Autonomous Design Report FS-AI

Atlas Racing - Heriot Watt University Dubai

Joseph Abdo, Moaiz Saeed, Aditya Shibu, Abdul Maajid Aga and Apsara Sivaprazad

***Abstract*—This report outlines the development of an autonomous system by Atlas Racing (Heriot-Watt University Dubai for the Formula Student UK Driverless (FS-AI) competition 2025. The system uses ROS 2 and integrates a 3d Lidar, stereo camera, GNSS, and IMU for perception and localisation. Google Cartographer was used for 3D SLAM, with an Extended Kalman Filter fusing GNSS and IMU data to produce accurate odometry. Cone detection and mapping were successfully implemented, enabling real-time localisation and navigation in a static environment. The system was deployed on the ADS-DV platform using the Jetson AGX Orin and achieved stable mapping and state estimation performance.**

I. System Architecture

This section outlines the hardware and software architecture of the autonomous driving system deployed on the ADS-DV platform. The entire software stack is developed using the ROS 2 (Robot Operating System 2) framework [1].

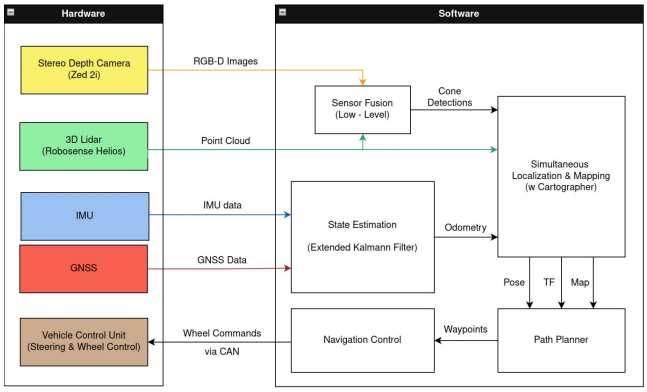

Fig. 1. High-level System Architecture

1) Sensors Onboard: The perception stack includes a ZED 2i stereo depth camera and a RoboSense Helios 3D LiDAR sensor. The stereo camera provides synchronised RGB and depth images, while the Lidar sensor generates 3D point clouds, which allows for accurate spatial perception of the environment. For localisation, the system is also equipped with an IMU (Inertial Measurement Unit) and a GNSS (Global Navigation Satellite System) module [2] [3].

2) Computing Module: All sensor data is processed on the primary onboard computational unit, the NVIDIA Jetson Orin AGX module. It is selected for its high-performance GPU capabilities and low power consumption, making it suitable for real-time data fusion [4].

3) Sensor Fusion: Low-level fusion is performed on the inputs from the camera and LiDAR for cone detection. These detected features are essential for estimating the drivable space and forming a reliable map of the environment [5].

4) SLAM: The system uses an EKF (Extended Kalman Filter) to estimate odometry by fusing data from the IMU, GPS, and LiDAR sensors [6]. The odometry and the cone detections are provided to the Cartographer SLAM (Simultaneous localisation and mapping) framework to perform real-time simultaneous localisation and mapping. The output of Cartographer is a continuously updated global map that reflects the vehicle's environment and position within it [7] [8].

The generated map is fed to the path planning module, which computes a sequence of waypoints to the current navigation objectives. This module sets environmental constraints and vehicle dynamics to ensure the generated trajectory is fail-safe.

5) Navigation Control: The final stage of the pipeline involves the navigation control module, which uses the computed waypoints to derive vehicle kinematics. It then generates wheel commands to appropriately steer, throttle, or brake the vehicle. These commands are transmitted to the ADS-DV (Autonomous Driving Systems-Dedicated Vehicle) platform via a CAN (Controller Area Network) bus interface, completing the control loop.

II. Perception

This section details the implementation of the vehicle's perception system, explaining how it interprets its environment using both LiDAR (laser-based) and vision-based technologies.

A. LiDAR (laser-based)

The RoboSense Helios series Lidar with a 16-beam configuration was selected, offering a balanced trade-off between angular resolution, detection range, update rate, and cost-effectiveness. Filtering and clustering techniques enhance data reliability and remove irrelevant or noisy points that could interfere with path planning. The code detection pipeline using LiDAR follows a multistage approach, as outlined below:

1) Raw Data Acquisition: Data is captured as points in 3D space and transformed to the vehicle's reference frame using sensor_transform matrices. The system implements a thread-based processing with Mut-Ex (Mutually Exclusive) locks to ensure thread safety while maintaining real-time performance.

