A ROS-Based Stereo Camera and LiDAR Fusion Approach to Obstacle Detection and Avoidance for Scaled Vehicles

1st Neelkant Teeluckdharry
Department of Electrical and Computer Engineering
McMaster University
Hamilton, Canada
teeluckn@mcmaster.ca

***Abstract*—In this article, the McMaster Autonomous Electrified Vehicle (MacAEV), a small-scale 1:10 RC vehicle, is used to simulate manual and autonomous driving modes. Currently, the autonomous driving algorithm solves an optimization problem using integrated data from the onboard Intel RealSense Camera and SLAMTEK A2M8 RPLidar to navigate around detected obstacles. However, this approach is highly computationally expensive, and many embedded systems cannot perform these calculations fast enough to course correct and avoid a collision. To resolve this issue, this article proposes an optimized methodology leveraging a ROS 2 (Robot Operating System) package designed for real-time performance in C++. This efficiently resolves the optimization problem and enhances obstacle avoidance capabilities for the MacAEV.**

***Index Terms*—Autonomous Vehicles, Electrified Vehicles, Data Fusion, Model Predictive Control (MPC), Intel RealSense Camera, SLAMTEK A2M8 RPLidar, ROS 2 (Robot Operating System), Embedded Systems, Computational Efficiency, Obstacle Avoidance**

I. Introduction

Autonomous driving has undergone substantial development over the past few decades, driven by researchers and industry partners alike. However, the testing and development of self-driving algorithms on real vehicles is often impeded by the risk posed to human lives. As such, scaled vehicles have emerged as the most suitable alternative for autonomous driving research, permitting researchers to perform rigorous testing of novel algorithms in low-risk settings.

For local path-planning, scaled vehicles leverage algorithms designed for obstacle avoidance, utilizing sensory modules to first detect objects followed by a control strategy to navigate around them. Light Detection and Ranging (LiDAR), an active sensor, is commonly used as a sensory module for these tasks due to the relability of its point cloud data. LiDARs are classified as either 2D or 3D: 3D LiDARs emit multiple laser beams in three dimensional space, while in contrast, 2D LiDARs produces a single beam at a time and yield a plane scanning. These advanced scanning features of 3D LiDAR scanners can be prohibitively expensive, restricting some researchers from employing scaled vehicles to the field of autonomous driving.

This project is supported by the National Sciences and Engineering Research Council of Canada

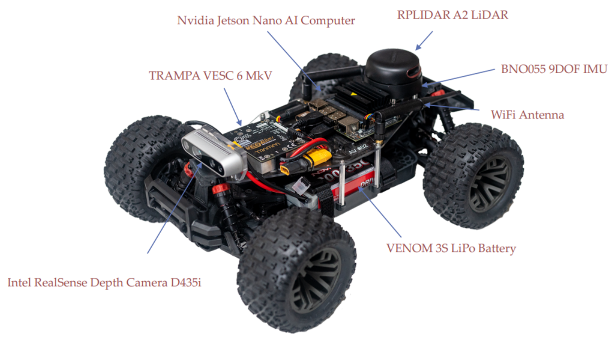

Fig. 1. Shows hardware components of the MacAEV. **Image Credits:** Electrical Systems Integration Project Course.

In this paper, the McMaster Autonomous Electrified Vehicle (MacAEV), a scaled vehicle designed at McMaster University (shown in Figure 1), utilizes sensor fusion to detect and avoid obstacles in real-time along a corridor. It is equipped with both a depth camera and a 2D LiDAR, which are interfaced with an embeded computer through Robot Operating System 2 (ROS 2). The data sampled by both sensors is manipulated and fused in C++, and an optimization-based control algorithm, described in Section II, is applied to avoid the detected obstacles.

The paper is structured as follows: Section II examines related work. Section III discusses the algorithms used to detect obstacles and the control strategy used to navigate around them. Section IV examines the hardware components of the MacAEV and interfacing them through ROS 2. Section V discusses the implementation details of the system with specific insights into the data fusion and ground plane filtering algorithms. Section VI demonstrates the experimental results, showing sucessful obstacle detection and avoidance and Section VII concludes the paper and presents future directions.

