Autonomous Design Report 2024

K. H. K. Liu

University of Leeds
Leeds Gryphon Racing AI
Class: FS-AI
Car Number: 77

AI Computer Platform

The autonomous system is deployed on the InCarPC CQ67G Vehicle PC Base Platform, provided by the Shared IMechE ADS-DV. The system utilises Linux Ubuntu 20.04 as its operating system, installed on a 2.5" SATA SSD[1]. For the software framework, Robot Operating System (ROS) Noetic was selected over other open-source robotics frameworks such as the newer ROS2 Humble. The preference for ROS Noetic, established in 2020, stems from its ongoing maintenance and development by the open-source community, which offers a broad array of available software packages[2].

To streamline the development and simulation of the autonomous system, the team has implemented a Docker container environment. Docker, a virtualisation tool, allows for the operation of a self-contained secondary operating system on the primary system. It does this by sharing access to the system kernel while maintaining direct hardware access for optimal efficiency[3]. This setup has automated the establishment of the necessary development environment, facilitating the installation of Ubuntu 20.04, ROS Noetic, and various software dependencies. Additionally, it offers the benefits of software portability, enabling cross-platform setup and rapid deployment on the vehicle computer[4]. Moreover, Docker enhances code maintenance through a consistent file structure across platforms and allows for quick, clean recovery, enabling easy reconfiguration in the event of file corruption or other complications.

Selected Sensors

The autonomous system developed necessitates a combination of a 2D RGB image feed for object recognition and a 3D point cloud to determine object poses within a three-dimensional space. To meet these requirements, the Intel® RealSense™ Camera D455 was chosen for its robust 2D RGB image capabilities, while the Velodyne LiDAR Puck Hi-Res was selected for its precise 3D point cloud generation.

Additionally, the wheel encoders and steer angle sensors, provided by the ADS-DV, capture wheel speed and steering angle data, respectively. When integrated with the Ackermann Steering kinematics model, these inputs enable accurate computation of the vehicle's odometry for predicting the vehicle's final pose followed by its trajectory.

Overall Software Structure

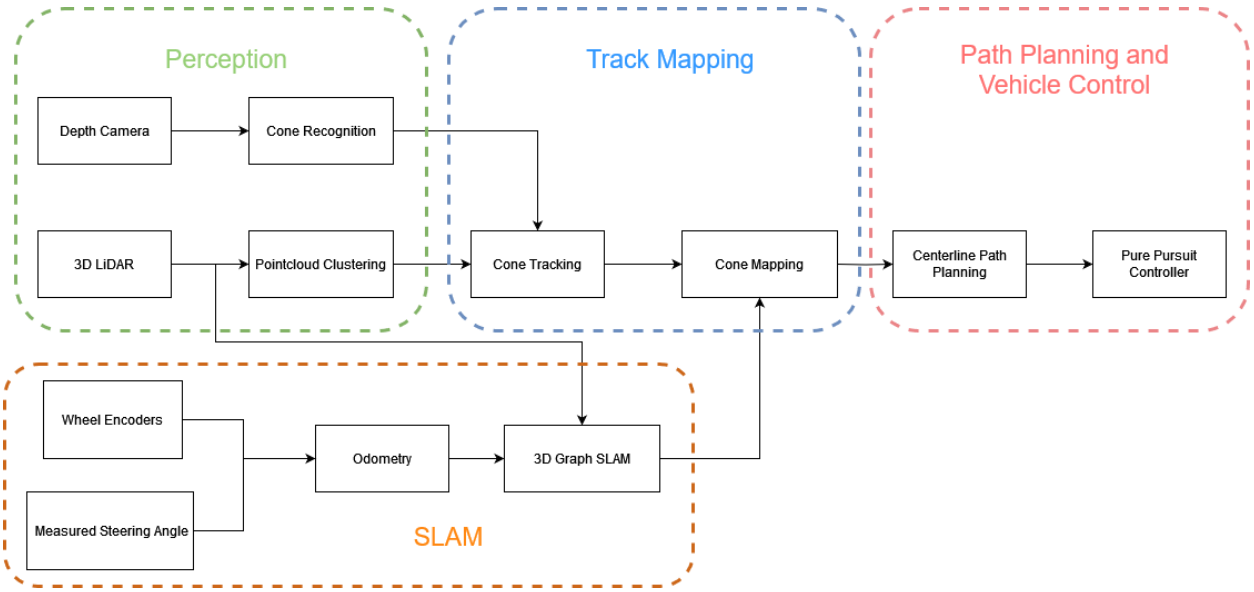

Figure 1 High-level software structure developed to perform the Dynamic Driving Task.

