community that provides support, useful open-source libraries, hardware drivers and complete modules, which makes the development process more efficient.

Communication between the PX2 and the car is done via the two already existing CAN buses. Here, we have the possibility to send messages to the steering actuator and inverters but can also receive various sensor-messages broadcasted from self-designed PCBs. An interface module is implemented to handle two-way communication between ROS and the CAN buses.

B. Machine Learning vs. classical AI and Cybernetics

It was important from an early stage to establish a structure for the autonomous system. That entails everything between raw sensor data and actuating signals. Two distinctively different approaches were considered; a complete ML method, applying either reinforcement or supervised learning, or a complete driving pipeline with several subsystems. The choice between the two systems can be described by an analogy to the works of Daniel Kahneman concerning System 1 and 2 of the human mind [1]. In his book *Thinking, Fast and Slow*, Kahneman describes how System 1 makes the fast and intuitive choices of the mind. In the same way we humans often make choices without active thought, ML systems often make choices we cannot explain. The driving pipeline, similar to System 2, is both slower and more effortful to both develop and compute on-line. An ML approach was considered both easy to implement and effective, but with the downside of black-boxing the whole system. Most ML systems do not give any indication of why their choices are made, making it difficult to both debug and understand. Furthermore, the data logger require information which is not readily available in an End-to-End system. Therefore, it was decided to develop both systems for redundancy. This way it is possible to use one system as a backup during a race.

III. End-to-end

A. Design and function

Our end-to-end system consists of a convolutional neural network that maps raw images directly to steering command predictions. For the network architecture, we use a design heavily inspired by NVIDIA's DriveNet [2]. In the early stages of testing, we accounted for the relatively small amount of training data with minor modifications to the network.

The process of training the neural network consists of several steps. First, we gather training data by driving our car around multiple tracks, featuring various lighting conditions and background noise. During this first step, steering commands and images from the front-facing camera(s) are recorded. We then balance the distribution of steering angles to counter biases in the network. Lastly, the network is trained using a large amount of data augmentation (e.g. random lighting, shadows, translations, etc.). With data augmentation, we reduce overfitting and prepare the network for varied environments.

B. Results

The most up-to-date version of the network is trained on 15000 images and achieves satisfactory generalization, resulting in a network which can maneuver on a priori unknown tracks. Furthermore, the network is not prone to influences from changes in the images unrelated to the track, i.e. background noise and lighting conditions. We achieve a root-mean-square deviation of just 0.19 (steering angles normalized between -1 and 1) with the testing data set. This testing data set consisted of 3000 images randomly selected from the same pool as the training set, however, the data sets are disjoint. Additionally, only 5% of the predictions have a deviation of more than 0.3 from the correct steering command.

IV. Driving Pipeline

A. Detection

Two Blackfly[1] USB3 cameras are used for camera detection, mounted above the driver's seat on the main hoop. The main reasons for this choice of camera are a global shutter and its 648x488 pixel resolution at 84 frames per second. The global shutter is crucial to prevent ghosting and other rolling shutter artifacts, as machine learning may become difficult with these effects. The resolution and frame rate of the camera provide us with enough detail without using too much bandwidth. The camera has a 1/3" image sensor and a 4 mm focal length lens, which achieves a $\sim$93 degrees diagonal field of view. Its 4:3 aspect ratio translates into $\sim$77 degrees horizontal field of view, which doubles when we change from a stereoscopic configuration to a dual camera setup.

Two different systems are developed for cone detection, in addition to an MVP. The first system (the "LiDAR system") uses one or two monoscopic cameras, mounted to maximize the field of view (FOV), in combination with a Velodyne VLP-16 LiDAR. The LiDAR has a $\pm$ 3 cm distance accuracy, and is mounted underneath the impact attenuator. The second system (the "stereo system") uses only two cameras, in a stereoscopic setup. The MVP simply uses an open-source LiDAR obstacle detection ROS node.

The LiDAR system obtains a proposed set of obstacles from a LiDAR detector. The LiDAR detector looks at spatial point clusters to classify regions of interest (ROI). These ROIs are projected to the image plane using the intrinsic and extrinsic camera parameters from calibration. Given the projected ROI and the distance to the obstacle, we can generate an accurate crop of the image which contains the obstacle. Lastly, the obstacles are classified from this crop using a convolutional neural network. The benefits of this system are a high FOV, precise obstacle localization from the LiDAR detection, and few false positives due to the double detection system, at the cost of more false negatives. An example of cones detected by the system can be seen in fig. 1.