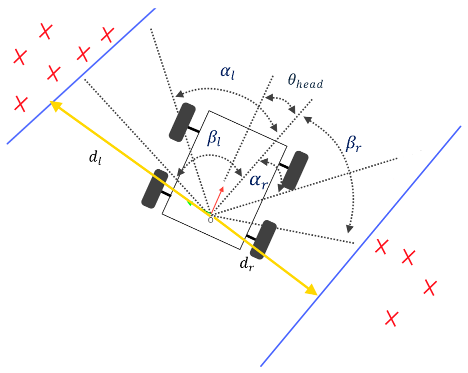

Fig. 2. Vehicle from top-view with labelled parameters used for retrieval of sensory information and control.

II. Related Work

Obstacle avoidance, a basic task in self-driving, is the backbone of autonomous navigation. Approaches to this task evolved from the Bug Algorithm in favor of path planning algorithms such as A* or Dijkstra on a local cost map [1]. In recent times, strategies to this task have now adopted machine learning flavor, giving rise to the optimization-based and model predictive control algorithms. Moreover, a critical aspect of successful obstacle avoidance hinges on accurate detection, for which a myriad of sensors can be employed to continuously scan the surroundings and identify obstacles. The most commonly used sensors for detection are LiDAR and camera, giving rise to LiDAR-based and vision-based approaches respectively [2].

Vision-based approaches use either computer vision (CV) or machine learning, specifically Convolutional Neural Network (CNN)-based algorithms such as in [3]. In the context of small-scale autonomous driving, the primary limitation to this approach is the restricted awareness of the surroundings due to the narrow field of vision captured by the camera sampling data. On the other hand, 2D Lidar-based methods, such as the Follow the Gap Method Method described by Sezer et al. in [4], analyze raw returns to detect obstacles. Yet, relying solely on data sampled from a single plane can lead to potential blind spots and missed obstacles. By integrating data from both systems, this paper aims to overcome the limitations of both approaches.

III. Used Algorithms

The algorithm used for autonomous navigation follow the pipeline shown in the Figure.

A. Calculating Heading Angle

The angle, θ_{head}, represents the desired direction of travel of the vehicle and acts as a reference to separate returns on the left and right sides of the vehicle in any orientation.

$$\theta_{head} = \frac{\sum_i r_i \theta_i}{\sum_i r_i}$$

The LiDAR scans are captured and pre-processed as pairs (θ_i, r_i) of beam angles and corresponding range measurements. The vehicle's heading angle θ_{head} is calculated as a weighted average of the LiDAR beam angles within a pre-defined field of vision:

Two design parameters α_l and β_l, which are angles oriented on the left of θ_{head}, are chosen. The range measurements between these two angles are sampled and used to calculate the left barrier, as described in Section B below. Similarly, on the right side, α_r and β_r are chosen, and the corresponding range measurements are used to calculate the right barrier.

B. Finding the Barriers

Suppose $w \in \Re^2$ is the vector which parametrizes the line separating the Lidar scans from the vehicle. The minimum distance, d, between the origin of the frame, centered at the Lidar, and the line is given by:

$$d = \frac{1}{w^T w}$$

The lines on both sides of the vehicle are constrained to be parallel, to better fit the barriers. The goal is to find a vector w which maximizes the total distance d_{total}, representing the total distance from the frame to the left barrier d_l and frame to the right barrier d_l.

$$d_{total} = d_l + d_r = \frac{2}{w^T w}$$

This gives rise to a convex model predictive control problem with linear constraints or an inequality-constrained quadratic program.

$$\begin{aligned}
\min_{w} \quad & J = \frac{1}{2} w^T w \\
\text{s.t.} \quad & w^T p_i + b - 1 \geq 0 \; \forall i \in \{1, ..., n_l\} \\
& w^T p_j + b + 1 \leq 0 \; \forall j \in \{1, .., n_r\} \\
& -1 + \epsilon \leq b \leq 1 - \epsilon
\end{aligned} \tag{1}$$

where n_r and n_l are the number of points sampled between α_r and β_r and α_l and β_l respectively. b is a parameter representing the translation of the lines and $\epsilon > 0$.