The autonomous system developed is structured around distinct objectives, each fulfilling a role in overall software pipeline. The system is divided into four main components:

Perception: This subsystem is responsible for extracting and processing raw data from onboard hardware sensors into actionable information. It uses depth cameras and 3D LiDAR for recognising cone types and performing point cloud clustering, thereby converting raw data into executable information for the downstream processes.

Track Mapping: Utilising the data processed by the Perception module, Track Mapping integrates this information to register the 3D locations of cones on a global map.

Simultaneous Localisation and Mapping (SLAM): This subsystem leverages vehicle odometry—which includes wheel encoder data and measured steering angles—and the 3D point cloud data to relocalise the vehicle within a previously known SLAM map. This continual update of position and orientation assists in accurate cone mapping and path correction.

Path Planning and Vehicle Control: Based on the track map created by the Track Mapping process, this component computes the centreline of the track. It then employs a Pure Pursuit control strategy to generate the necessary steering commands to guide the vehicle along this planned path.

The following sections will explore the specific algorithms employed in each of these subsystems, evaluate their performance, and discuss the rationale behind the selection of each method.

Perception

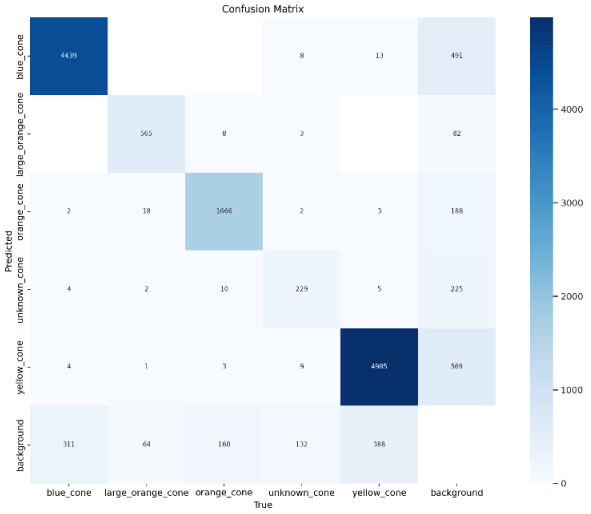

Figure 2 Confusion Matrix derived from YOLOv8n model training and validating process.

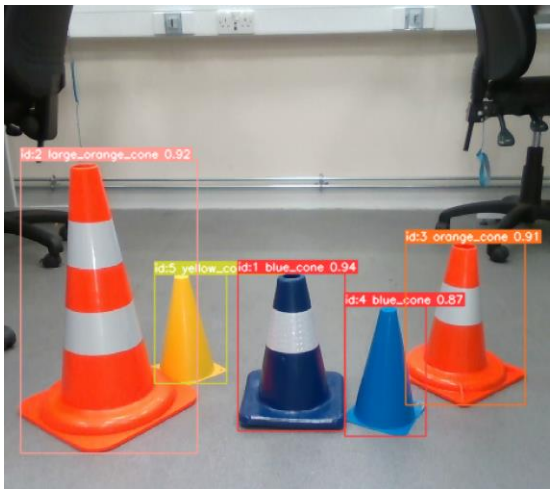

Figure 3 Verifying YOLOv8n model's capability in detecting coloured cones of different forms, as included in the expanded dataset.

The autonomous system's perception capabilities are powered by an object detection module utilising the YOLOv8n model, trained on a combined dataset of FSOCO and an additional set prepared by the team to accommodate cones that have different forms[5][6][7]. The dataset consists of 4 coloured cones classes, including blue, yellow, orange, large orange, and unknown types, contributing to a total of 5762 images. This model was trained over 40 epochs with a split of 70:30 for training and validation. The effectiveness of the model is quantified by a confusion matrix, which illustrates a model accuracy of 81.46%, calculated as the sum of true positives across all detection instances. The matrix further reveals high precision in detecting yellow and blue cones, although some misclassifications were observed in the background where no objects are present, indicating areas for potential improvement in model training.

