MILK: 1/10th Autonomous Racecar

Ajay Mopidevi, Ava Abderezaei, Huilin Han, Rameez Wajid, Siddarth Chintalapati, Venkata Guddeti *

***Abstract*— Autonomous racing is a challenging problem. In this project, we develop a 1/10th scale autonomous car able to safely and reliably navigate a race circuit. The navigation is achieved using the RGB-D and IMU sensors while employing multiple ROS nodes for real-time sensor integration and efficient computation of control actions. We also attempted to undertake certain performance challenges for the race car including; taking and landing a jump, avoiding a ball rolling toward the vehicle, stopping at a stop sign, developing an accurate sparse map of the race course, and real-time estimates of the coefficient of friction.**

I. INTRODUCTION

Autonomous racing is vital for self-driving research. For decades, the automotive industry has relied on racing platforms like Indy Racing, F1, etc., for technical advancements in the field [1].

The advancement in autonomous racing will allow researchers in both academia and industry to develop and test state-of-the-art safety algorithms and other essential features which will result in better adaptation of self-driving technology in the public sector. Also, it may eventually lead to crucial breakthroughs in the fields of control theory and sensor fusion, enabling growth of the automotive industry, while also bleeding the effects to other related fields. An interesting by-product can be the development of autonomous racing as a sport and we may see events like F1 racing becoming driverless.

However, the self-driving / autonomous driving field is still in a nascent stage, and autonomous racing events are still not organized frequently, hence, most of the community relies on simulations to test their designs and algorithms. Recently, the F1tenth autonomous racing platform has become prominent for testing and evaluation of 1/10th scale autonomous cars using low-cost hardware [2].

Autonomous racing is challenging. The unique problems arise mainly from the integration of hardware and software components. The high-speed operation requires real-time sensing and computation capabilities which can be difficult to achieve owing to the presence of uncertainties in sensor measurements and the limited computation available onboard the platform. Another important factor is that a race setting is inherently dynamic and highly uncertain. Hence, the perception, planning, and action loop needs to be robust and accurate for the safe operation of vehicles.

In this project, we aim to develop a 1/10th scale autonomous race car that will have the capability to autonomously complete a race circuit reliably. We also plan to undertake certain performance challenges for the car including; taking and landing a jump, avoiding a ball rolling toward the vehicle, stopping at a stop sign, developing an accurate sparse map of the race course, and real-time estimates of the coefficient of friction.

The paper is divided into seven sections. Sections I and II introduce the problem and some recent solution approaches. Section III discusses the platform design and Section IV presents details of the software implementation. These are followed by testing and discussion. We conclude the paper in Section VII.

II. RELATED WORK

A. Overview

There is a lot of literature surveying the developments and latest methods in autonomous driving [3]. Categories within autonomous driving literature include perception, planning, control, system, communication, testing, and human-machine interaction, with themes such as localization, static and dynamic object detection, tracking, prediction, decision-making, and human behavior. There are many datasets published for autonomous driving, which have contributed to the recent advancements in the field. Newer topics that are being discussed in the field of autonomous driving include ethics and scenario engineering.

Various end-to-end frameworks have been proposed for autonomous racing, which is considered a special case within autonomous driving. A framework [4] proposed for the F1/10 International Autonomous Racing Competition in Torino, Italy uses a ROS-based software architecture mounted on an Odroid XU4 computer and AnyFCF7 board equipped with a 32-bit micro-controller. The software computes odometry for mapping and localization using sensor data from lidar and IMU. The hardware and software setup in this paper is similar to the setup for our project. DeepRacing is an end-to-end framework training autonomous racing algorithms in a simulation environment [5]. It integrates a CNN with LSTM cells and represents spatial-temporal states with optical flow.

B. Perception

Perception is the key upstream component in autonomous driving systems. Most autonomous systems perform perception by receiving sensor data and generating vehicle status and spatial models. Perception includes a few stages: sensing, localization and mapping, and object detection [6].

Sensors can be categorized as proprioceptive sensors for sensing the state of the vehicle, such as IMUs, and exteroceptive sensors for monitoring the exterior environment, such as lidars and RGB cameras. Initially, most autonomous

*Listed in alphabetical order by first names.

vehicles related on vision-based sensors, but in recent years sensor technology and data processing have improved and have allowed the use of lidar and radar sensors for self-driving. The advantage of having a larger suite of sensors is utilizing the strengths of different sensors to compensate for their individual weaknesses. In the situation where one sensor fails, other sensors can be utilized for perception. Another large task in self-driving perception is integrating information from multiple sensors. Algorithms such as Kalman Filters are one way of determining the amount of certainty to assign to data from each sensor and how much they should affect localization, mapping, and decision-making.

Object detection algorithms can be dynamic or static. For example, algorithms for detecting pedestrians and other moving obstacles are generally dynamic, while algorithms for detecting road signs and lane markers can be static. Most traditional obstacle detection algorithms contain two steps: extracting important features from an image and applying a learning algorithm. One of the most widely used features in object detection is a histogram of oriented gradients (HoG), which calculates normalized local histograms of image gradients and is computationally efficient. Object detection often relies on accurate depth information from lidar sensors. Other approaches have also been introduced, such as estimating depth through stereo images captured by pseudo-lidars [7], which are cheaper and sparse lidar sensors. With the massive development of neural network-based vision methods in computer vision, CNN is widely used in object-detection tasks in autonomous driving [8].

C. Decision Making

There are a variety of approaches to decision-making in autonomous systems. Different approaches are also often applied to different parts of the decision-making processes and integrated into the same system.

State machines are a common approach to decision-making. Most autonomous driving systems implement state machines in parts of their system to transition between different tasks. State transitions are generally triggered by perception, such as object detection and terrain identification. One proposed vehicle system integrates a motion finite state machine and a control finite state machine [9]. The two finite-state machines allow the system to respond flexibly to driving in urban environments.

Motion planning is a subtask of decision-making. Common planning methods include graph-based methods, sampling methods, optimization, and deep-learning based methods. One challenge to solve in decision-making for autonomous vehicles is collision avoidance. One proposed approach for personalized collision avoidance is to approach trajectory planning as optimal control by transforming the requirement of passengers into performance index constraints [10].

D. Control

Proportional-Integral-Differential (PID) controller [11] is a classic control approach that adjusts parameters through feedback control. PID controllers can be used for tasks such as line-following, speed adjustment, flight control, and more. PID controllers are linear, thus not the most suited for extremely nonlinear systems. Alternatives to PID control are fuzzy control which adds nonlinearities using rules and fuzzy membership functions to the control logic.

III. DESIGN

A. Hardware

The following equipment was utilized for the project:

- ODROID XU-4 with 2GB memory
- Intel RealSense D435
- 1/10 AMP MT 2WD Monster Truck RTR ECX03028T2
- Mini Maestro 18-Channel USB Servo Controller
- Adafruit IR Distance Sensor GP2Y0A710K0F (qty: 1)
- PhidgetSpatial 3/3/3 Basic 1044/1
- 64GB eMMC 5.0 Module XU3/XU
- 7.2V 1.8A Ni-MH battery
- 11.1V (3S) LiPo Battery

B. Electronics/power system design

To enable autonomous control of the vehicle, a robust set of electronic and electrical components must be deployed. The electronic sub-system includes a power supply, speed and servo controllers, sensors, and onboard processing. The overall wiring diagram is as seen in Figure 1.

The power supply consists of 2 sources: A 11.1V (3S) LiPo Battery, and a 7.2V 1.8A Ni-MH battery. The Ni-MH battery is used to power the electronic speed controller, the rear drive 12T 550 DC motor, servo motor for front drive and the servo controller board. The LiPo battery is used to power the processing unit, Odroid XU4 and the Realsense camera for vision.

An LM2596 step-down converter was used in order to supply the required 5V to the processing board.

To enable the robot's autonomy, a sensing/feedback option was deployed in the form of a depth camera, driven/monitored by the Odroid. The Odroid receives and sends signals to and from the camera and the Pololu servo controller via USB.

C. Mechanical Design

The requirements for the mechanical design involved the design and manufacture of the mount for the Intel real sense depth camera and also a platform that can be placed on the chassis which can hold the Odroid, IMU, and the mount for the depth camera. The chassis already came with a bumper, so, building a bumper was not necessary. The platform was made out of an acrylic sheet which was then laser cut to the appropriate shape and the mount for the depth camera was 3D printed using PLA filament. The CAD models for the mount and the platform are given below.