The quadratic program is implemented in C++ by augmenting the vector w with b as a third component. This is accomplished by assigning a very small weight to b in the positive semi-definite matrix Q in $w^T Q w$, which represents the cost function J. The constraints are represented as linear combinations of w_x, w_y and b, which are easily expressible in matrix form. Therefore, the dimension of the quadratic program is $n = 3$, indicating it is small-sized and can be feasibly solved on many embedded computers in real time. In the C++ implementation on the MacAEV, the Dual-Active Set Method proposed by Goldfarb and Idnani in [8] was used to compute the solution to the optimization problem, which demonstrated good real time performance.

C. Steering and Velocity Controller

To drive the motor, the VESC controller is passed an *ackermann_msg* through ROS, which is a time-stamped message containing both the input velocity v and steering angle δ. The MacAEV acheives autonomy through precise control of v and δ within a finite state-machine, where the behaviour of the vehicle changes in response to specific inputs like the detection of an obstacle in-front or behind. The MacAEV's state machine has three states: *normal*, *backup* and *turn*. Developed in this section, are the associated calculations and behavioural descriptions of the vehicle within these states, each of which outputting a corresponding v and δ as *ackermann_msg* to the VESC controller at every time step.

In the *normal* state, the vehicle slows down in the presence of detected obstacles and maintains an appropriate distance from the barriers. Thus, the control objective for steering is to control the distance $d_{lr} = d_l - d_r$ between the left and right barriers. These distances are represented in Figure 2. The non-linear dynamics of these distances are governed by these equations:

$$\dot{d_{lr}} = -v_s \sin\alpha_l - v_s \sin\alpha_r \tag{2}$$

$$\ddot{d_{lr}} = -\frac{v_s^2}{l}(\cos\alpha_l + \cos\alpha_r)\tan\delta \tag{3}$$

Equation (3) arises as a result of the bicycle kinematic model, used to derive the dynamics of the system. Moreover, the tracking error of d_{lr} is minimized using a PD controller, the output of which is used to subsequently calculate the required steering angle δ.

$$\delta_{lr} = \arctan\left(\frac{l(k_p\tilde{d_{lr}} + k_d\dot{d_{lr}})}{v_s^2(\cos\alpha_r + \cos\alpha_l)}\right) \tag{4}$$

These distances can be expressed in terms of the horizontal components of the vectors found by solving the quadratic program in the previous section. Specifically,

$$\cos\alpha_l = \begin{bmatrix}0 & -1\end{bmatrix}\vec{w_l} \tag{5}$$

$$\cos\alpha_r = \begin{bmatrix}0 & 1\end{bmatrix}\vec{w_r} \tag{6}$$

Combining equations (4), (5) and (6) produces the closed form expression used in the implementation of the controller to find δ.

$$\delta_{lr} = \arctan\left(\frac{l(k_p\tilde{d_{lr}} + k_d\dot{d_{lr}})}{v_s^2(w_{ry} - w_{ly})}\right) \tag{7}$$

where w_{ry} and w_{ly} denote the y-component of the right and left wall vectors respectively.

As previously stated, velocity is controlled based on distance to an incoming obstacle. This is given by:

$$v = v_{s_0}\left[1 - e^{-\max(d_{min} - d_{stop}, 0)}\right] \tag{8}$$

where v_{s_0} is the maximum vehicle velocity, d_{min} is the minimum Lidar range reading, and d_{stop} is the vehicle stop distance for an obstacle in front of it. It can analogously be seen as a potential field, where obstacles have a high potential and the vehicle is repelled from them. If the velocity reaches below a defined threshold for a certain amount of time, it indicates that the vehicle has stopped, which triggers a state shift to *backup*.