2) Point Cloud Filtering and Accumulation: Employing several filtering strategies:

- Distance-based filtering:

$$close\_points\_mask = (d < 50.0)$$

  Where **'d'** is the distance to each point, this step filters out all points beyond 50 meters, allowing the system to focus on relevant track sections.

- Height-based filtering:

  $$near\_ground\_mask = (z < h_{\text{ground}} + 0.5) \wedge (z > h_{\text{ground}})$$

  where **'z'** is the z-coordinate of the LiDAR points, and **'h_{ground}'** denotes the estimated ground height.
- Point accumulation: Points from multiple frames are appended to improve cone detection density, especially in sparse or occluded regions.

3) Cone Identification: The cone identification uses DBSCAN (Density-Based Spatial Clustering of Applications with Noise) clustering with the parameters eps=0.5 and min_samples=5; finding this to be the sweet spot, which gave a good balance between noise reduction and the ability to detect distinct clusters in the data points. The team proceeded with DBSCAN over alternatives such as OPTICS (Ordering Points To Identify the Clustering Structure) because DBSCAN offers better computational efficiency at O(n log n) vs. O(n^2) [9], which is critical for real-time performance. Although OPTICS provides more flexibility, since such granularity was unnecessary in this context, where cones are relatively uniform in size and spacing, DBSCAN effectively filters out noise while reliably clustering actual cone detections [10].

4) Cone Validation: Cone detection is done through size validation, ensuring that each cluster matches the expected physical dimensions of a cone. This is followed by confidence scoring, which evaluates point density and other cluster characteristics. Additionally, multi-frame tracking is implemented to reduce false positives and enhance detection stability over time.

B. Image and Object Detection (vision-based)

The object detection system is designed to accurately identify and classify the track cones in real-time while providing precise spatial information for navigation. After several approaches, YOLOv8 was selected as the primary detection model.

1) Model Development and Training: Three cone datasets from Kaggle were created to form a unified training dataset with standardised labelling conventions. The classification scheme used numeric labels (0=orange, 1=yellow, 2=blue, 3=large orange, and 4=unknown) to ensure consistent identification across various racing environments. The training process involved a train-test split of 80% (10,003 images) 20% (10% validation, and 10% testing, 1,250 images each) along with a training configuration with 100 epochs, 32 batches, and 25 early stopping patience [11] [12] [13].

Testing Result: The model accuracy is shown in Figure 1, with appropriate results for yellow and blue cones (highest actual favourable rates). The confusion matrices revealed that most misclassifications occurred between visually similar cone types.

2) YOLOv8 vs. HOG Performance Analysis: On comparing YOLO (You Only Look Once) and HOG (Histogram of Oriented Gradients) with SVMs (Support Vector Machines), referencing Kaplan and Şaykol's research. Although HOG with SVM achieved an 86.7% success rate when trained on

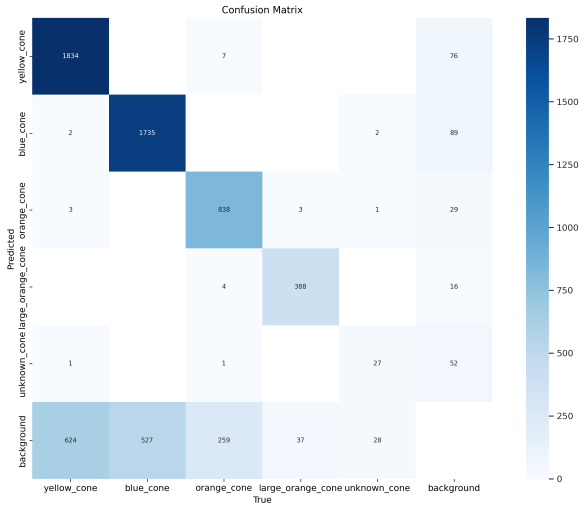

Fig. 2. Confusion Matrix of the trained YOLO Model

80% of the dataset, YOLO demonstrated superior reliability and accuracy in racing environments with significantly fewer false positives (5 vs. 84 for HOG), Higher actual positive detection rate (218 vs. 154 for SVM) and a better overall performance in dynamic racing environments [14].

3) Performance metrics and Evaluation: Model evaluation demonstrated the relationship between confidence thresholds and detection accuracy. The F1 score peaked between confidence thresholds of 0.1 and 0.6. Precision remained high across most confidence levels for all cone classes. Recall started high at low confidence thresholds, stabilised between 0.6 and 0.4, and then dropped at higher thresholds.