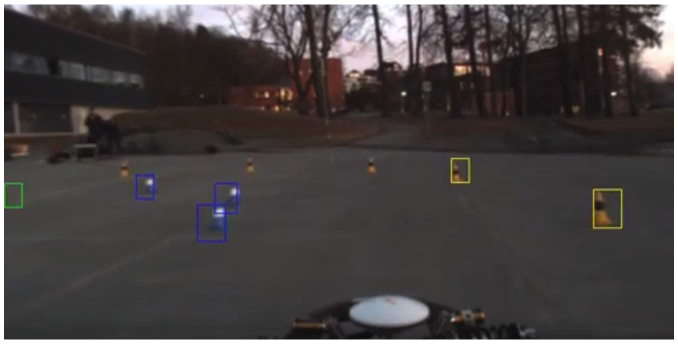

Fig. 1. Cones found by using the LiDAR-based detection system. Cones detected as yellow are shown with a yellow box, cones detected as blue are shown with a blue box, and ROI that are detected to not contain a cone are marked with a green box.

For the stereo system, a fully convolutional neural network for detection of objects is run on each frame. The detection system is based on the network architecture YOLO, and specifically the YOLOv3 [3]. The system is trained on a large dataset of labeled cones. Cones detected in a frame can be seen in fig. 2. The distance to each detected cone is found based on stereoscopic camera calculations, which is less precise than LiDAR distance measurements. This system has a lower FOV than the LiDAR system, but can run at a higher frequency due to the 20 Hz limitation of the LiDAR. Most importantly, with this setup, the LiDAR system can still be used, albeit with a reduced field of vision, in which case we would have a fully functional detection system even if either of the three sensors should fail during an event.

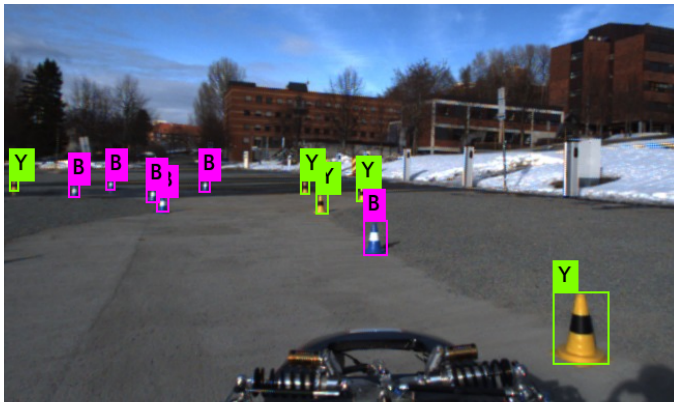

Fig. 2. Cones found by using YOLO detection. The network is correctly distinguishing between the blue and yellow cones. Yellow cones are marked Y, blue cones are marked B.

[1]Blackfly camera model: BFLY-U3-03S2C-CS

B. Navigation and mapping

Our SLAM algorithm use a back-end called iSam2 (incremental Smoothing and mapping 2) [4]. ISam2 is a factor graph based approach that model the world with variable nodes and function factors constraining these variables, see e.g. [5]. Variable nodes are modeled as poses, $p = (x, y, z, \phi, \theta, \psi)$, or landmarks, $l_k = (x, y, z)^T$, whereas function factors model the relationship between consecutive poses and landmarks. The function factors utilize wheel odometry measurements to predict how poses evolve, position from a Swiftnav RTK GPS and a Vectornav VN-200 INS to correct current estimates, and landmark measurements from detection to map cones. This high level of abstraction creates an intuitive and modular system where new sensors are easily integrated.

One of the main innovations of iSam2 is the novel data structure known as the Bayes tree which combines the benefits of a graphical model with that of sparse linear algebra. This allows for incremental inference of the current state and environment at a high rate. High rate translates to an algorithmic complexity of $O(1)$ during exploration and $O(n^3)$ during full factorization, e.g. loop closures, see [4]. Thus, providing faster update rates than methods like Fastslam 2.0 and EKF, see [6] [7].

We use the GTSAM framework to implement iSam2 in the driving pipeline. The computationally heaviest part of this framework, when applied to iSam2, is the graph optimization. Graph optimization entails re-eliminating new and old variables, relinearizing variables and solving a system of equations, see section 5 in [4]. The two former effects are directly controlled by adjusting the re-elimination and relinearization thresholds respectively. The latter is mainly affected by the type of factorization (Cholesky or QR), the variable ordering and the affected number of variables. Based on the results in [8], we use a Cholesky factorization. For variable ordering, we use COLAMD and to affect the number of variables, we have increased the re-elimination threshold to minimize the number of variables in each update.

