(CSCI 5302-001) Advance Robotics Final Project Report

Aidan Luczkow[1], Srikanth Popuri[2], Rashikha Jagula[3]

***Abstract*— In this Final Project report for CSCI 5302-001 Advance Robotics we highlight the approaches used to autonomously race our AWS DeepRacer around a race course, the challenges that we took up, how we completed them, and all of our results.**

I. INTRODUCTION

Autonomous vehicle technology has made significant advancements in recent years, with applications ranging from urban mobility to industrial automation. In this paper, we use combination of proportional (P) control for a more responsive and stable control to enable our vehicle with autonomous racing capabilities. We then implement LIDAR-based EKF for Localization. We attempt to incorporate Reverse Driving capabilities into our system and finally enable a controller such that it brings the vehicle to a complete stop when it sees a Stop Sign. This knowledge of implementation falls in line with the growing interest in autonomous systems, particularly in the realm of robotics, mastering precise localization and navigation in dynamic environments.

II. CHALLENGES

A. Part A: Successfully traversing the racing circuit

The objective of this challenge is to complete a time trial race around the course and successfully traverse the racing circuit without crashing into any obstacles.

B. Part B: Group A challenges chosen

1) 5 Point: LIDAR-based EKF for Localization: In this challenge we implement a LIDAR-based Extended Kalman Filter (EKF) for localization. The EKF should predict the vehicle's poses utilizing a vehicle kinematics model (Bicycle model) and a vehicle sensor model (Iterative Closest Point (ICP)).

2) 2 Point: Reverse Driving: In this challenge we aim to complete the course using the LIDAR sensor on the vehicle while driving backwards.

3) 1 Point: Adding Stop Signs: In this challenge, the vehicle is programmed to obey stop signs are added to the environment with a controller. The vehicle should come to a complete stop, pause for a moment, and then continue.

III. PART A: TRAVERSING THE CIRCUIT

A. Problem Description

The goal of the controller designed for traversing the racetrack is to be able to help the bot complete the track without running into walls and making successful right turns. The bot is designed to be right turning.

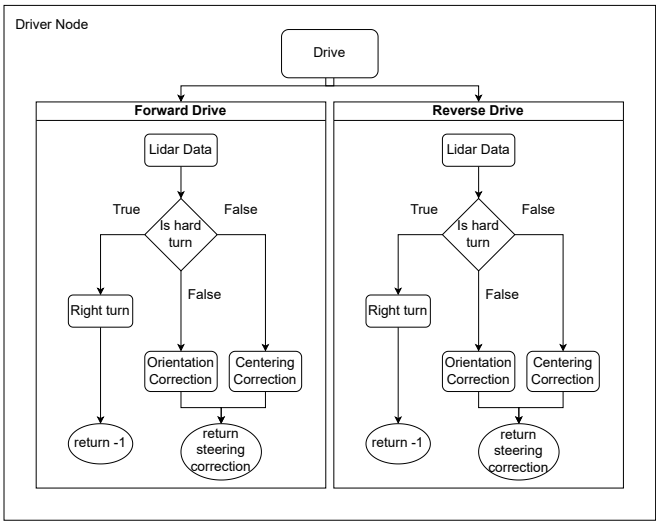


Fig. 1: Driver Node Code Flow

B. Implementation

We define the driver node in the drive_pkg such that, it has two modes. The first mode is Forward Drive and the second mode is Reverse Drive.

Forward Drive: This is the mode that we use to complete the track. The data from the LIDAR is used to perceive the surroundings and assess if we need to perform a hard turn or correcting its position to relative to wall

C. Feedback controller

The system has 3 states x- horizontal direction of bot, y- vertical direction of bot, θ orientation of bot but we are interested in only two states theta and x , we are basing of the x as the relative position between walls theta to be in parallel to system

D. Sensing Mechanism

1) sensing x position: LIDAR data is clipped for 5 meter which makes a circle of radius 5 meter, since the walls of the race track is 1.5 meter we consider only the LIDAR reading that are intercepted by objects that are less than 1.5 meter and relative x position is the average of all these readings that filters out the cut outs in the walls, this also makes the bot independent of the orientations The LIDAR readings are taken as shown in figure 2.