IV. METHODS

The challenges we approached are the following:

- Taking and landing a jump
- Avoiding a ball rolling toward the vehicle
- Stop at a stop sign

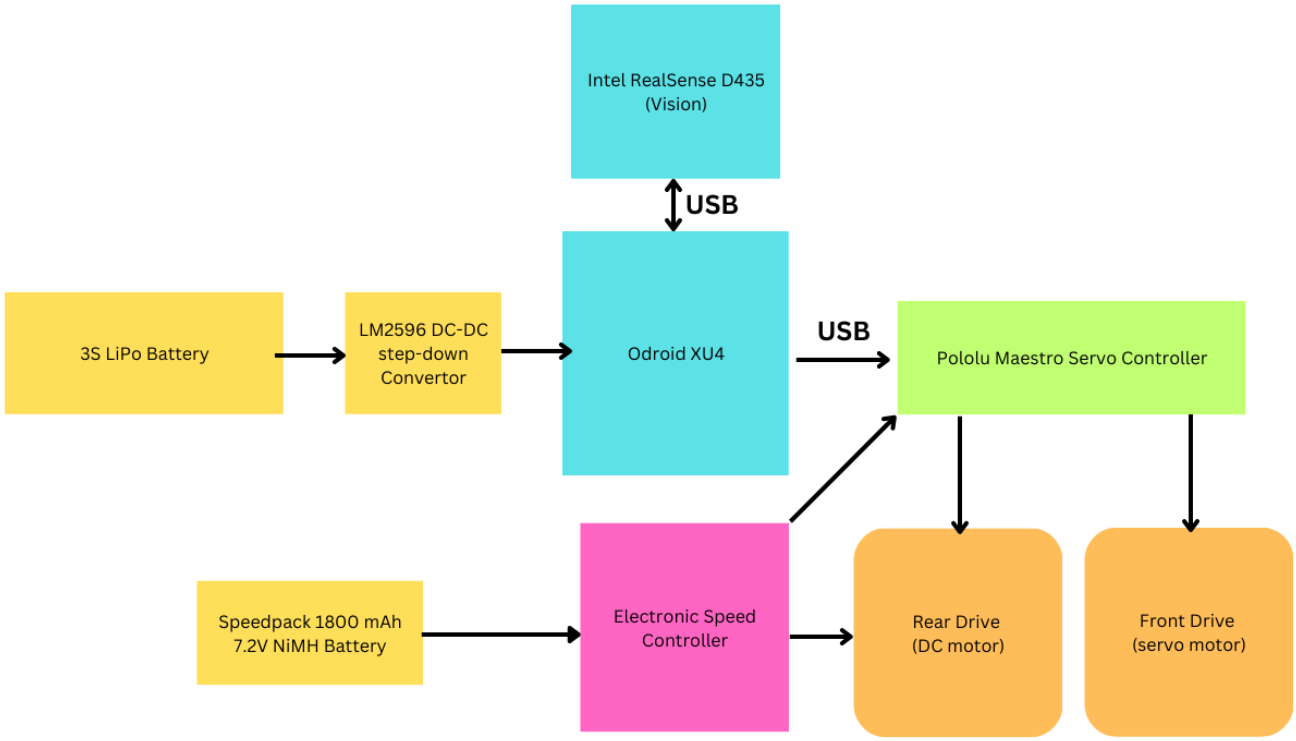


Fig. 1: Wiring Diagram

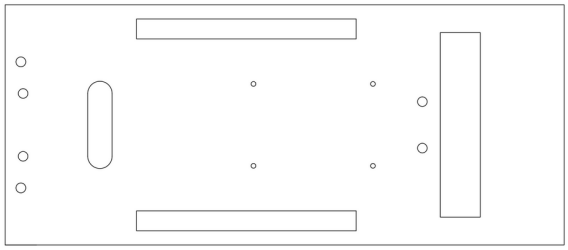


Fig. 2: The platform for the chassis

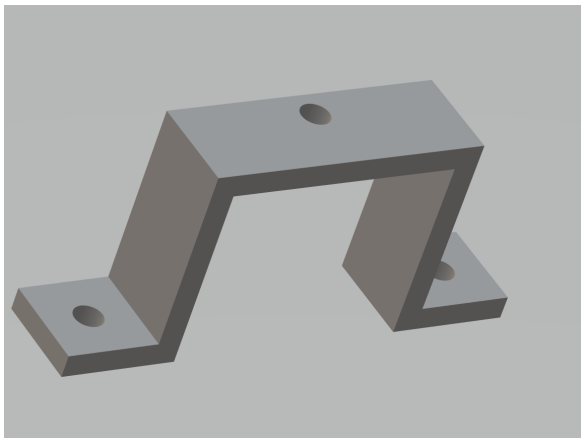


Fig. 3: Mount for the Real sense depth camera

- Developing an accurate sparse map of the race course
- Real-time estimates of the coefficient of friction

The code architecture is defined by taking the challenges in mind. The high-level architecture is shown in figure 4.

Each box in the figure represents a ROS node and the arrows demonstrate the publisher-subscriber relationship. The blue boxes are our sensor inputs, the pink boxes represent the low-level strategic planning, and the yellow box represents the node that controls our Maestro Pololu. A summary of our code architecture is presented as follows:

Our vehicle is equipped with a spatial Phidget and an Intel RealSense depth camera sensor. Hence, we have two ROS nodes that publish IMU and depth sensor data. The published depth data is analyzed in the Depth Analyzer node and is converted to meaningful data which will be used later in our planning algorithm. The Friction Coef node uses the data published by IMU and the Depth Analyzer to calculate the friction coefficient by moving the car. Ball Detection and Stop Detection nodes subscribe to the Depth camera node and publish whether they detected a ball and a stop sign respectively. Auto Drive node subscribes to Depth Analyzer and publishes the corresponding commands to move the car. Stop at Sign is similar to Auto Drive however it also subscribes to the Stop Sign detection node and will stop at stop signs. Avoid Ball node subscribes to the Ball Detection and Depth Analyzer to move the car while avoiding the ball. All the published planning commands from Avoid Ball, Auto

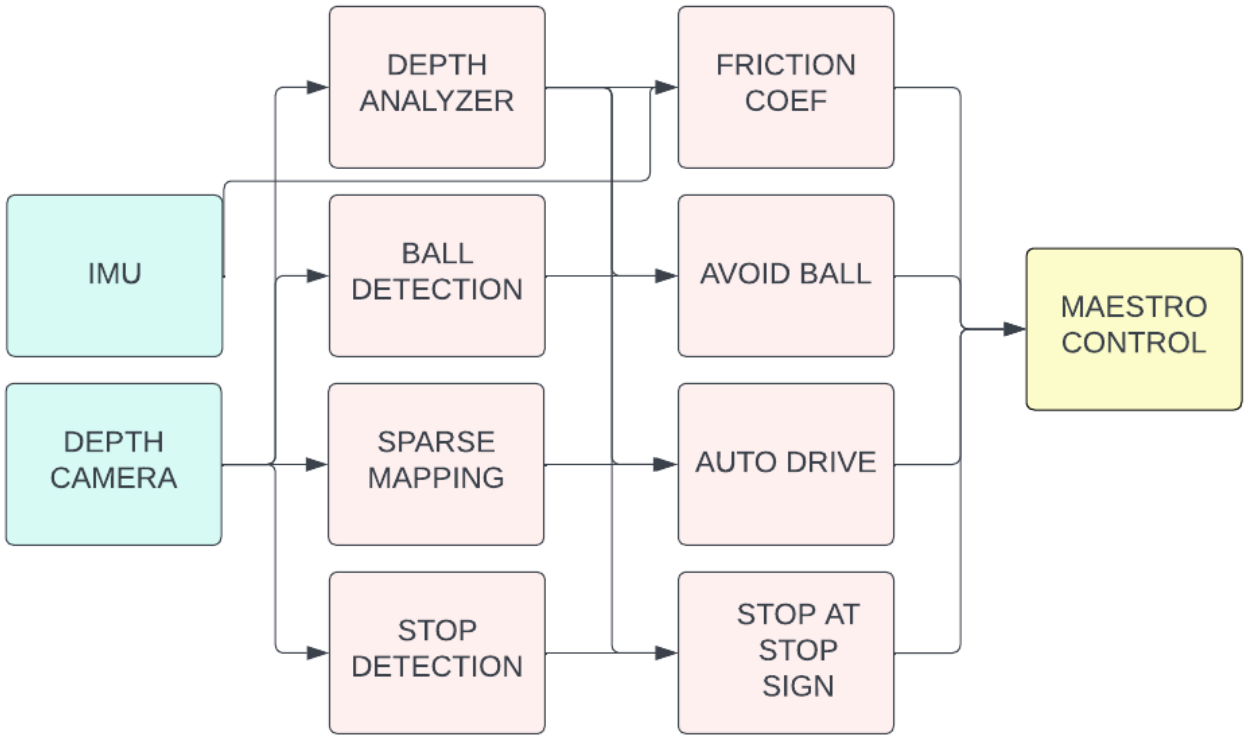


Fig. 4: Code Architecture

Drive, Stop at Stop Sign, and Friction coef nodes are read by the Maestro Control node to publish the corresponding commands to Pololu Maestro which is connected to our ESC and Servo.