C. Track Identification

The problem of track identification entails extracting the left- and right hand edge of the track from a set of detected cones. To support the different detection modes, the track identification system must function without relying on cone colour, but should make use of this when available.

The MVP approach is to start with a single known cone on each side of the track, and connect them to their nearest neighbors. However, there is no guarantee that for a given cone, the cone which should follow it will be closest. Even if it is assumed that all cone colours are known, problems arise in hairpin corners and areas where the track doubles back onto itself.

As an alternative to the above approach, a cost function evaluates the likelihood that a cone directly follows a set of three (assumed to be consecutive) others. The cost function takes into account the change in curvature, difference in spacing, whether or not a plausible cone can be found on the opposite side of the track, and the cone's colour (if known).

An early implementation of our track finder algorithm used the above cost function to greedily locate the track, one pair of cones at a time. The major advantage of this approach was that viewing the track as a set of cone pairs allowed the algorithm to simultaneously classify both sides of the track. However, this cone pair based implementation struggled in sharp turns where only one side of the track was visible. In addition, the greedy nature of the algorithm made it especially vulnerable to false positives.

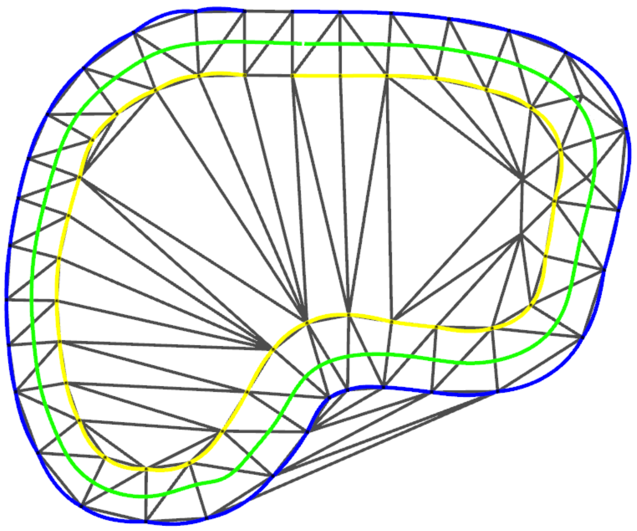

Fig. 3. Track limits, shown as the blue and yellow lines, the center line shown as the green line, and Delaunay triangulation visualized after our system had driven one lap in the Gazebo simulator

The final algorithm builds a directed graph, where each node is a set of three potentially linked cones (a triplet) and each edge is weighted twice (once per side), using the above described cost function. Triplets only have edges leading to triplets, which are continuations of themselves. For example triplet (c_1, c_2, c_3) might have an edge leading to a triplet (c_x, c_1, c_2). Because of the complex and cyclic nature of the graph, each side of the track is found individually, using a series of shallow, recursive, best path comparisons.

To further reduce computational complexity, two additional steps are taken. First, as the car passes a cone, it is "locked", and no longer considered by the classification algorithm. Second, a Delaunay triangulation is performed on the remaining cones to reduce the amount of potential triplets. The result of the track identification algorithm is shown in fig. 3.

D. Trajectory Planning and Control

Two different solutions are implemented for trajectory planning and control, in addition to an MVP. The MVP and the first high-performance solution are divided into path planning, velocity planning and lateral and longitudinal control.

The MVP for trajectory planning uses the center line, which is trivial to find once the track is identified, and sets a constant target velocity. The lateral control system is a simple line-of-sight (LOS) guidance controller which directs the vehicle towards a point on the path a given distance ahead, see fig. 5, in sequence with a PD heading controller. The same velocity controller is used for the MVP as for the first high-performance system, which is described below.

For the first high-performance solution a race-line path is generated by a heuristic optimization problem. A simple velocity profile is generated by first using path curvature to find isolated maximum velocities along the path, and then modifying the speed profile so that it respects acceleration constraints [9]. Lateral control is done using a feedback linearization controller based on a single-track vehicle model and a linear tire model. Velocity is controlled by a PI controller with feed-forward terms for aerodynamic drag and velocity profile gradient.

In the second approach, *Model predictive contouring control* (MPCC) is used to solve trajectory planning and high-level control in a single step. At every iteration, a *quadratically constrained quadratic program* (QCQP) is solved using an interior point solver provided by Forces PRO. This optimization problem takes a mathematical model of the vehicle, as well as the track.