2) orientation: To get the orientation of the bot relative to walls, we apply the regression of the LIDAR data that are being intercepted by the walls , by this we get the slope of the walls with respect to the bot by making the slope 90 degrees we can get the orientation parallel to wall

Algorithm 1 Driver

```
procedure DRIVER(mode 0 or 1)
    if mode equals 1 then
        Forward Drive
    else if mode equals 0 then
        Reverse Drive
    return Initialized DRIVER object
function GET_CONTROLS(self, distance_matrix)
    if mode equals 1 then
        Forward Drive
    else if mode equals 0 then
        Reverse Drive
    if absolute value of a is greater than 0 then
        if mode equals 1 then
            Set angle = a (Forward Drive)
        else if mode equals 0 then
            Set angle = −a (Reverse Drive)
    else
        if mode equals 1 then
            Set angle = scaled error (Forward Drive)
        else if mode equals 0 then
            Set angle = − scaled error (Reverse Drive)
function SCAN_FOR_TURN(self, left_distances, right_distances)
    if hard turn detected then
        return -1.0
    elseNo turn detected
        return 0.0
function STEERING_NARROW(left_distances, right_distances)
    Checking the left and right distances.
    if points are not detected then
        Generate a linear regression line between -1 and 1 with 10 points.
        Set the slope
    else
        Compute the linear regression line for the points detected.
function STEER_BETWEEN_WALLS(left_distances, right_distances)
    Identifies points on the left and right sides that are within a certain
  distance threshold (1.5 in this case) from the wall.
    if No points are detected then
        Set average distance to 1.5
    else
        Calculate the average right distance as the mean
```

E. Controller

We used simple proportional gain controller for the x position and θ orientation,

$Controlleroutput(\mu) = e_x \times p_x + e_\theta \times p_\theta$

e_x- error in position

p_x- proportional gain of x

$e_/theta$-error in orientation

$p_/theta$-proportional gain of θ

IV. 5 POINT: LIDAR-BASED EKF FOR LOCALIZATION

A. Problem Description

The goal of the LIDAR based EKF for localization is to combine a model of the car's dynamics with measurements from the LIDAR data in order to create the best estimate of the cars location and orientation relative to its starting position. In the prediction step, the current state estimate and control input are fed through a simple bicycle dynamic model in order to create an estimate of the new state. In the correction step, a LIDAR measurement is collected and compared to the LIDAR data from the previous step, in order to determine another estimate of the new state of the car. Finally, these two estimates are combined to give the best estimate of the cars current state, as well as the uncertainty

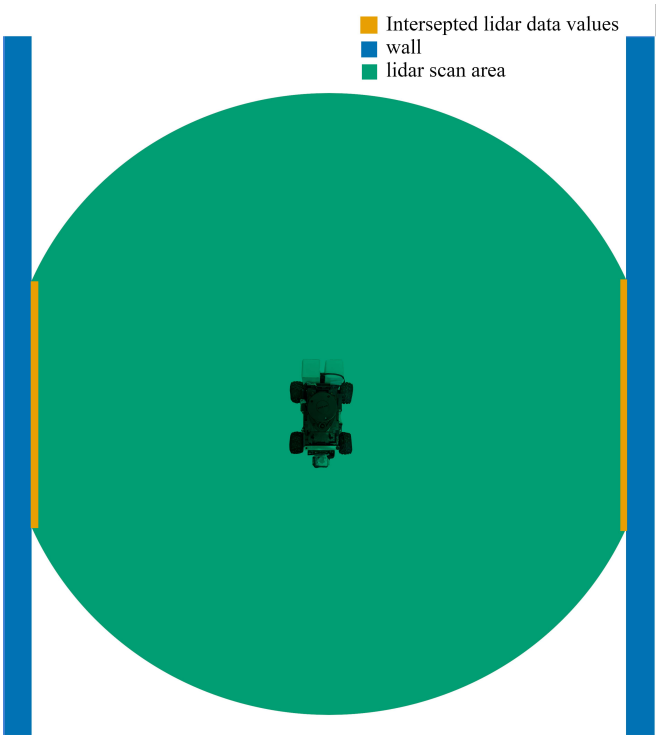


Fig. 2: LIDAR SENSING for relative position between walls

in this estimate. This process is repeated indefinitely as the car continues along its trajectory.