In the subsequent sections, we will delve deep into the details. The code is implemented using ROS Noetic and Python3.6^{+} on Ubuntu 20.04 LTS.

A. Maestro Control

We created a ROS node that sets the motor speeds and servo angles, which subscribes to topics publishing real-time speeds and angles. Our ROS node integrates FRC4564/Maestro [12], a Python library that supports Pololu Maestro through USB serial. Before publishing speeds through the state machine, we send neutral signals to the motor and servo for three seconds, to calibrate our ESC. After calibration, our default 0 servo angle is set at -0.23 to correct a slanted back wheel. Our ESC sends forward signals to the motor when the speed is positive, and brakes when the speed is 0 or negative.

B. Sensors Integration

1) RGB-D: We use the realsense-ros [13] library which launches multiple nodes that publish relevant data from our Intel RealSense camera including raw image data and depth data.

2) IMU: We launch the spatial Phidget driver which is an open-source ros node for Phidgets [14] driver. Phidget22 driver reads data from the IMU Phidget and the IMU node publishes the 3-axis linear acceleration data in real-time.

C. Depth Analyzer

In order to convert the depth data published by the camera into meaningful data we created a Depth Analyzer node. Depth Analyzer reads depth data and generates the distance to the wall on the left, the distance to the center, and the distance to the wall on the right.

To calculate the distance to the walls on both sides of the vehicle, we use the OpenCV [15] library. Using OpenCV, we create a mask to detect all the depth values that are less than a certain threshold to make sure we are detecting the side wall closest to the vehicle, and then we find the largest contours. As seen in Figure 7, we are able to detect the walls on both sides of the vehicle. These depth values in these contours are then used to calculate the average distance from the vehicle to the wall on the left and the wall on the right.

In order to calculate the center depth values, we use the average of the depth values in the center of the depth image.

All the published distances are then used later by our planners to generate appropriate motion planning commands.

D. Stop Sign and Ball Detection

To detect the stop sign we used an open-source Haar cascade stop sign classifier [16] and used the raw RGB images

Fig. 5: RGB image of a ball

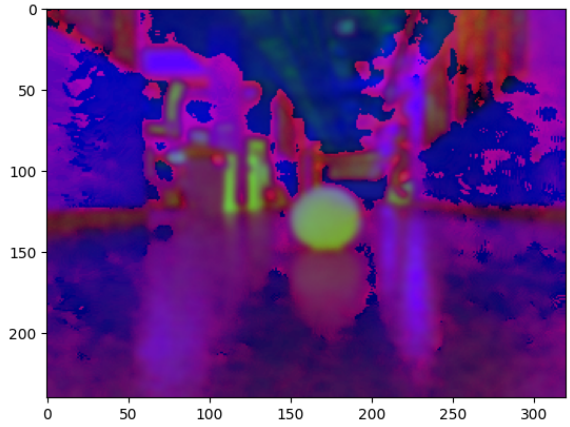


Fig. 6: Detecting a purple ball from raw RGB images

from the Depth Camera node as the input. For ball detection, we took inspiration from an open-source Python gist [17] which detects green balls. Our approach involves finding the corresponding HSV values for our specific colored ball and then performing a color mask using OpenCV. Figures 5 and 6 show how our code detects a purple ball.

In order to avoid detecting the stop sign and ball too early, we make sure the detected object is bigger than a certain threshold.

E. Auto Drive

The auto drive node reads data from the depth analyzer node and publishes the corresponding Maestro Polulu commands which set the car steering angle and motor speed. In order to send move commands efficiently, we defined three motion primitives. Our motion primitives are as follows:

- Straight: Makes sure the car keeps going straight.
- Turn Right: Makes sure the steering angle changes the car direction elaborately to the right
- Stop: Sends a zero speed and a zero steering angle command

1) Straight: To ensure that the car travels straight, we use the simple-PID [18] package. We use the distance to the left wall and the distance to the right wall and use the PID controller to try to keep these two values close to each other, so the error for the PID is quantified as

$$\varepsilon = \delta_{right} - \delta_{left},$$

where, ε denotes the error term, and δ_{right}, δ_{left} are the distances from the right and left walls, respectively.

The PID controller will try to minimize the error only if the error is more than a certain threshold. We then use the updated PID value to adjust our steering angle. The values we are using for our PID are $p = 0.25$, $i = 0$, and $d = 0$. These values were tuned through multiple trials. In order to travel at a safe speed, we define a fast and slow speed. The fast speed will be performed when the distance from the car to the center is more than a certain threshold, otherwise, slow speed will be performed.

To account for increased left and right depth values which should be ignored, such as a reflective window or an open door that shouldn't be entered, we decrease the error value by a factor if the sum the the left and right depths are above a certain threshold.

2) Turn Right: The turn-right command will be triggered when the distance from the car to the center is less than a certain threshold. Then it will send a constant turn angle and constant turn velocity.

3) Stop: The stop will be triggered when we stop the ROS node. It will set the car speed to zero and the car steering angle to zero degrees.

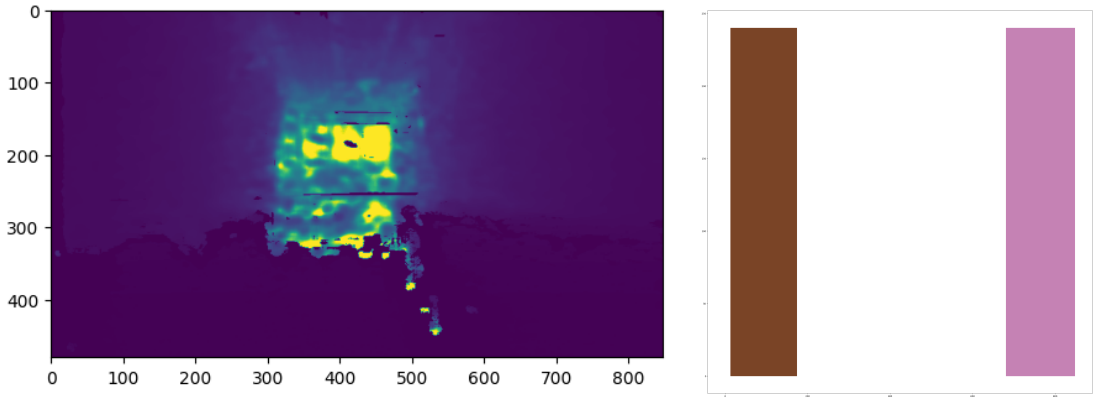
A determine state function is written which outputs which motion primitive should be executed. The straight motion primitive is executed as long as the distance to the center is larger than a certain threshold, if it's not, a turn-right motion will be initiated. The stop command will be sent if we kill the ROS node.

F. Avoid Ball and Stop at Stop Sign

The avoid ball and stop at stop sign nodes both use the same motion primitives as the auto drive node.

1) Avoid Ball Strategy: In order to avoid the ball, when the ball is detected, we enter the ball-avoidance state where we send signals for a sharp right turn for a small number of timesteps, after which we enter the ball-dodging state where it corrects towards the left for a small number of timesteps. The right turn is sharp and at a high speed to account for the velocity of the ball moving toward the vehicle. The left correction is necessary to avoid crashing into the wall, as the PID controller does not adjust swiftly enough to steer the car out of a sharp right turn. The left correction is performed at a low speed to avoid steering back into the trajectory of the ball.