The *backup* state will cause the vehicle to move with the same velocity scale as in (8), however, in the opposite direction. There are two cases: either the pitch measured by odometry will indicate that the vehicle is oriented away from the closest obstacle or the vehicle will continue attempting to move backwards for a defined amount of time, after which the state will shift to *turn* in both cases.

The *turn* state is also defined with the same velocity scale as in (8). If the vehicle is already oriented away from the closest obstacle, then a state shift will be triggered to return to *normal*. Otherwise, the steering angle will be set to maximum and if the vehicle is still unable orient itself away from obstacles, then it will return to the *backup* state.

IV. System Design

In this section, the hardware architecture and technical specifications of the MacAEV are presented.

A. Hardware Architecture

The McMaster AEV car, powered by a VENOM 3S lithium-polymer (LiPo) Battery, is built on an ARMA GRANITE 4x4 vehicle platform and consists of a multitude of sensors and actuators to interact with the environment. Central to these interactions are onboard computations handled by an Nvidia Jetson Orin Nano embedded computer. Some key features of this device include a microSD card slot, a 40-pin Expansion Header, 4x USB 3.2 Type A ports and a Display Port Output Connector [5]. The microSD card slot allows for Ubuntu 20.04 to be flashed onto the embedded computer through the Jetpack Software Development Kit 5 (Jetpack SDK 5). The USB Type A ports permit for sensors to be serially connected to the computer, which publish data through topics within the ROS 2 middleware. In this study, the data is manipulated within C++ nodes which are subscribed to the relevant topics.

The sensory modules on the MacAEV consists of a SLAMTEK A2M8 RPLIDAR, Intel RealSense Depth Camera D345, Adafruit BNO055 IMU and TRAMPA VESC MARK VI. Key features of the SLAMTEK A2M8 RPLIDAR (shown

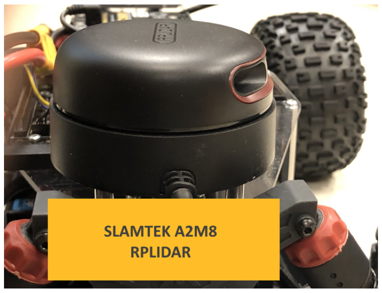

Fig. 3. Shows the SLAMTEK A2M8 attached on the vehicle extremity.

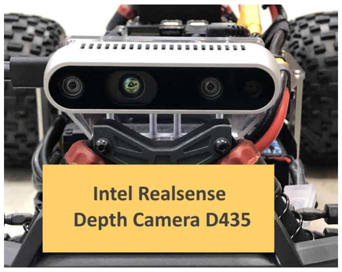

Fig. 4. Shows the Intel RealSense Depth Camera attached to the vehicle extremity.

in Figure 2) include the use of TTL UART to communicate with the embedded computer, a sample rate of 8 kHz, an angular resolution of 0.225° and serial baud rate of 115 200 bits per second. The RPLIDAR is connected to the Nvidia Jetson through a USB type A port and publishes both the angular heading and distance of captured point clouds [6]. Using the RPLidar in ROS 2 requires the installation of SLAMTEK's *rplidar-ros2* package which allows a node capable publishing data to a topic called */scan* to be initialized.

The Intel RealSense Depth Camera D345 shown in Figure 3 combines both RGB and infrared stereoscopic depth cameras with an inertial measurement unit (IMU). Within the camera, the stereo-depth module captures images at 90 frames per second at a resolution of 1280 by 720 pixels, using left and right imagers. Then, data is processed by the vision processor to correct the captured image. Finally, the processed data is stored in the camera's memory through an EEPROM [7]. Using the camera in ROS 2 requires the installation of the *realsense-ros* package, which enables the use of the *realsense2_camera* node. This node publishes color, depth, extrinsic and inertial measurements across eight different topics. The algorithms used for autonomous navigation in this paper rely on depth imaging.