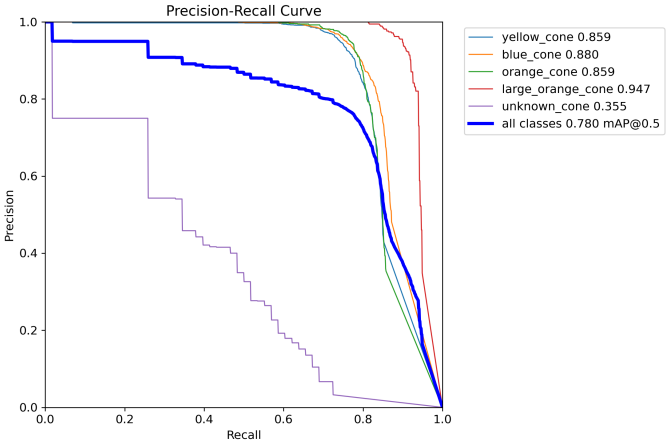

Fig. 3. Precision-recall curve

The precision-recall curve (Figure 2) illustrates the trade-off between detection accuracy and coverage. The training metrics (Figure 3) showed consistent improvement across epochs for bounding box loss, classification loss, and mean average precision (mAP), confirming stable training without significant overfitting.

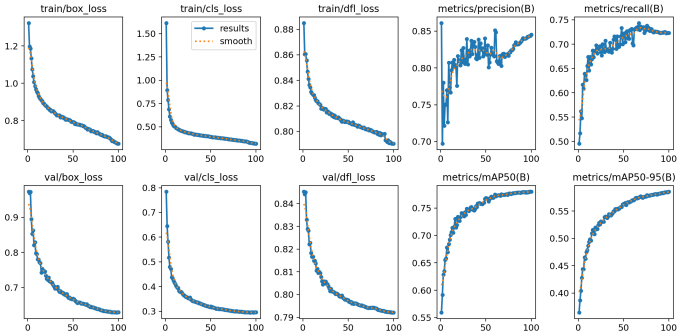

Fig. 4. Training metrics graph

C. Depth Camera (vision-based)

The system uses the ZED 2i stereo camera as a secondary perception pipeline to enhance system reliability through sensor fusion and provide critical colour classification capabilities that complement the Lidar system. The ZED 2i was explicitly selected for its built-in stereo capabilities that provide accurate depth measurements up to 20m, critical for cone detection requirements. The camera subsystem consists of two primary components.

1) Advanced Depth Estimation: A key innovation in the system is a multi-faceted depth estimation approach that combines several techniques: direct depth sampling from the stereo camera's depth map, temporal smoothing using a sliding window of recent detections, position-based depth estimation, and box–size–normalised depth correction. The final depth is computed as:

$$\textit{final\_depth} = 0.5 \cdot \textit{smoothed\_depth} + 0.5 \cdot \textit{position\_depth}$$

This provides more reliable depth estimates than any single method alone. This approach is particularly effective at handling the rapidly changing distances encountered in racing scenarios.

2) Track Boundary Visualization: The system automatically identifies and visualises track boundaries by first classifying detected cones based on colour (yellow, blue, and orange). The cones are sorted by depth to determine their sequence along the track. To visually represent the track edges, lines are drawn between consecutive cones of the same colour. This process provides precise visual feedback that supports real-time path planning and offline validation, helping to ensure accurate and reliable vehicle navigation.

Through extensive testing, it was concluded that a camera-based perception system achieved detection ranges of 1-20m with centimetre-level accuracy after calibration. The integration with the Lidar system through sensor fusion not only improved overall detection reliability but also colour-based cone detection, which simplified the path planning algorithm.

D. Laser-vision Fusion

The system employs a fusion technique to combine LiDAR data with camera detections, leveraging the strengths of both perception systems to create a more robust and reliable cone detection system.

1) Low-level Fusion Approach: While many autonomous racing solutions employ mid-level fusion due to its lower computational complexity, our system implements low-level fusion to achieve peak performance and racing speeds on track. This approach processes raw data from both sensors jointly before making detection decisions, providing key advantages such as enhanced detection accuracy through complementary sensor information, Improved reliability across a wide range of weather conditions, More precise spatial localisation by combining depth information from multiple sources, and reduced false positives through cross-validation between sensors [5].