B. Implementation

Each iteration of the EKF algorithm assumes takes the previous state estimate and covariance (μ_{t-1} and Σ_{t-1}), the current control input (u_t), and the current observation (z_t) as inputs. To initialize the algorithm, the state estimate and covariance matrix are both assumed to be zero. This is because the starting position is considered to be the origin of the coordinate system, and there is no uncertainty in this initial state, since it was defined to be the origin. For state variables, x and y represent the spatial coordinates of the car, while θ represents the orientation of the car. For control variables, v is the speed of the car and δ is the steering angle of the car. The subscript t indicates the current time step, while subscript $t-1$ represents the previous time step.

During the prediction step of the algorithm, the bicycle dynamic model is used to estimate the current rate of change of the state based on the previous state estimate and the current control input. To estimate the new state, this rate of change is multiplied by the time elapsed since the previous prediction step (Δt) and added to the previous state estimate. This gives the relation between the new state and the old state as $\bar{\mu}_t = g(\mu_{t-1}, u_t)$, where g is the non-linear function shown in Equation 1. In this equation, L represents the length of the car, which we estimated as 0.15 m. The Jacobian of the non-linear dynamics equation is shown in Equation 2. This can be used with Equation 3 to determine the estimate of the new covariance matrix of the state after only the prediction step ($\bar{\Sigma}_t$). In this equation, R_t represents the covariance of

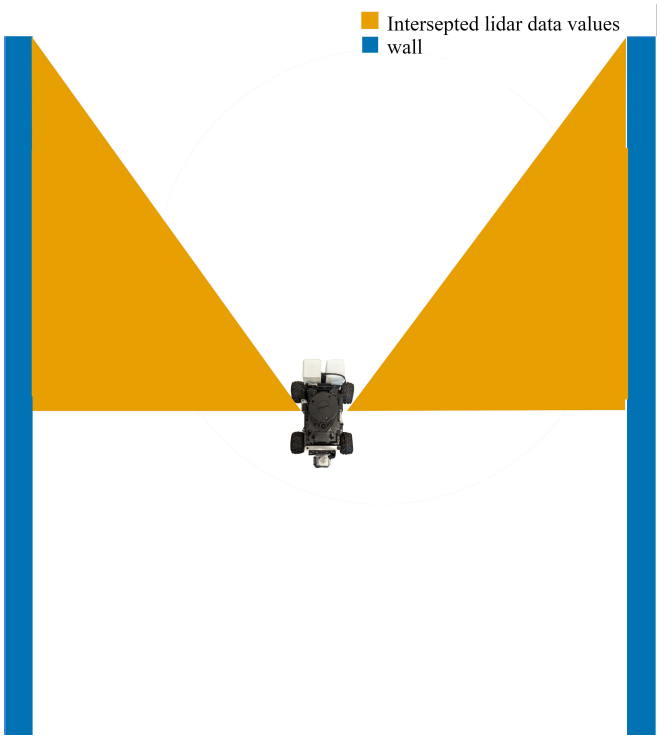


Fig. 3: LIDAR SENSING for relative orientation with respect to walls

the prediction noise.

$$g\left(\begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix}, \begin{bmatrix} v_t \\ \delta_t \end{bmatrix}\right) = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} v_t * cos(\theta_{t-1}) \\ v_t * sin(\theta_{t-1}) \\ \frac{v_t * tan(\delta_t)}{L} \end{bmatrix} * \Delta t \tag{1}$$

$$G_t = \begin{bmatrix} 1 & 0 & -v_t * sin(\theta_t) * \Delta t \\ 0 & 1 & v_t * cos(\theta_t) * \Delta t \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t \tag{3}$$

In order to perform the correction step of the algorithm, the LiDAR measurements need to first be converted in to a state estimate. This was done using the SimpleICP python library. When provided with the previous and current LiDAR point clouds, the algorithm produces the rotation and translation that would most closely match the points, allowing the robot to estimate its change in position and orientation. By adding this change to the previous state estimate, a new state measurement z_t is generated. Since z_t already takes the form of the state vector μ, the sensor model function $h(\mu)$, which converts state estimates into predicted measurements, is simply an identity. Since there are three states, the covariance of the sensor model, H_t, is the 3x3 identity matrix. Next, the Kalman gain is calculated using Equation 4. In this equation, Q_t represents the covariance of the sensor noise. Finally, the corrected state estimate (μ_t) and covariance (Σ_t) are generated using Equations 5 and 6.