The VESC MKVI, shown in Figure 4, is a motor controller, which offloads the computational expense associated with the control and positioning of the Brushless DC (BLDC) motors and servos in the MacAEV. Furthermore, the built-in wheel odometry allows for estimation of the vehicle velocity, which facilitates the integration of closed-loop control systems for speed control. It is connected to the Nvidia Jetson through a

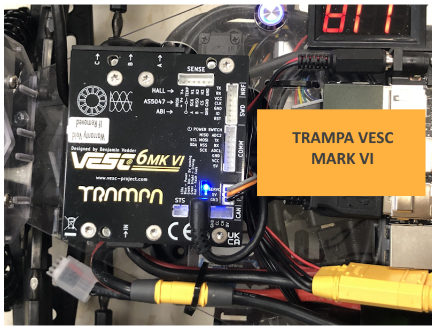

Fig. 5. Shows the TRAMPA VESC MARK VI attached on the vehicle body.

USB Type A port and powered by LiPo battery. Communication between the Nvidia Jetson and the VESC is handled through ROS-based drivers.

V. Software Architecture

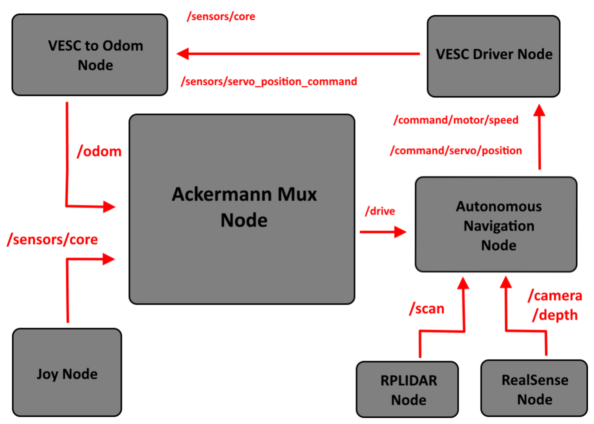

Fig. 6. Shows software architecture for autonomous navigation in the package.

In this section, the software architecture of the MacAEV is presented, along with explanations of key algorithms used to fuse the data from the Lidar and stereo camera.

MacAEV's autonomous navigation software architecture is derived from the University of Pennsylvania's F1Tenth Simulator, an open-source vehicle platform to facilitate driving simulations through ROS. The architecture of this simulator is modified to integrate a navigation node, which publishes ackermann messages to the VESC Motor Controller node when enabled by joystick input. A diagram of the software architecture, illustrating the connections between nodes and topics can be seen in Figure 6.

A. Data Fusion

A key limitation of 2D Lidars is their inablity to detect obstacles below or above their scanning plane. One method of addressing this issue involves the incorporation of a stereo depth camera, and fusing the resulting data obtained from both sensors. This is done through a frame transformation, from the camera's frame of reference to the frame of the Lidar, then comparing the relative distances sampled by both sensors. If the camera depth measurement is less than the Lidar measurement, it indicates that the vehicle is approaching an obstacle below or above the plane of the Lidar scans. In this case, the corresponding lidar scan should be replaced by the camera depth measurement, and the fused ranges are passed in matrix form to the Dual Active Set algorithm.

B. Ground Plane Fitting

The strategy to fuse lidar and camera data is susceptible to errors, as the camera may inadvertently sample ground points due to misalignment, reflective surfaces and other camera intrinsic parameters which create noise. To mitigate this issue, a ground plane is fitted to a selection of the points within a certain threshold using the Random Sample Consensus (RANSAC) algorithm shown in Algorithm 1. This approach leverages the relatively low height of the MacAEV's camera, as the dominant plane in the image will always be the ground.

Each iteration of Algorithm 1 samples three valid distinct random points and the coefficients a, b, c, d with $ax + by + cz + d = 0$ satisfying the points are stored. The number of points which satisfy the plane, num_pts is then compared to num_pts_best and the best plane is appropriately updated. This process repeats until either a maximum number of generations max_gens is reached or num_pts_best exceeds a desired number of points which fit the plane, min_fit.

At each time step, the plane is updated by calling the algorithm and points fitting the plane are omitted from the previously described fusion process. For best results, the hyperparameters max_gens, min_fit, $threshold$ and $height$ require tuning.