2) Stop at Stop Sign: When the car detects a stop sign, it will send a stop command. The stop command will be sent for 3 seconds, at which the speed for the motor is set to zero. After 3 seconds, the car resumes forward movement.



(a) Depth image (b) Contours

Fig. 7: Side walls detected from depth image

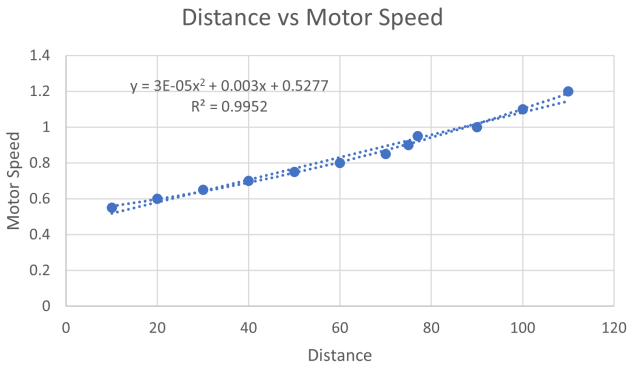


Fig. 8: Calculating speed for a given distance

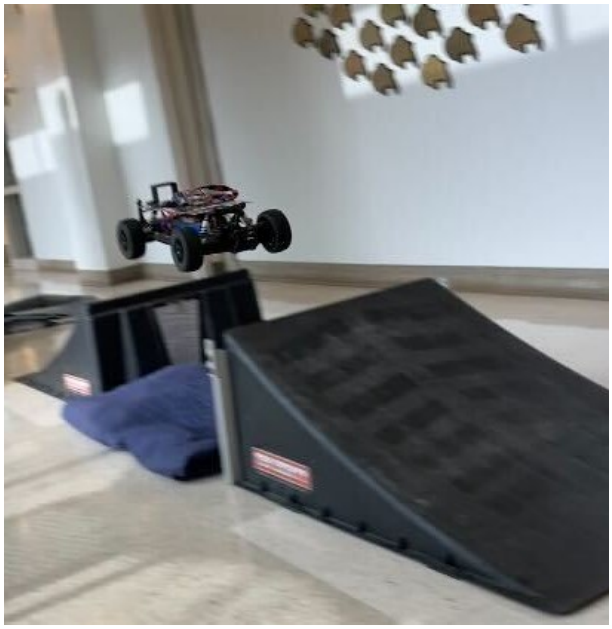


Fig. 9: Successful jump for 1-meter distance

G. Jump

In order to complete the jump, we tested different motor speeds for various distances between the ramps. We lined up the car with the ramp and attempted jump distances ranging from 0.1m to 1.1m with increments of 10cm. We recorded the motor speed required to make each distance and determined a function to calculate the speed based on the desired jump distance.

The following equation was used to calculate the desired jump velocity.

$$\nu_{des} = 3 \times 10^{-5}(\delta_{jump})^2 + \frac{3}{1000}(\delta_{jump}) + 0.5277, \quad (1)$$

where, ν_{des} denotes the desired velocity and δ_{jump} is the jump distance.

H. Sparse Map

Hector mapping[19] is a probabilistic SLAM algorithm that creates a map of an environment with also tracking the robot's position and orientation with it. Hector's mapping algorithm is able to provide an accurate and robust mapping of even unknown environments. Some of the key advantages of hector mapping are its real-time performance and ability to handle environments with large portions of the map occluded. It detects the mapping features such as walls, corners, and objects. It is capable of generating a map even without the IMU data, just by estimating the poses using laser scans.

The Hector mapping generally uses a combination of laser scan data and IMU readings to create the environment map. The laser scans measure the distance between the robot and the objects in the environment, while IMU measures the robot's acceleration and rotation. By fusing data from both sensors, the algorithm estimates the robot's pose in 3D space.

For generating the sparse map, we used the Realsense D435 depth images and camera metadata. The depth images are converted to laser scans and then fed as input to the hectormap node which generates a 2D mapping of the EC Hallways (Downstairs). One of the limitations of the hector mapping is it is unable to perform loop-closures.

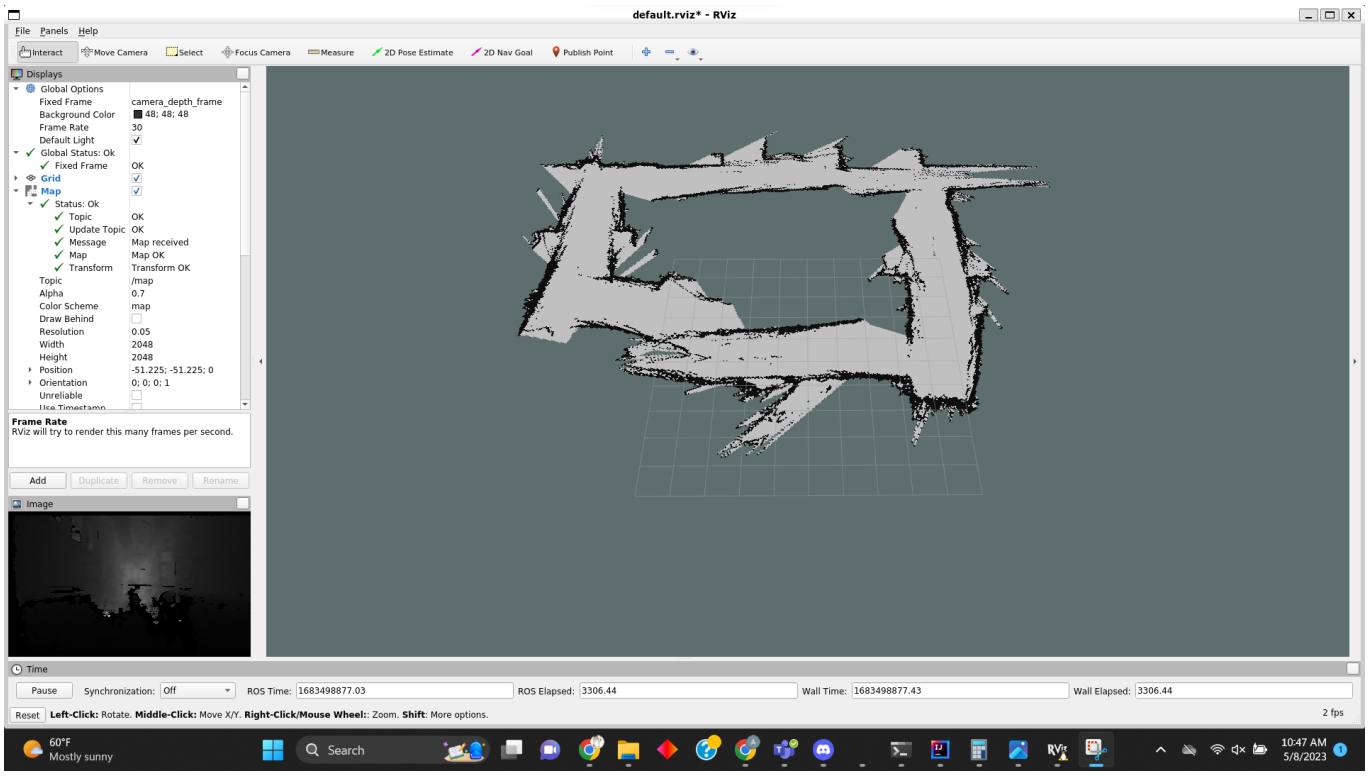


Fig. 10: Race map

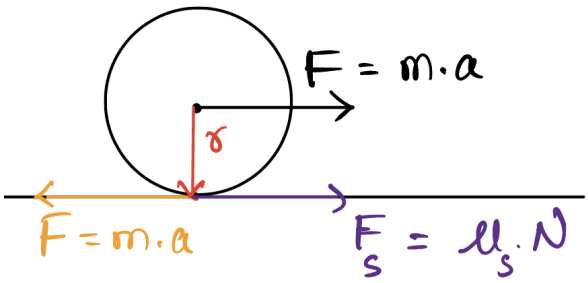


Fig. 11: Free-body diagram explaining forces on a tire

Nevertheless, despite only using the depth images and having our base frame and odom frame set to camera depthframe, we generate a reasonable structure of the hallways.