In conjunction with 2D object detection, the system processes 3D point clouds to estimate the positions of cones in three-dimensional space, with the aim of associating these estimated positions with the cone types predicted from 2D detection. This process initiates with voxel grid filtering to downsample the 3D point cloud, effectively reducing computational load[8]. The downsampled point cloud is then aligned with the depth camera's coordinate system using transformations facilitated by ROS tf library and subsequently projected onto the camera's image plane[9]. This projection necessitates the use of camera calibration parameters, which include the camera's extrinsic, intrinsic, and distortion characteristics, managed by the ROS image_geometry library[10].

Following alignment, the point cloud corresponding to each detected bounding box from the YOLO model is further refined to retain only relevant points. These points are then subjected to Euclidean clustering to isolate distinct objects, thus providing precise information on the location of detected cones in relation to the camera frame[11].

The accuracy of projecting the 3D point cloud onto the camera's 2D image plane depends heavily on the precise mounting of the depth camera and 3D LiDAR, as this allows ROS tf to accurately render the projection image. To minimise errors associated with mechanical calibration, an alternative approach has been explored. This method involves utilising the depth cloud from the depth camera as a substitute for the 3D LiDAR point cloud. While this approach simplifies the mechanical setup and reduces calibration errors, it comes at the expense of decreased accuracy and range compared to using 3D LiDAR.

SLAM

Wheel-based odometry calculates the movement of a vehicle by measuring the rotation of its wheels, providing estimates of its trajectory and speed based on the distance and velocity at which the wheels turn. In this system, odometry data is acquired through the four vehicle wheel encoders and the measured steering angle provided by the ADS-DV. The forward kinematics of the ADS-DV are computed using a model of Ackermann steering dynamics thereby associating the vehicle's pose within a global coordinate system.

Several localisation algorithms were explored to enhance the accuracy and reliability of the SLAM subsystem, each presenting unique advantages and challenges. The first approach, Range-Only SLAM (RO-SLAM), estimates the vehicle's pose based on the locations of static beacons, which in this case correspond to the positions of the cones detected in 3D space[12]. This process utilises vehicle odometry in conjunction with Adaptive Monte Carlo Localisation to align current beacon scans with an existing map[13]. The principal benefit of RO-SLAM is its direct complementarity to the cone mapping software, significantly reducing computational demands[14]. However, this method presupposes a high degree of accuracy in the initial cone mapping, as it does not compute the noise covariance in this process. Moreover, the sparsity of cone distribution tends to increase reliance on wheel-based odometry, thereby amplifying the overall uncertainty in localised pose estimation.

The second approach utilises 3D graph SLAM, adapted from the "hdl_graph_slam" ROS package, which was developed by researchers at Toyohashi University of Technology[15]. This method employs normal distributions transform (NDT) scan matching, which segments space into a grid; each cell within this grid houses a probability distribution modelling the 3D point cloud data[16][17]. This approach facilitates precise alignment of new scans with pre-existing maps, thereby enabling accurate updates to the vehicle's position and orientation. A significant advantage of this method is its utilisation of the range and accuracy provided by 3D LiDAR. However, the effectiveness of this SLAM implementation is dependent on the presence of sufficient environmental features within sensor range. Due to their small size and sparse distribution, cones alone provide limited locational cues. Therefore, this method requires additional, sizable infrastructure features to perform effectively. As a result, the selection of SLAM method will depend on the characteristics of the venue.

Track Mapping

To map the overall layout of the track, 3D bounding boxes identifying cones are initially transformed from the camera's coordinate frame to a fixed global reference frame. This

transformation utilises outputs from both odometry and SLAM systems, with the global frame established at the onset of the autonomous mission. Once transformed, all perceived cones are registered within this frame according to their global coordinates and identified types.