Although the low-level fusion approach is computationally more demanding, it yields superior detection quality. It is critical for the high-speed racing environment, where precise cone positioning is essential for optimal path planning.

2) Sensor Alignment and Calibration: The fusion pipeline begins with sensor alignment using rigid transformation matrices between the LiDAR and camera reference frames, depth correction factors to address systematic sensor biases, and point cloud projection into the camera's image plane for correlation. These calibration steps ensure data from both sensors is registered in a standard coordinate frame before fusion.

3) Data Association and Fusion Algorithm: The core of the fusion system is a matching algorithm that associates LiDAR-detected cone clusters with camera-detected bounding boxes using the following steps:

- Projection Step: Lidar points are transformed to the camera frame and projected onto the image plane using the camera's intrinsic parameters
- Association: Each 3D point is matched with its corresponding 2D detection using a weighted distance metric that considers both spatial proximity and confidence scores
- Confidence Fusion: For matched detections, a confidence-weighted averaging is performed using the equation:

$$P_{\text{fused}} = \frac{P_{\text{camera}} \cdot w_{\text{camera}} + P_{\text{LiDAR}} \cdot w_{\text{LiDAR}}}{w_{\text{camera}} + w_{\text{LiDAR}}} \quad (1)$$

  Where **'P'** stands for the position and **'w'** stands for the weight/confidence [15].
- Classification Transfer: Colour information from the camera detections is transferred to the corresponding LiDAR points
- Unmatched Point Handling: Points detected by only one sensor are preserved but assigned a reduced confidence score to maintain the continuity of detection

4) Adaptive Confidence Weighting: The system employs dynamic confidence weighting that adjusts based on sensor reliability under different conditions. For example, when Camera detection confidence is weighted higher in good lighting conditions, LiDAR measurements are prioritised in low-light environments or at longer distances. Distance-dependent

weighting gives preference to the camera for colour classification and LiDAR for precise positioning, and Historical detection consistency is factored into confidence calculations.

This adaptive approach ensures optimal fusion performance across the diverse conditions encountered during the competition.

5) Performance Improvements: Although the system initially employed a mid-level fusion approach, the decision was made to transition to a low-level fusion due to mid-level fusion causing lower vehicle speeds. Specifically, the mid-level fusion method introduced delays, requiring each sensor to detect and process data independently before fusion. Low-level fusion integrates raw sensor data prior to processing, significantly reducing latency. This shift resulted in several key improvements, including an extended detection range exceeding 20m, cone colour classification accuracy above 98%, reduced overall detection latency, and enhanced tracking continuity during temporary sensor occlusions.

These improvements directly translate to higher racing speeds and more accurate path planning, giving the vehicle a competitive edge.

III. SLAM

State Estimation is used to generate an accurate mapping of the environment containing the cone positions detected during the first run. This mapping could then be used to create an optimized path around the track for the next run.

3D SLAM was implemented using Google's Cartographer integrated with an external EKF. Cartographer was chosen for its compatibility with ROS 2, its capability to subscribe to externally computed odometry and fuse it with Lidar point cloud data to perform real-time localisation and mapping, making it suitable for the static, structured environment. The EKF was chosen over a UKF (Unscented Kalman Filter) primarily due to its lower computational overhead and proven effectiveness in applications where system dynamics can be approximated linearly [6]. In the pipeline, GNSS data is received as global-position messages, which are not directly usable in Cartesian coordinates. These are processed by the `navsat_transform` node [16], which uses real-time orientation from the IMU to convert the GNSS `/fix` into a local Cartesian odometry estimate in the map frame. The resulting GNSS-derived odometry is then fused with the IMU data in the EKF to produce a centimetre-accurate odometry, which expresses the vehicle's estimated pose in the odom frame relative to its own frame. This output corrects for GNSS drift and IMU bias while providing a consistent and continuous pose estimate. Although wheel encoder data could have been used to derive RPM and compute wheel odometry, this approach was not implemented due to the vehicle's physical unavailability during the development and testing phase. Cartographer subscribes to this filtered odometry along with 3D LiDAR scan data. It uses this odometry as a motion-prior[1] between LiDAR scans and performs real-time scan matching to align point clouds and construct a globally consistent map. Cartographer outputs the transformation between the global map and its frames, the optimised pose graph, and a dense environmental sub-map. This approach allows for accurate localisation relative to local odometry and global GNSS constraints, without requiring a tightly coupled GNSS/IMU fusion.