$$K_t = \bar{\Sigma}_t H_t^T \left(H_t \bar{\Sigma}_t H_t^T + Q_t\right)^{-1} \tag{4}$$

$$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t)) \tag{5}$$

$$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t \tag{6}$$

C. Performance and Results

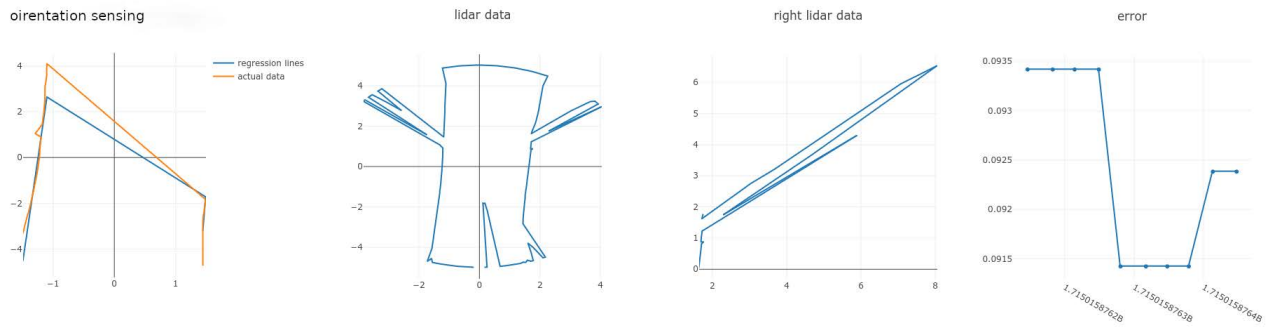
This EKF model was unable to achieve a high level of accuracy when attempting to localize the vehicle. A plot of the vehicles estimated trajectory is displayed in Figure . Additionally, as time continues, the covariance matrix tends to increase rapidly in size, meaning the the uncertainty in the estimates grows very large. A plot of the estimated trajectory over the duration of one lap of the course is illustrated in Figure 5.

D. Discussion

As can be seen from the trajectory in Figure 5, the EKF state estimate shows significant error from the true trajectory. Although the track was a loop, the filter was not able to achieve loop closure, and the estimates only loosely follow the direction of the course. There are many reasons why our team believes that the EKF was unable to maintain accuracy and precision over a long duration of time. For one, using a basic EKF model, without landmarks or loop closure, means there is essentially no way for the uncertainty in the state estimate to ever be reduced. Every new piece of information comes with its own uncertainty, and only serves to modify the previous state estimates, so the state uncertainty tends to grow endlessly. This can be seen in the inaccurate state estimate and large covariance matrix after the end of one loop around the track. While this growth in uncertainty is typical of all EKF models, our specific application had many of its own unique challenges.

There is quite a bit of uncertainty in the prediction step of the filter. For one, the bicycle dynamic model is at best an approximation, so it can not fully capture the true dynamics of the vehicle. Further reducing the accuracy of the model is the fact that it assumes the car has a fixed heading angle for each time step. The bicycle model is a continuous differential equation, but in order to use it to solve for the change in state, it must be discretized over a small time step. If the only concern was propagating the motion, the Δt for each step of the EKF could be split into arbitrarily small time units during the motion propagation, in order to iteratively achieve a more accurate solution. However, the use of iteration in the $g(\mu)$ function would make the computation of its Jacobian matrix unfeasible. For this reason, the motion is propogated for the full EKF time step Δt, while assuming the turning and steering angles remain constant, so that the Jacobian of the dynamics can be determined. This causes a loss in accuracy of the prediction step.

The correction step of the filter also introduces significant error in the trajectory. The sensor readings from the LiDAR