Algorithm 1: RANSAC Ground Plane Estimation

```
Data: Image in OpenCV Format
Result: Coefficients $a, b, c, d$ for best fit plane
gens ← 0;
num_pts ← 0;
max_gens ← 20;
min_fit ← 1000;
num_pts_best ← 0;
while gens ≤ max_gens or num_pts _best ≤ min_fit do
    Sample 3 distinct random points in the lower half of image;
    Compute $a, b, c, d$ for plane containing all three points;
    for row = $0 \dots$ cv_sample_rows do
        for col = $0 \dots$ cv_sample_cols do
            if $(row, col)$ has been sampled to form plane then
                continue;
            end
            Deproject pixel at $(row, col)$ using camera instrinsics to find coordinates $x, y, z$;
            Compute distance between point and plane;
            if distance ≤ threshold and y ≥ height then
                num_pts ← num_pts + 1;
            end
        end
    end
    if num_pts ≥ num_pts_best then
        a_best ← a;
        b_best ← b;
        c_best ← c;
        d_best ← d;
        num_pts_best ← num_pts;
    end
    gens ← gens + 1;
end
```

VI. Results (Work In Progress)

VII. Conclusions and Future Work

In this article, a data fusion approach to obstacle detection was proposed for the MacAEV, combining measurements obtained from a 2D Lidar and a stereo depth camera. The inherent noise present in the depth camera measurements required additional filtering, which motivated the described RANSAC ground plane approximation algorithm. Using the fused and filtered measurements, an optimization problem was solved, yielding the vectors parametrizing the lines separating obstacles from the vehicle on both sides. These vectors were used in a finite state machine controller to facilitate autonomous navigation. The approach, implemented in C++, worked in real-time as demonstrated by the results.

Overall, this strategy to local planning represents a proposed alternative to laser depth scanning, promising substantial cost savings for both researchers and partners when prototyping self-driving algorithms. Future work will focus on improving both the software and hardware architecture of the system, testing performance in more complex scenarios and feasibility studies of the approach.

Acknowledgment

The author of this paper extends his gratitude to the National Research Council of Canada for supporting this project through funding, as well as Dr. Shahin Sirouspour and other members of the Telerobotics, Haptics and Computational Vision laboratory for their support.

References

[1] Susnea, I., Minzu, V., Vasiliu, G.: Simple, real-time obstacle avoidance algorithm for mobile robots. In: International Conference on Computational intelligence, man-machine systems and cybernetics. World Scientific, Engineering Academy and Society. pp. 24–29. WESEAS'09 (2009)

[2] D. Li, P. Auerbach, and O. Okhrin, "Towards Autonomous Driving with Small-Scale Cars: A Survey of Recent Development.," CoRR, vol. abs/2404.06229, 2024.

[3] S. Badrloo, M. Varshosaz, S. Pirasteh, and J. Li, “Image-Based Obstacle Detection Methods for the Safe Navigation of Unmanned Vehicles: A Review.,” Remote. Sens., vol. 14, no. 15, p. 3824, 2022.

[4] V. Sezer and M. Gokasan, “A novel obstacle avoidance algorithm: ‘Follow the Gap Method’.,” Robotics Auton. Syst., vol. 60, no. 9, pp. 1123–1134, 2012.

[5] “Jetson AGX Orin developer KIT User Guide,” NVIDIA Developer, https://developer.nvidia.com/embedded/learn/jetson-agx-orin-devkit-user-guide/index.html

[6] RPLIDAR A2 Low Cost 360 Degree Laser Range Scanner, https://www.slamtec.ai/wp-content/uploads/2023/11/LD310 _SLAMTEC _rplidar _datasheet _A2M12_v1.0 _en.pdf

[7] A. Grunnet-Jepson et al., “Intel-Realsense D400 Series Datasheet,” Intel RealSense Technology

[8] D. Goldfarb and A. Idnani, “A numerically stable dual method for solving strictly convex quadratic programs,” Sep. 1983, doi: 10.1007/bf02591962.