[1]Cartographer estimates motion using odometry and refines it by matching LiDAR point clouds to build a drift-corrected map in real time

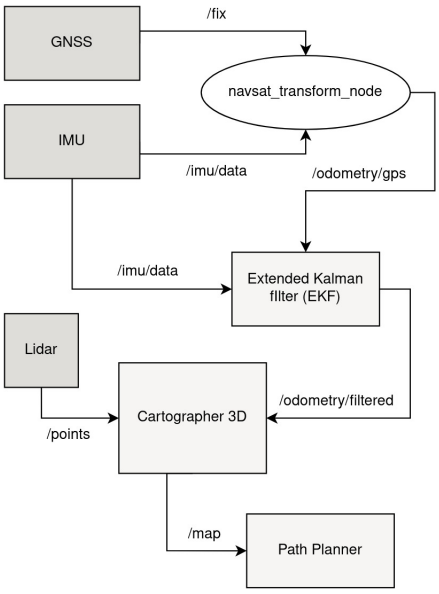

Fig. 5. High-level SLAM Architecture

IV. Path Planning

For path planning, the pure pursuit tracking algorithm [17] was used to compute steering commands required for the vehicle to follow the desired path in real time. The method works by continuously selecting a goal point on the planned path ahead of the vehicle's current position at some lookahead distance $\mathbf{L_d}$. The algorithm features a lookahead distance of 2.5-5m that is adaptively adjusted based on the car's current velocity to improve tracking stability based on the path's curvature. The lookahead distance is calculated by:

$$L_d = K_{ld} \times V + L_{fc} \tag{2}$$

Where **'$\mathbf{L_d}$'** is the dynamic lookahead distance, **'$\mathbf{L_{ld}}$'** is the speed gain constant, **'V'** is the instantaneous forward velocity of the vehicle, and **'$\mathbf{L_{fc}}$'** is the minimum lookahead distance.

Once the goal point is identified, the algorithm determines the required steering angle by using the formula:

$$\delta = \arctan\left(\frac{2L\sin(\alpha)}{L_d}\right) \tag{3}$$

Where **'L'** is the wheelbase of the vehicle, α is the angular deviation between the vehicle's heading and the vector connecting the real axle to the lookahead point, with **'$\mathbf{L_d}$'** being the lookahead distance.

The calculation is repeated for each control cycle, ensuring the vehicle constantly aligns with the updated path based on the latest cone detections. Unlike map-based navigation systems, this implementation operates in a dynamic and unstructured environment where the vehicle relies solely on visual cone detection to build a local track representation. Hence, Pure Pursuit is preferred for its computational simplicity, low latency, and closed-loop geometric control, making it robust to perception noise and irregular cone spacing. Once the vehicle has run through the track once and mapped out the environment, it can move ahead and build an optimised path using algorithms such as MPC (Model Predictive Control).

V. Performance Evaluation and Optimisation Plan

While the autonomous system is fully functional, the team is actively working on the following optimisations to enhance its performance and achieve its full potential. The team has identified areas where the autonomous system falls behind in terms of performance.

A. CUDA for Raw data acquisition

While a multi-threaded approach using mutually exclusive locks is currently employed for transforming 3d point cloud data into the vehicle's reference frame, CUDA (Compute Unified Device Architecture) is being incorporated to enhance the system's real-time performance, scalability, and computational efficiency, particularly as sensor resolution and update rates increase. Benchmarks have demonstrated that GPU-accelerated implementations can outperform multi-core CPU methods by substantial margins [16]. For instance, a study reported that transforming 100 million vertices took approximately 0.096 seconds using CUDA, compared to 0.508 seconds with a CPU-based approach, indicating a speedup of over 5 times [18].

B. MPC

While a Pure Pursuit algorithm is already implemented for path tracking in the system, MPC (Model Predictive Control) will be incorporated to enhance the controller's ability to consider dynamic constraints, improve tracking accuracy, and ensure smoother trajectory following under more variable driving conditions.

The advantage of MPC lies in its ability to calculate high-precision control inputs within a limited prediction horizon, based on the vehicle model and reference trajectory. It uses a mathematical representation of the plant to predict the future, and together with a cost function, outputs the optimal control inputs. The vehicle kinematic or the bicycle model describes the vehicle's motion in terms of position and orientation on the track. Aiming to implement this 'bicycle model' to describe the car's dynamics and assume constant normal tire load, i.e., F_{zf}, F_{zr} = constant, as this model captures the most relevant non-linearities associated with lateral stabilisation of the vehicle.