I. Friction Coefficient

There are 3 types of friction, rolling, sliding, and static, and with all these types of friction, there also exist 3 types of coefficients of friction. In our experiment, we are calculating the static friction. A wheel or ball on a surface is held there by static sliding friction. In order to start the rolling motion, a force or torque must be applied to the wheel. The force of the static sliding friction prevents the wheel from sliding and thus initiates the rolling motion. Say, our car has an acceleration 'a' and our wheels don't slip between the point of contact of the tire and the surface (road), then we can represent the forces acting on the surface in Figure 12.

$$F = F_r$$

$$m * a = \mu_s * N$$

The frictional force is represented by F_s which is equal to the force applied by the tire on the ground when accelerating by 'a', but as the weight is distributed between the four wheels the equation can be written as:

$$\frac{m * a}{4} = \frac{m * g * \mu_s}{4}$$

The friction node reads the acceleration values published from the IMU sensor to compute the coefficient of static friction. In order to compute the coefficient a straight command is sent so the car starts going straight and can collect multiple acceleration data. At each time step, the below formula is used to calculate the friction coefficient [20]:

$$\mu = \frac{a_r}{g}.$$

This formula is derived from these formulas:

$$F_r = \mu N,$$

where, F_r is the force required to overcome friction and N is the total normal force acting on a car. which is mg. We also know that

$$F_r = ma_r,$$

hence if we put the two formulas together, we get the coefficient formula as mentioned above. During testing we estimated the coefficient to be 0.57 for the EC basement floor.

V. EXPERIMENTATION

We set up various utilities for the testing process of each challenge. We had two sets of bash scripts: One set of launching bash scripts that launch all required rosnodes for a certain operation or challenge, such as a race launching script and a ball avoidance launching script; another set of bash scripts that each launch an individual node. This setup allowed us to launch operations conveniently when we needed to, and also launch nodes separately to analyze the output values of various nodes in real-time.
Besides the rosnodes described in the methods section for autonomous driving, we have two rosnodes, test_servo, and test_motor, for sending manual angle and speed updates to the odroid in real-time. These nodes were used in testing to tune angles and speeds for the challenges, especially the jumping challenge.

The vehicle was rigorously tested in the basement corridors of the engineering center.

The initial tests were aimed at verifying the platform driving in a straight line. We started by giving fixed torque inputs to the chassis motor in order to control the vehicle speed. The next step was making the car stop after a certain distance/time traveled. Optimizing the motor inputs for this simple maneuver took a lot of time. Primarily because we had trouble with the ESC as it was unreliable and resulted in either no inputs reaching the motor or a considerable delay before inputs were realizable. This was resolved by replacing the ESC.

Next was the phase of sensor integration to ensure autonomous navigation. This was primarily achieved using the RGB-D sensor being mounted at the front of the vehicle for obstacle detection. The idea was to never allow the vehicle to drive too close to the obstacles. The integration was functional for straight-line driving and our car was able to stop whenever it got close to the walls.

The ability to turn corners was by far the most challenging phase of the testing regimen. After we tuned the values to make sure the car stays in a straight line, we tried different distances to the center values to initiate right turns. One of the issues that we fixed was that the car was able to successfully perform a right turn, however, even though it would enter a straight motion command, the PID controller was not able to correct the steering angle fast enough and the car would crash into the wall. In order to fix this, we decided to take a sharp left turn after performing our right turn, so our PID controller can safely adjust the steering angle. Once the vehicle was successfully doing laps we pivoted towards attempting other challenges.

Since our code architecture was modular, it proved to be much easier to use the existing code to perform all of the challenges. The only issue we faced was in the ball detection challenge. Since we were using a purple ball, our code was specified to detect balls that have a color ranging between light purple and dark purple. During our experiments, the detection algorithm would sometimes detect other objects that had dark blue colors which would disrupt the planning decision. In order to fix this issue, we limited the radius of the detected object and it turned out to greatly improve the performance.

VI. DISCUSSION

Working with various hardware and sensor inputs poses a lot of challenges, in this section, we will go over the challenges we faced and how we overcame them.

A. Versioning

Initially, our eMMc had Ubuntu 18.04 with kernel 4.15 installed on it. We faced a lot of challenges working with these specific versions. For starters, our WIFI dongle did not work with that specific kernel version. We also faced challenges with installing the Intel Realsense version. Since the WIFI dongle was not working, we had to use a cable to access the internet on the Odroid. After trying various methods, we resorted to flashing the eMMC and installing Ubuntu 20.04 with kernel 5.4. In order to flash the eMMC we used the Etcher software [21]. After updating our Ubuntu and kernel versions, we were able to effortlessly use the WIFI dongle and install various drivers.

B. Servo control and ESC

At some point when working with the ESC, our ESC stopped receiving commands. We realized that Pololu maestro channels were closing unexpectedly and we were required to switch channels or restart the Pololu connection for the ESC to receive signals and power the motor again. We did not have this issue previously and through exhaustive tests, we found that the calibration command has changed. We resolved this issue by sending neutral signals to the ESC for three seconds every time before starting to send move commands to the ESC.

Our ESC has two configurations: Forward-Break and Forward-Break-Backward, however, our motor is unable to run in reverse when we have the Forward-Break-Backward configuration.

C. Calculating distances from the walls

Working with a real-sense depth camera posed various challenges. Initially, to calculate the distance from the car to the wall on the left and right, we calculated the mean of the depth values on the left and right sides of the depth image. In our experiments, we realized that these values are very unreliable and do not represent the true value of the left wall and the right wall. As a result, our car was not able to keep going straight and would crash into the wall. This happened because our PID controller used the distance to the left wall and the distance to the right wall, to fix its steering angle, and these values were way off. In order to fix this problem, we decided to use the contouring method which was mentioned in the method section, Depth Analyzer. Using the contour method helped greatly, because instead of taking the mean of all the values on the left or on the right, we would first cluster the values that are close to each other on the left and right side of the image, and after that, we

would calculate the average. So a lot of noisy data won't be considered in the calculation and the distance to the left and the distance to the right wall will be a lot more accurate.

D. Turning

Turning was a big challenge throughout this project. We implemented and tested many strategies for turning, however, it is hard to account for the different spatial configurations of different turns and avoid making false turns.

Our initial approach to turning was based solely on the center depth value. The predominant issue we faced is that the vehicle would not be able to exit the turn state due to not detecting a high center value for a long enough interval to successfully exit the turning state. Other issues we faced were making turns when the vehicle was angled towards a sidewall and perceiving it as an approaching front wall.

To account for these cases, we changed to initializing the turn based on the center, left, and right depth values, which reduced the turn time and reduced false turns but were not able to avoid them completely.

We also experimented with setting the turn state for a set time interval and correcting towards the other direction after turning.

E. IR sensor

We were interested in using the IR sensor to augment our RGB-D one, especially for near obstacle detection accuracy. In order to use the IR sensor, it needs to receive a 5V input. However, Odroid only supplies 1.8V. So, we decided to connect the IR sensor to one of the channels on Maestro and configure that channel to be an input channel. We were able to read IR sensor values from Maestro Control Center, however, the values turned out to be unreliable. Particularly, the data received had a high uncertainty and it was difficult to ascertain obstacle distance. We thought about designing a custom filter to make it work. But, the connectivity was very intermittent. Hence, we ended up not using it at all.

F. Hardware

Although the hardware setup seemed quite straightforward for this project, there were a few challenges. Initially, we opted to use 2 NiMH batteries, to power the Odroid and the drive subsystem separately. However, we noticed a very high discharge rate of the battery due to the Odroid's high current requirements. Then we switched to a 3S LiPo battery for the Odroid and continued to use the NiMH for the drive.

We also faced issues with the electronic speed controller. At first, the servo controller rails were powered by another battery. However, this caused damage to the ESC. We then realized that the ESC back-powers the servo controller and this extra voltage might have caused the ESC to malfunction. Although the ESC is technically able to operate with a 2/3S LiPo battery as well, we opted to use an NiMH as it seemed to be charging quicker and was more stable for the purpose of the drive subsystem.

Throughout the project, due to electronic complications, we switched two ESCs and one servo motor. As we were unable to use the full range of the servo while testing on the track, we decided to test the servo motor with an Arduino and observed that the motion of the shaft was very stiff and restricted. We attributed that to a broken gear and tried to replace the motor.