In classical model predictive control (MPC), an objective function is minimized subject to an inter-stage constraint given by the system dynamics. In MPCC, the system is augmented with a virtual input and a virtual state, in order to obtain an objective function that is meaningful in a racing setting. These represent the speed and progress along the track, respectively. With these variables introduced, an objective function can be defined to maximize the progress along the track over the given time horizon. This generates a trajectory that is as close as possible to the optimal race line, see fig. 4.

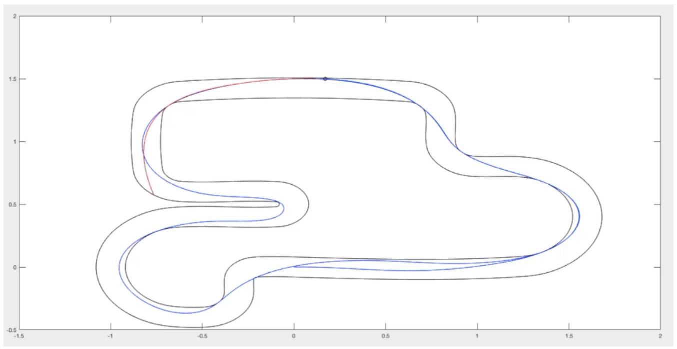

Fig. 4. MPCC visualized. The blue line marks the vehicles path after one lap, while the red line marks the predicted trajectory

1) An a priori trajectory estimate is made using the previous solution as well as the most recent vehicle state measurement.
2) The a priori trajectory onto the track.
3) The vehicle dynamics equation is linearized about the a priori trajectory.
4) The cost function is optimized about the a priori track and the projected arc lengths that are associated with it.
5) The generated QCQP is solved using Forces PRO.
6) The first input in the control input sequence is applied.

The outputs from the high-level controllers are passed on to the mid-level control systems, of which only the control systems for the tractive system needs elaboration. The purpose of these systems is to fully utilize the tire-road friction. The lateral and longitudinal tire forces, as well as the friction limit ellipses, are estimated using the Magic Formula [10] in combination with a complete suspension model to estimate the normal forces for each wheel. The coefficients for the Magic formula are estimated on-line using a gradient approach method.

A torque vectoring module uses an active set solver to find the optimal torque allocation for the individually driven hub mounted motors. This to ensure that the vehicle maintains the yaw rate desired or predicted by the high-level control systems, as well as provide the requested longitudinal force.

Traction control and power limiting are used in post-processing, to avoid excessive wheel spin and power usage, respectively. Additionally, a launch control mode is implemented, which optimize slip ratio during acceleration.

V. Implementation

A. Agile development

The project was organized using principles from agile development, with a SCRUM-like process management.

Iterative testing and development has been the driving force for the project. We used four layers of testing. The two lower levels, *Unit Testing* and *ROStesting* perform boolean tests on single functions and whole ROS nodes, respectively.

The third level is integration testing in a simulation environment. V-REP, Unreal Engine 4 and Gazebo were all considered as bases for the simulation environment. Gazebo was chosen primarily because it is already integrated with ROS, which greatly reduced the lead time. This was the most important factor, as a simulation environment enables rapid and frequent integration testing, which is a huge boost to the iterative development process.

A vehicle model was implemented, with virtual sensors matching those used on the physical system. A script was written which auto-generated random race tracks, marked according to the FS rules. This allowed us to test the robustness of the autonomous system with regards to track layout.

Other simulation tools were used for more specialized development, such as IPG CarMaker for mid-level tractive control systems, and Project Cars for development of the end-to-end system.

The final level of testing is hardware-in-the-loop (HIL) testing. As the base vehicle, ELD, would not be mechanically and electrically ready for autonomous driving before April, it was necessary to find an alternate platform for HIL testing. A Traxxas XO-1 1:7 scale radio controlled race car was procured for this purpose. An NVIDIA Jetson TX2 processing unit and a ZED stereo camera was mounted on the large and relatively flat chassis structure, which was enough to train and successfully run the end-to-end system already in November.

The pipeline system was continually tested on this platform, and new sensors were added as they arrived.

An example of the value of the Traxxas car for hardware testing is the camera choice. The ZED stereo camera seemed like a good candidate, seeing as several teams used it for last year's competition. However, during testing with the Traxxas car it quickly became evident that the rolling shutter effect blurs the image at higher speeds, which prompted us to chose a camera with global shutter instead.