To assess the effectiveness of this subsystem, a series of simulation tests were conducted. These tests involved comparing the perceived cone map, generated by the system, against a ground truth map. To ensure the integrity of the test results, upstream sources of noise such as odometry discrepancies and 3D point cloud inaccuracies were minimised. During these tests, the vehicle navigated various track layouts, following the predefined centre paths. It was observed that the cone mapping process achieved an average accuracy of 78.9%. Notably, mapping accuracy was higher on straight segments of the track and diminished during manoeuvres involving sharp turns.

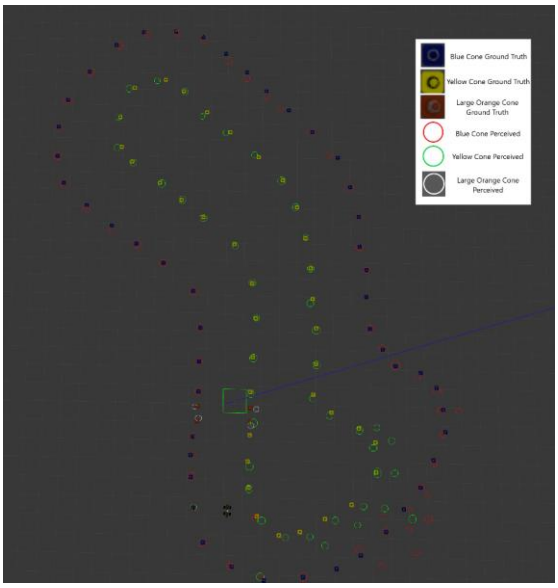

Figure 4 Track Mapping simulation test result. Perceived cone locations are compared against the actual cone locations.

Path Planning & Vehicle Control

The path planning algorithm, developed by The Technical University of Berlin Formula Student Team, was fitted in the autonomous system software pipeline[18]. The algorithm initiates by identifying the starting cones, defined as the pair of coloured cones that are closest to and directly ahead of the vehicle. The algorithm then sequences the cones in order based on the vehicle's current direction, connecting all neighbouring cones.

A breadth-first search (BFS) technique is employed to explore cones that are reachable from the starting point, effectively constructing a logical outline of the track[19]. As the path is developed, cones on opposite sides of the track are paired based on the normal angle derived from adjacent cones. In instances where direct matches between cones are not feasible due to the limited perception range, virtual cones are placed three meters from the normal angle to facilitate pairing and maintain track continuity. Finally, the track's centreline, in the form of a series of waypoints, is computed from the midpoints of these matched or virtually created links.

In the vehicle control module, a pure pursuit control strategy is employed to follow the calculated path. This decision-making process is based on analysing the vehicle's current position relative to the path, with a focus on maintaining a constant cruising velocity[20]. The steering angle required for path following is computed using three key metrics: the beta (β) angle, the cross track error (CTE), and the heading error[21].

The beta angle is defined as the angle between the vehicle's current heading and the line that connects the vehicle's current position to the next waypoint on the path. The cross track error represents the shortest perpendicular distance from the vehicle to the path line between the current and the next waypoints. Lastly, the heading error is the difference between the vehicle's current heading and the desired heading towards the next waypoint. These metrics collectively guide the adjustments to the steering angle, enabling navigation along the path waypoints.

Bibliography

[1] FS-AI, "Remove Linux dependencies," FS-AI_Compute, Issue #1, 2022. [Online]. Available: https://github.com/FS-AI/FS-AI_Compute/issues/1. [Accessed: Jan 7, 2024].

[2] ROS.org, "Distributions," ROS Wiki, 2023. [Online]. Available: http://wiki.ros.org/Distributions. [Accessed: Oct 12, 2023].

[3] Docker, "What is a Container?," Docker Resources, 2024. [Online]. Available: https://www.docker.com/resources/what-container/. [Accessed: Apr 28, 2024].

[4] R. White and H. Christensen, "ROS and Docker," in *Robot Operating System (ROS): The Complete Reference (Volume 2)*, A. Koubaa, Ed. Cham: Springer International Publishing, 2017, pp. 285-307. doi: 10.1007/978-3-319-54927-9_9.

[5] R. Munawar, G. Jocher, and A. Exel, "YOLO Licenses: How is Ultralytics YOLO Licensed," Ultralytics Documentation, Nov. 12, 2023, last updated Apr. 17, 2024. [Online]. Available: https://docs.ultralytics.com/#yolo-licenses-how-is-ultralytics-yolo-licensed. [Accessed: Apr 21, 2024].