However, the servo motor was permanently fixed to the base forcing us to switch to an entirely different chassis. This switch had to be made only three days before the final.

VII. CONCLUSION

The development of an autonomous vehicle proved to be a difficult problem with unique challenges. The hardware and software integration was a major hurdle due to the uncertain nature of sensors and the varying responsiveness of the motors. We were successful in using the RGB-D and IMU sensors by writing multiple ROS nodes for real-time sensor integration and vehicle control.

REFERENCES

[1] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 458–488, 2022.

[2] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," *Proceedings of Machine Learning Research*, vol. 123, 2020.

[3] L. Chen, Y. Li, C. Huang, B. Li, Y. Xing, D. Tian, L. Li, Z. Hu, X. Na, Z. Li, *et al.*, "Milestones in autonomous driving and intelligent vehicles: Survey of surveys," *IEEE Transactions on Intelligent Vehicles*, 2022.

[4] A. Gotlib, K. Łukojć, and M. Szczygielski, "Localization-based software architecture for 1: 10 scale autonomous car," in *2019 International Interdisciplinary PhD Workshop (IIPhDW)*, pp. 7–11, IEEE, 2019.

[5] T. Weiss and M. Behl, "Deepracing: A framework for autonomous racing," in *2020 Design, automation & test in Europe conference & exhibition (DATE)*, pp. 1163–1168, IEEE, 2020.

[6] G. Velasco-Hernandez, J. Barry, J. Walsh, *et al.*, "Autonomous driving architectures, perception and data fusion: A review," in *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 315–321, IEEE, 2020.

[7] Y. You, Y. Wang, W.-L. Chao, D. Garg, G. Pleiss, B. Hariharan, M. Campbell, and K. Q. Weinberger, "Pseudo-lidar++: Accurate depth for 3d object detection in autonomous driving," *arXiv preprint arXiv:1906.06310*, 2019.

[8] W. Shi, M. B. Alawieh, X. Li, and H. Yu, "Algorithm and hardware implementation for visual perception system in autonomous vehicle: A survey," *Integration*, vol. 59, pp. 148–156, 2017.

[9] S.-H. Bae, S.-H. Joo, J.-W. Pyo, J.-S. Yoon, K. Lee, and T.-Y. Kuc, "Finite state machine based vehicle system for autonomous driving in urban environments," in *2020 20th International Conference on Control, Automation and Systems (ICCAS)*, pp. 1181–1186, IEEE, 2020.

[10] X. Zhang, W. Zhang, Y. Zhao, H. Wang, F. Lin, and Y. Cai, "Personalized motion planning and tracking control for autonomous vehicles obstacle avoidance," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 5, pp. 4733–4747, 2022.

[11] K. J. Åström and T. Hägglund, "The future of pid control," *Control engineering practice*, vol. 9, no. 11, pp. 1163–1175, 2001.

[12] "Frc4564 / maestro." `https://github.com/FRC4564/Maestro`.

[13] "Ros wrapper for intel® realsense™ devices." `https://github.com/IntelRealSense/realsense-ros/tree/ros1-legacy`.

[14] "Phidgets spatial ros driver." `https://github.com/ros-drivers/phidgets_drivers/blob/noetic/phidgets_spatial/README.md`.

[15] "Opencv: Open source computer vision library." `https://github.com/opencv/opencv`.

[16] "Stop sign detection." https://github.com/maurehur/Stop-Sign-detection_OpenCV.
[17] "Ball tracking with opencv." https://pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/.
[18] "A simple and easy to use pid controller in python." https://github.com/m-lundberg/simple-pid.
[19] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *2011 IEEE international symposium on safety, security, and rescue robotics*, pp. 155–160, IEEE, 2011.
[20] A. Balter, "How to calculate the coefficient of friction." https://sciencing.com/calculate-coefficient-friction-5200551.html, 2023.
[21] "Flash. flawless.." https://www.balena.io/etcher.

APPENDIX1:HUMAN-ROBOT INTERACTION WITH AUTONOMOUS VEHICLES

Human-robot interaction (HRI) is a critical area of research in the development of autonomous vehicles. As the technology behind autonomous vehicles continues to advance, it is increasingly important to ensure that humans can interact with these vehicles safely and effectively.

In this report, we will begin by defining Human-Robot Interaction and then explore its crucial role in the context of autonomous driving cars.

A. Human Robot Interaction

HRI is an interdisciplinary field at the intersection of artificial intelligence and computer science, as well as cognitive science, psychology, and social science [22]. HRI involves interaction between humans and robots in a shared space. Some examples of these robots include:

- Industrial Robots: Amazon Robotics uses multiple warehouse robots, some of which are still prototypes but they all show promising potential [23]. Some of these robots include pinch-grasping robots that perform quickly moving an item without damaging it. One other robot is Sparrow, which uses computer vision to detect and select from multiple products.
- Social Robotics: The feature that separates social robotics from other robots, is that their design is focused on encouraging human interaction with them. They can come in different shapes and they usually have approachable and affable statics [24]. Social robots can be used to help children build social skills, they can also be used to help with education and provide a personalized learning experience.

B. Potential of Autonomous Vehicles

Autonomous vehicles have immense potential for revolutionizing society in general and the field of transportation in particular. There are several aspects of HRI that can be enhanced by autonomous vehicles. Primarily the vehicles, if designed well, promise adherence to traffic laws and ensure precision driving. This will not only affect the behavior of human drivers positively, but also the passengers of these vehicles will have a more leisurely commute.

Another aspect is accessibility. The concept of driverless vehicles will provide transportation services to individuals who are unable to drive. They may include the differently-abled and the elderly.

A promising direction is the use of autonomous vehicles to improve social interaction. The vehicles me be equipped with external interfaces and displays to engage with people; both other drivers and pedestrians. For example, an autonomous ice cream truck may have interfaces designed to give children easy transactions and an enjoyable experience while purchasing treats. The displays may also be employed for advertising, conveying public service announcements, or streaming events. The possibilities are endless.

1) Safety: Ensuring the safety of human drivers, passengers, and other road users is a critical challenge in the development of autonomous vehicles. HRI research must address issues related to safety, such as how to avoid collisions, how to handle unexpected situations, and how to ensure the vehicle can be safely operated in all conditions. [25]

2) Trust issue: One of the primary concerns in HRI with autonomous vehicles is the issue of trust. Studies have shown that humans tend to be more hesitant to trust autonomous vehicles than they are to trust human drivers. This lack of trust can lead to issues such as over-reliance on manual control, which can reduce the effectiveness of autonomous systems.

3) Effective communication: Another key issue in HRI with autonomous vehicles is the need for effective communication between humans and the vehicle. In order to ensure that humans can understand the behavior and intentions of the autonomous vehicle, it is important to develop effective communication systems that can provide feedback to the human driver.

4) Privacy and Security: As autonomous vehicles collect and transmit data, it is important to address issues related to privacy and security. HRI research must consider how to protect personal data and how to ensure that the vehicle's systems are secure against hacking and other forms of cyber attack.

5) Ethical Issues: HRI research must also consider ethical issues related to autonomous vehicles, such as who is responsible for accidents involving autonomous vehicles, how to ensure the equitable distribution of benefits and risks, and how to ensure that autonomous vehicles are used in an ethical and responsible manner.

C. Prediction Intent

An autonomous-driving car should be able to process all of its sensory inputs into meaningful data that can be used to drive safely. Autonomous driving cars are equipped with various sensors including Cameras, Radars, Lidars, Ultrasonic sensors, GNSS, angle and torque sensors and etc. [26] They should be able to process all the data published from these sensors in a timely manner so they will be able to make decisions such as not colliding with moving obstacles, such as other cars or pedestrians, or when to stop at a stop sign and etc.

A very challenging topic in prediction is predicting human intent. Because humans are very unpredictable, and if the car doesn't react safely in a timely manner, disaster could happen. The current methods for predicting human actions

in autonomous vehicles rely heavily on information gathered from sensors. However, there are certain factors that can shape individual behavior, which cannot be detected by sensors or by observing human gaze or arm movements. While these methods may work for predicting the actions of an industrial robot worker or the movements of children at home to avoid collision with a service robot, they are not sufficient for self-driving cars. Often, a person's actions may not even be clear to them, making it difficult for autonomous vehicles to predict their next move. Even if a pedestrian is visible to the car's camera and walking at a particular speed, humans do not always follow traffic rules in the same way.[27]

Autonomous cars should also be able to comprehend data the same way humans do. For example, the way that we avoid obstacles on the road is different than how we avoid humans on the road. If we see a human on the road, we usually wait for them to finish passing the street, however, when we pass by a loose object on a freeway, we don't stop and instead, we try to go over it or change lanes.