As an added bonus, the Traxxas gave a continuous sense of progression, which contributed greatly to the morale of the team.

B. Version control

The codebase for the autonomous system is organized and updated through the use of Git Submodules, giving a way of creating releases for the entire system, ensuring compatibility throughout the pipeline. This is done by creating a repository consisting of multiple sub-repositories linked to the parent-repo as submodules, conveniently grouping all correlated repositories under one umbrella. Individual commits and changes within one submodule will not be distributed to the parent-repo until explicitly told to do so, meaning one can develop features without causing poor compatibility with the rest of the system. The result is an intuitive way of creating common releases across multiple repositories.

C. Visualization

The visualization of topic information in the ROS network has been prioritized from the beginning, as both a debugging tool and a valuable asset to demonstrate the inner-workings of the pipeline. Most of this visualization is implemented through the ROS-specific program Rviz.

The concurrent structure of ROS allows visualization nodes to run entirely separately from other nodes, as only access to the ROS network is required to run the visualizations.

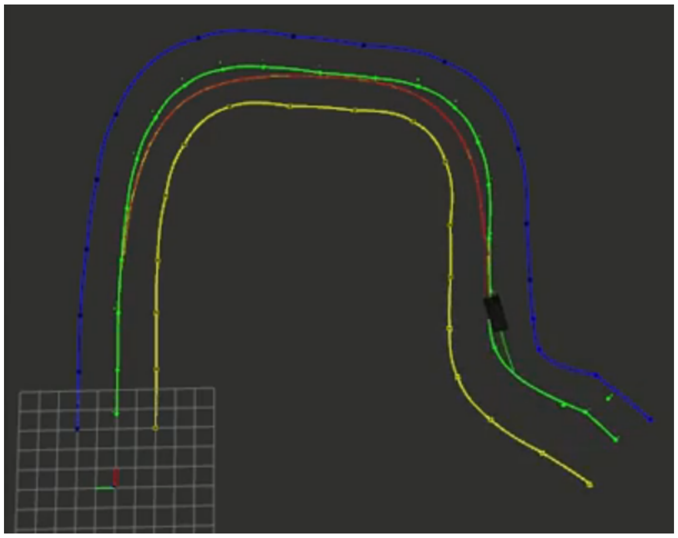

Fig. 5. Visualization in Rviz of track, past and current poses and LOS guidance target heading

Visualization for the different parts of the pipeline is often grouped, meaning that each major subsystem has a combination of visualization topics that best suit the functionality of said subsystem. As of the date of the ADR-delivery, some of the most important visualizations are:

- Mapping: the visualization of SLAM-information, including landmarks, association circles, rejection circles and SLAM odometry.
- Planning: the graphical representation of the pipeline's path planning, including the classification of a track, planned waypoints and splines representing the planned trajectory.
- Vehicle State: the information representing the state of the vehicle in comparison to the surroundings, including position, orientation and driven path as well as control-information such as steering/speed setpoints and control horizon.

VI. Conclusion

This paper presents the state of the autonomous system for the FSD team Revolve NTNU Driverless. With two separate overall autonomous systems, and redundant solutions for each sub-system in the modular approach, our autonomous vehicle is prepared to handle most situations it may encounter. The end-to-end system was the first to drive autonomously and can function during the first round or as a backup. On the other hand, the full Driving Pipeline, with the best performing solutions for each sub-system, will operate at higher velocities and with more precision.

References

[1] D. Kahneman, *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

[2] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self driving cars," April 2016. [Online]. Available: "https://arxiv.org/pdf/1604.07316.pdf"

[3] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," April 2018. [Online]. Available: "https://pjreddie.com/media/files/papers/YOLOv3.pdf"

[4] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "isam2: Incremental smoothing and mapping using the bayes tree," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216–235, February 2012.

[5] F. Dellaert, "Factor graphs and gtsam: A hands-on introduction," 2012.

[6] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *Robotics, IEEE Transactions on*, vol. 32, no. 6, pp. 1309–1332, December 2016.

[7] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *Robotics Automation Magazine, IEEE*, vol. 13, no. 2, pp. 99–110, June 2006.

[8] X. Wang, R. Marcotte, G. Ferrer, and E. Olson, "Aprilsam: Real-time smoothing and mapping."

[9] C. Sprunk, "Planning motion trajectories for mobile robots using splines," Student project, University of Freiburg, Germany, 2008.

[10] H. Pacejka, *Tire and vehicle dynamics*. Elsevier, 2005.