The following is the cost function in MPC:

$$J = \sum_{k=0}^{N} \left[\underbrace{\lambda_y (y_k - y_{\text{ref},k})^2}_{\text{Cross-track error}} + \underbrace{\lambda_\psi (\psi_k - \psi_{\text{ref},k})^2}_{\text{Heading error}} + \underbrace{\lambda_\delta \delta_k^2}_{\text{Steering effort}} \right]$$

where **'J'** represents the total cost, **'N'** is the prediction horizon, and **'k'** is the current step, ranging from 0 to $N-1$. '$\lambda_{\mathbf{y}}$' is the weight for the cross-track error, while '$\mathbf{y_k}$' denotes the vehicle's lateral position at step k, and '$y_{\mathbf{ref},k}$' is the reference lateral position at step **'k'**. 'λ_ψ' represents the weight for the heading error, with '$\psi_{\mathbf{k}}$' being the vehicle's heading at step **'k'** and '$\psi_{\mathbf{ref,\ k}}$ the reference heading at step **'k'**. Finally, 'λ_δ' is the weight for the steering effort, and '$\delta_{\mathbf{k}}$' is the steering angle at step **'k'**.

References

[1] Open Robotics, "ROS 2 Documentation: Humble Hawksbill." https://docs.ros.org/en/humble/index.html, 2022. Accessed: 2025-05-08.

[2] Stereolabs, "ZED 2i Stereo Camera." https://www.stereolabs.com/en-ae/store/products/zed-2i, 2022. Accessed: 2025-05-08.

[3] RoboSense, "RS-Helios LiDAR Series." https://www.robosense.ai/en/rslidar/Helios, 2022. Accessed: 2025-05-08.

[4] NVIDIA Corporation, "Jetson AGX Orin Series Technical Brief." https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf, 2021. Accessed: 2025-05-08.

[5] M. Pollach, F. Schiegg, and A. Knoll, "Low latency and low-level sensor fusion for automotive use-cases," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6780–6786, 2020.

[6] ROS.org, "robot_localization Package Documentation." https://docs.ros.org/en/melodic/api/robot_localization/html/state_estimation_nodes.html, 2018. Accessed: 2025-05-08.

[7] M. I. Ribeiro, "Kalman and extended kalman filters: Concept, derivation and properties," tech. rep., Institute for Systems and Robotics, Instituto Superior Técnico, 2004. Accessed: 2025-05-08.

[8] ROS 2 Industrial Workshop, "ROS 2 Cartographer Documentation." https://ros2-industrial-workshop.readthedocs.io/en/latest/_source/navigation/ROS2-Cartographer.html, 2021. Accessed: 2025-05-08.

[9] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 226–231, 1996.

[10] E. Schubert, J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Dbscan revisited, revisited: Why and how you should (still) use dbscan," *ACM Transactions on Database Systems*, vol. 42, no. 3, pp. 1–21, 2017.

[11] 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. 1, pp. 12–05–01–0003, 2022.

[12] Trdaxy, "Fsoco-no-unknown dataset." https://www.kaggle.com/datasets/trdaxy/fsoco-no-unknown, 2022. Accessed: 2025-05-08.

[13] L. D. Giustina, "Fsoco bounding boxes yolo dataset." https://www.kaggle.com/datasets/dellagiustinalorenzo/fsoco-bounding-boxes-yolo, 2022. Accessed: 2025-05-08.

[14] Özgür Kaplan and E. Şaykol, "Comparison of support vector machines and deep learning for vehicle detection," in *Proceedings of the International Conference on Computer Science and Engineering (UBMK)*, pp. 1–5, 2018.

[15] M. Siegel and H. Wu, "Confidence fusion," *IEEE Instrumentation & Measurement Magazine*, vol. 5, no. 3, pp. 35–40, 2002.

[16] M. Alzyout, A. A. al Nounou, Y. Tikkisetty, and S. Alawneh, "Performance analysis of lidar data processing on multi-core cpu and gpu architectures," 04 2024.

[17] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," tech. rep., Carnegie Mellon University Robotics Institute, 1992. Accessed: 2025-05-08.

[18] S. Nawfal and F. Ali, "The acceleration of 3d graphics transformations based on cuda," *Journal of Engineering, Design and Technology*, vol. 16, no. 6, pp. 925–937, 2018.