Teaching autonomous cars to process data the same way as humans do, is an active area of research and is one of the important factors in making autonomous driving cars more safe and robust.

D. Driving Style Prediction

Predicting driving style can help autonomous cars in several ways. By analyzing data on how human drivers typically navigate the roads, an autonomous car can make more informed decisions and adjust its driving style accordingly. For example, if the car detects that a driver typically accelerates quickly from stop signs, it may be able to anticipate this behavior and respond accordingly. This can help improve the car's overall efficiency and responsiveness, which can be particularly useful in situations where the car needs to make split-second decisions to avoid accidents. Additionally, by predicting driving style, autonomous cars may be better equipped to communicate with human drivers on the road. For example, if the car detects that a driver is hesitant or nervous, it may be able to adjust its driving style to help the driver feel more comfortable. This could involve things like adjusting the car's speed or giving the driver more space on the road.

In this paper [28], they present a deep neural network architecture to learn driving style from trajectory data. Their model is inspired by convolutional recurrent neural networks, and they aim to enhance the model by identifying semantic patterns in trajectories and encoding the dependencies between these patterns. They discuss that deep learning models can be very useful in determining driving style however, geolocation bias can pose a challenge when designing frameworks based on data with spatial information, and the use of such data may not always be unbiased. For some people starting to drive in a different city or another country may pose a challenge. Different traffic rules can take time to adjust, for example, in Britain, the traffic law is to drive on the right side of the road whereas in the USA, we should drive on the left side of the road. Different behavioral expectations also play a big role. Ozkan et. al. [29] performed a comparison of driving in six different countries. They found that in some countries is more probable to break traffic laws and drive more aggressively.

E. Integrating human-style driving in autonomous vehicles

For autonomous vehicles to become widely adopted, users must feel comfortable with them. By replicating human-style driving, autonomous vehicles may feel more familiar and intuitive to users, increasing their acceptance and adoption. If humans don't feel comfortable with the way their car behaves, they would switch to manual control and that would defeat the purpose of autonomous car production. Driving styles have been shown to make a big impact on the acceptance and adoption of autonomous vehicles. One study presents an approach that encodes human driving styles [30], such as aggressive, neutral and defensive driving styles in autonomous driving systems. They use a penalty structure and evaluate against a set of signal temporal logic formula. One aspect of human-style thinking is known as common sense, which is an extremely efficient logical inference made by humans. Many driving decisions humans make can be contributed to common sense. On the other hand, autonomous driving technology primarily relies on control frameworks such as state machines, kalman filters, and machine learning techniques for decision making, which are geared towards optimization. While these techniques can have high success rates, they often produce sequences of actions that are unnecessarily complex, or spend too much time coming to conclusions that would have taken the average human driver almost no thought. the AUTO-DISCERN system [31] automates commonsense reasoning using answer set programming and a goal directed system. By doing this, they develop an autonomous driving system that simulates how a human driver thinks and makes decisions.

The importance of integrating human-style driving in autonomous vehicles also ties into the larger phenomenon of the cultural interruption caused by advanced technologies. Advanced technologies such as drones, autonomous cars and artificial intelligence are typically designed with a focus on optimization, without considering human's cognitive habits or cultural norms. As these technologies start to be widely adopted, there is a disconnection between technology driven behaviors and previous socio-material practices [30]. Integrating human-style driving in autonomous driving allows more cognitive comfort which encourages human understanding of technology. Failing to consider human-friendly driving practices can instill technophobia and lead to distrust in technology.

VIII. Conclusion

In conclusion, Human-Robot Interaction is a critical aspect in the development of autonomous vehicles. The ability to predict and interpret human behavior is a key challenge in developing effective autonomous driving systems. HRI research is focused on developing methods that can interpret,

predict, and mimic human behavior to make self-driving cars safer and more efficient.

REFERENCES

[22] K. Dautenhahn, "Methodology & themes of human-robot interaction: A growing research field," *International Journal of Advanced Robotic Systems*, vol. 4, no. 1, p. 15, 2007.

[23] "6 warehouse robotics innovations amazon showcased in 2022." https://www.supplychaindive.com/news/warehouse-robotics-automation-innovations-amazon-showcased-2022/638115/,.

[24] "What is a social robot?." https://builtin.com/robotics/social-robot,.

[25] "Human-robot interaction (hri) – current challenges?." https://roboticsbiz.com/human-robot-interaction-hri-current-challenges/,.

[26] "Autonomous cars 101: What sensors are used in autonomous vehicles?." https://levelfivesupplies.com/sensors-used-in-autonomous-vehicles//,.

[27] A. A. Mokhtarzadeh and Z. J. Yangqing, "Human-robot interaction and self-driving cars safety integration of dispositif networks," in *2018 IEEE International Conference on Intelligence and Safety for Robotics (ISR)*, pp. 494–499, 2018.

[28] S. Moosavi, P. D. Mahajan, S. Parthasarathy, C. Saunders-Chukwu, and R. Ramnath, "Driving style representation in convolutional recurrent neural network model of driver identification," *CoRR*, vol. abs/2102.05843, 2021.

[29] T. Özkan, T. Lajunen, J. Chliaoutakis, D. Parker, and H. Summala, "Cross-cultural differences in driving behaviours: A comparison of six countries," *Transportation Research Part F-traffic Psychology and Behaviour - TRANSP RES PT F-TRAFFIC PSYCH*, vol. 9, pp. 227–242, 05 2006.

[30] J. Karlsson, S. van Waveren, C. Pek, I. Torre, I. Leite, and J. Tumova, "Encoding human driving styles in motion planning for autonomous vehicles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1050–1056, IEEE, 2021.

[31] S. Kothawade, V. Khandelwal, K. Basu, H. Wang, and G. Gupta, "Auto-discern: autonomous driving using common sense reasoning," *arXiv preprint arXiv:2110.13606*, 2021.

APPENDIX 2: DEEP LEARNING APPROACHES

The development of GPU parallel processing has had a significant impact on machine learning research. Initially limited by the computational capabilities of the test system, machine learning algorithms are now able to compute trillion parameters in reasonable amounts of time to efficiently understand and solve the problems. With the ability to perform computations in parallel, deep learning techniques have been able to make significant strides in their ability to learn and understand complex challenges. This has led to a rapid expansion of research in many fields, as deep learning approaches have become more powerful and capable of solving a wide range of problems across multiple domains.

Some of the major challenges in autonomous driving are understanding and predicting the environment, such as the other vehicles, pedestrians etc. In this section, I will briefly discuss how deep learning has impacted different modules of autonomous driving cars. Most of autonomous cars have five modules[32] for the autonomous capabilities to behave similarly to humans in driving:

A. Perception:

Similar to the human visual cognition system, the perception module is responsible for being able to sense and be aware of the environment such as detecting stop signs, passengers, or lanes. This module is the basic block that influences most of the other modules. So, achieving high accuracy in the perception challenges becomes crucial for autonomous systems.

Prior to deep learning, the traditional approaches have been used to hand-craft different sets of features that are able to highlight interest regions in the sensor data and use machine learning approaches at the end of the pipeline. These features require lots of tuning to solve a particular task like detecting a ball or a stop sign. Also, these features can't be generalized in different scenarios such as changes in lightning conditions can heavily impact these algorithms. Deep learning approaches have achieved state-of-the-art performance in these tasks.

Convolutional Neural Networks(CNN)[33] are a class of neural networks that have been proven to outperform traditional algorithms for the computer vision tasks like object detection, object classification, and semantic segmentation. CNNs are specifically designed to capture useful information from visual data, such as RGB images. CNNs introduced the concept of convolutions to neural networks, applying a set of learnable kernels sliding across each block of the image, to generate feature maps. Traditional neural networks prior to CNN are fully connected, where each neuron in one layer has a connection to each neuron in the next layer. This requires a lot of computational power and also doesn't capture local information in a region in the image. On the other hand, CNNs are able to effectively capture the spatial information by the convolutions while only requiring a single kernel(of max size 11x11). By increasing the number of kernels, it can capture a larger variety of spatial representations.