[6] G. Jocher, A. Chaurasia, and J. Qiu, *Ultralytics YOLO*, version 8.0.0. Ultralytics, Jan. 10, 2023. [Online]. Available: https://ultralytics.com. [Accessed: Apr 21, 2024]. Repository: https://github.com/ultralytics/ultralytics. AGPL-3.0

[7] N. Vödisch, D. Dodel, and M. Schötz, "FSOCO: The Formula Student Objects in Context Dataset," *SAE International Journal of Connected and Automated Vehicles*, vol. 5, no. 12-05-01-0003, 2022.

[8] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, Shanghai, China, May 9-13, 2011.

[9] T. Foote, "tf: The transform library," in *Proc. IEEE Conf. Technologies for Practical Robot Applications (TePRA)*, 2013, pp. 1-6. doi: 10.1109/TePRA.2013.6556373.

[10] ROS.org, "Image Geometry," ROS Wiki, 2024. [Online]. Available: https://wiki.ros.org/image_geometry. [Accessed: May 7, 2024].

[11] Y. Liu and R. Zhong, "Buildings and Terrain of Urban Area Point Cloud Segmentation based on PCL," *IOP Conference Series: Earth and Environmental Science*, vol. 17, no. 1, pp. 012238, Mar. 2014. doi: 10.1088/1755-1315/17/1/012238.

[12] J. L. Blanco, J. A. Fernandez-Madrigal, and J. González, "Efficient Probabilistic Range-Only SLAM," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2008, pp. 1017-1022.

[13] D. Fox et al., "Monte Carlo localization: Efficient position estimation for mobile robots," in *Proc. AAAI/IAAI*, 1999, pp. 343-349.

[14] K. Zheng, "ROS Navigation Tuning Guide," in *Robot Operating System (ROS)*, A. Koubaa, Ed., vol. 962, Studies in Computational Intelligence. Cham: Springer, 2021. doi: 10.1007/978-3-030-75472-3_6.

[15] K. Koide, J. Miura, and E. Menegatti, "A portable three-dimensional LIDAR-based system for long-term and wide-area people behavior measurement," *International Journal of Advanced Robotic Systems*, vol. 16, Apr. 2019. doi: 10.1177/1729881419841532.

[16] H. Wang, Y. Yin, and Q. Jing, "Comparative Analysis of 3D LiDAR Scan-Matching Methods for State Estimation of Autonomous Surface Vessel," *Journal of Marine Science and Engineering*, vol. 11, no. 4, p. 840, 2023. doi: 10.3390/jmse11040840.

[17] M. Magnusson, A. Lilienthal, and T. Duckett, "Scan Registration for Autonomous Mining Vehicles Using 3D-NDT," *Journal of Field Robotics*, vol. 24, pp. 803-827, Oct. 2007. doi: 10.1002/rob.20204.

[18] FaSTTUBe, "ft-fsd-path-planning: Formula Student Driverless Path Planning Algorithm," 2023. [Online]. Available: https://github.com/papalotis/ft-fsd-path-planning. [Accessed: Feb 21, 2024].

[19] A. Bundy and L. Wallen, "Breadth-First Search," in *Catalogue of Artificial Intelligence Tools*, A. Bundy and L. Wallen, Eds. Berlin, Heidelberg: Springer, 1984. doi: 10.1007/978-3-642-96868-6_25.

[20] L. Bascetta, D. A. Cucci, and M. Matteucci, "Kinematic trajectory tracking controller for an all-terrain Ackermann steering vehicle," *IFAC-PapersOnLine*, vol. 49, no. 15, pp. 13-18, 2016. doi: https://doi.org/10.1016/j.ifacol.2016.07.600.

[21] J. M. Snider, "Automatic steering methods for autonomous automobile path tracking," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08, 2009.

[22] A. Alvarez et al., "The software stack that won the formula student driverless competition," *arXiv preprint arXiv:2210.10933*, 2022. Available: https://arxiv.org/abs/2210.10933

[23] O. Gorriz Viudez, "Enhancement of a Formula Student car perception system using a global 3D map," Bachelor's thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 2022.