Typically, the output from CNNs is fed to a non-linear activation function (like ReLU), to introduce non-linearity to capture more complex features and patterns in the data. The output from the activation function is fed to the Pooling layer (typically Maxpool layer), to reduce the spatial dimension while preserving the most important features. CNNs are translational invariant as the same kernel is used to capture a feature vector. The process of convolution, non-linear activation and pooling are stacked one after the other, where the starting layers capture low-level spatial features like edges, and lines while the layers at the end capture much more complex representations, such as digits or objects.

Region-based CNN (RCNN) architectures like Fast RCNN[34] have achieved a greater improvement in accuracy for object detection. Fast RCNN proposed a Region Proposal Network(RPN) that shares convolutional features with another CNN classification network, which finally detects the objects. As CNNs are only designed for extracting features from an image, they fail to capture the temporal features in a sequence of images. Although most advanced architectures like Transformers outperform compared to CNN architectures, CNN layers still remain a backbone to these architectures.

B. Prediction:

The prediction module has the ability to anticipate future events based on the current perception of the environment. It

often involves predicting the behavior of the other vehicles, during lane detection, or the pedestrians crossing the Z-walking. As mentioned above in the Perception section, CNN architecture considers only 1 frame to make a decision such as detecting the object. Most of these tasks require a sequence of inputs to predict the next action. So, traditional CNN architectures fail to capture this capability.

Long Short-Term Memory(LSTM) networks are used in processing sequential data, as they have the capability to selectively remember or forget the previous states based on the current state. An LSTM cell has 3 gates: input, output, and forget. The input gate determines which features of the input should be stored in the cell, and the output gate decides which parts of the cell state should be considered for generating the output of the cell. The forget gate decides on which parts of the cell state should be discarded, based on the current state, which allows its unique capability to select the essential information represented in the temporal sequence. To have non-linearity, these gates have sigmoid or tanh (-1 to +1) non-linear activation functions. This capability off storing the previous inputs and discarding them when they are not relevant, adds the memory component to the LSTM cell enabling them to estimate future events. LSTMs can be stacked to create architecture, which allows for extracting the higher-level features from the input sequence and can lead to better performance.

Once the objects like other vehicles and pedestrians are detected in the perception module, LSTM can interpret the long-term dependencies effectively to estimate the complex dynamics of vehicle and pedestrians' motion, including their speed and direction. These estimations can be used to interpret whether a vehicle is about to take a right turn or slow down or a pedestrian crossing the road and use this information to effectively predict and plan future trajectories.

LSTMs usually take high-level feature representation from either a CNN or a fully connected layer and try to estimate the temporal information embedded in those features to make a decision. Even though LSTMs are effective at short-term dependencies, they fail to capture the long-term dependencies of how each input in related to another input in the sequence. Also, the LSTMs cells have multiple parameters that need to be trained, which increases the computational complexity of the model.

C. Localization and Mapping:

Localization is used to determine the position and orientation of the car in the environment, which helps in navigating and interacting with the environment. One major challenge in localization is to process large amounts of sensor data, such as camera images, lidar point clouds, IMU acceleration, and angular velocities etc. Localization tasks such as odometry estimation often require long-term dependencies of the sequences to estimate the current pose of the system. Transformers[35] can help address this challenge by understanding the long-range poses in a sequence and providing a more efficient and accurate representation in predicting future poses.

First introduced in NLP tasks like Machine translation, these architectures have been widely adopted in most of the other challenges, outperforming the existing state-of-the-art models. One of the major blocks in the attention module, allows the model to attend to different parts of the input sequence selectively. This allows each input to have a mapping of how it related to other inputs in the entire sequence. With calculating all the possible relations between each input in the sequence, the computations are expected to increase, but these relations between the inputs are just vector computations and can be computed in parallel. This allows the computational complexity to go very high, even while capturing all the dependencies in the sequence. These capabilities can attract the research community to explore Transformers, replacing the CNN and LSTM architectures.

[36] replaces LSTM used in DeepVO[37] architecture with transformers to estimate the long-term dependencies between the images. They used pre-trained feature representations from Flow-Net, which is better at geometric tasks and fused the features to the transformer encoder. The self-attention modules in these transformers are able to understand the relations between the images in the entire sequence to estimate the odometry and outperformed the estimating the state-of-the-art techniques in odometry estimation.

Mapping provides the system with a structured representation of the environment to plan efficient future paths and avoid obstacles. Some of the challenges in mapping arise due to gaps in the sensor data, which creates open spaces or drastic changes in rotation which makes it hard for the loop closures. PoinTr[38] is able to effectively reconstruct the point cloud from the sparse pointcloud, by introducing a new geometric-aware attention block, which can also capture the geometric relations of the points in its locality. This research is now being explored further to fill in the unexplored regions on the map, allowing to have more information about the surrounding in planning.

D. Planning:

Being aware of the surroundings and how they might change, the planning algorithms try to estimate the optimal high-level trajectory to reach the desired location. THey typically receive input from mapping and prediction. Traditional path planning algorithms like RRT, RRT* have been powerful and flexible to support search and planning in complex and dynamic environments.

Reinforcement Learning(RL) is a subfield of machine learning that involves training an agent to learn a policy that maximizes the reward. In the initial training iterations, the agent explores all the possibilities of navigation and tries to maximize only those that result in maximum reward over the iterations.

[39] proposes an end-to-end Deep Reinforcement framework for autonomous lane maneuvering. They used the RGB images and Lidar pointclouds to generate a bird-eye (top angle view) of the surroundings. For training the agent, they used a variant of Double Deep Q-Network(DDQN) with input state as bird-eye view, vehicle speed, and acceleration

along with the current lane and target lane. During training, the DDQN receives a reward signal that reflects how well it performed the task i.e reached the corrected lane, and the weights of the network are updated using backpropagation to maximize the reward signal.

E. Control:

The control module has two subsystems, mainly the motion planning and actuation subsystems. Motion planning subsystems generate a detailed motion plan and generate control commands with the high-level trajectory from the planning module, considering the vehicle dynamics, speed, acceleration, braking etc. The actuation subsystem executes these control commands and controls the vehicle actuators like throttle, steering etc.

Instead of mapping these control commands to actuators, the actual values of the actuators can also be learned using deep learning approaches. [40] uses an end-to-end framework to predict the steering angle and throttle from the sensor data. They use CNN with inputs as images and outputs as a vector with the control outputs. They tested their approach on a real car with an electronic control unit (ECU) and a steering actuator, which outperformed the traditional hand-crafted feature design algorithms.

F. Conclusion:

One of the main challenges with deep learning approaches is that they are highly dependent on the data they are trained for and can introduce additional bias into the system. And to actually deploy the deep learning approaches in real world scenarios, these need to be run in real-time. Overall, deep learning approaches have been better at solving the most of the challenges in autonomous driving applications. Further research in this direction can potentially expand the boundaries of the field, to achieve higher levels of autonomy.

REFERENCES

[32] R. Fan, J. Jiao, H. Ye, Y. Yu, I. Pitas, and M. Liu, "Key ingredients of self-driving cars," *arXiv preprint arXiv:1906.02939*, 2019.

[33] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[34] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[36] H. Zhao, X. Qiao, Y. Ma, and R. Tafazolli, "Transformer-based self-supervised monocular depth and visual odometry," *IEEE Sensors Journal*, 2022.

[37] S. Wang, R. Clark, H. Wen, and N. Trigoni, "Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 2043–2050, IEEE, 2017.

[38] X. Yu, Y. Rao, Z. Wang, Z. Liu, J. Lu, and J. Zhou, "Pointr: Diverse point cloud completion with geometry-aware transformers," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 12498–12507, 2021.

[39] P. Wang, C.-Y. Chan, and A. de La Fortelle, "A reinforcement learning based approach for automated lane change maneuvers," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1379–1384, IEEE, 2018.

[40] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2174–2182, 2017.

Fig. 12: Team MILK

ACKNOWLEDGMENT

We would like to thank Professors Hayes and Heckman for providing us this exciting opportunity of developing an autonomous racecar and persevering with our questions during the project. Also like to thank the TAs for being constantly available to answer any queries, especially regarding the vehicle hardware. Last but not least, we would like to thank the other teams in the course for opening sharing their insights and problems they faced, which helped us in our process. Hopefully we were able to do the same.