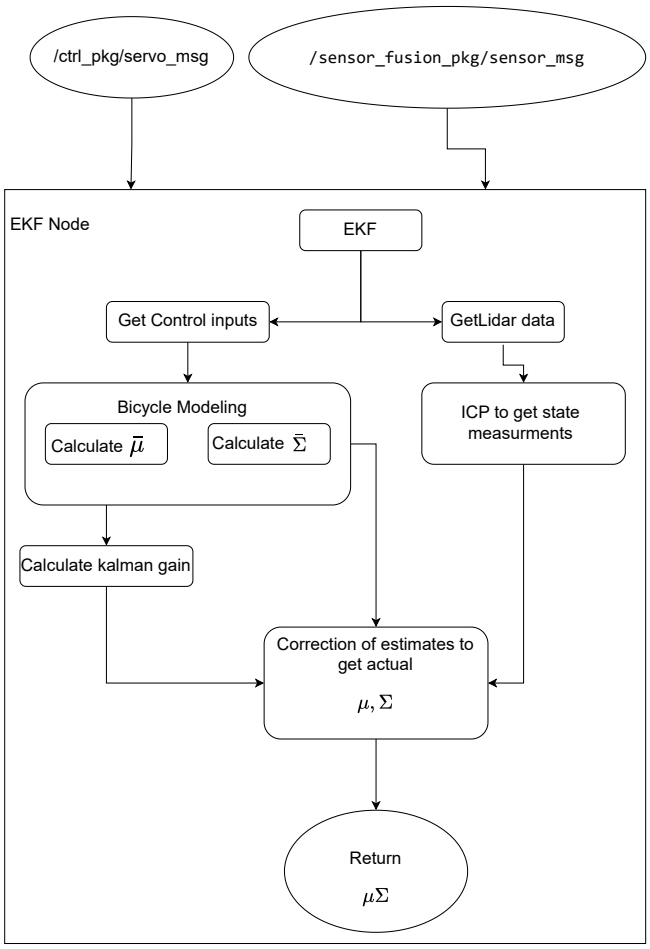


Fig. 4: Driver Node Code Flow

Algorithm 1 Extended Kalman Filter

function BICYCLEMODEL($u1, dt$)
  Initialize the state derivative vector
  Calculate the derivative of (x, y, θ)
  Update the (x, y, θ)
function EKF_STEP(u_1, *point_cloud*, dt)
  Define Process noise covariance R_t
  Define Measurement noise covariance Q_t
  Predict the state mean
  Calculate Jacobian of bicycle model function G_t
  Calculate prediction step covariance: $\Sigma_1^{'} \leftarrow G_t \cdot \Sigma \cdot G_t^T + R_t$
  Define sensor model adidentity, Jacobian H_t as identity
  Calculate Kalman gain: $K_t \leftarrow \overline{\Sigma} \cdot H_t^T \cdot \text{inverse}(H_t \cdot \overline{\Sigma} \cdot H_t^T + Q_t)$
  Perform correction step to update state estimate based on observations: $\mu \leftarrow \overline{\mu} + K_t \cdot (z_1 - \overline{\mu})$ $\Sigma \leftarrow (I - K_t \cdot H_t) \cdot \overline{\Sigma}$
  Save data to file
function OBSERVATION(point_cloud)
  if Previous point cloud is empty **then**
    Run SimpleICP algorithm
    Calculate state change based on transformation parameters
    Update previous pointcloud for next iteration

are incredibly noisy, and their conversion to a state estimate using SimpleICP is inconsistent. SimpleICP attempts to estimate the rotation and translation of a point cloud by trying to reduce the residuals between points in one cloud and the points nearest to them in the other cloud. Because this step assumes only rigid transformations apply to the point clouds, its accuracy is highly dependent on the fact that the point clouds have a relatively fixed structure. Since the LiDAR readings tend to fluctuate between measurements of the exact same location in the environment, it is no surprise that the SimpleICP method often only provides high uncertainty estimates of the change in state of the car. Additionally, in order for the EKF node to update fast enough to keep track of the cars changing position, the tolerance of the SimpleICP algorithm had to be increased. This causes the algorithm to be faster, but causes additional losses in accuracy. Even with this change, the SimpleICP algorithm was too slow to get an accurate sample of the control inputs, only receiving one input every few seconds. We believe that this is why the trajectory does not display many sharp turns, as we would expect. Often, the sharp turns happened during a period where the EKF algorithm was not sampling the control, so the large turning angle was never recieved as an input.

There are certain parameters in the Kalman filter algorithm that were difficult or near impossible to estimate. Although we assigned what we believe to be reasonable values to these parameters, we were unable to verify their accuracy. For one, we assumed that the distance between the axels of the car is 15 cm. Values also had to be estimated for the throttle and maximum steering angle, since these values are given no units in the AWS DeepRacer packages. We selected 0.25 m/s for the velocity of the car, and 30 degrees for the maximum steering angle of the car. Based on our observations, we believe these to be reasonable estimates; however, it any inaccuracies in these values would be reflected in the esti-

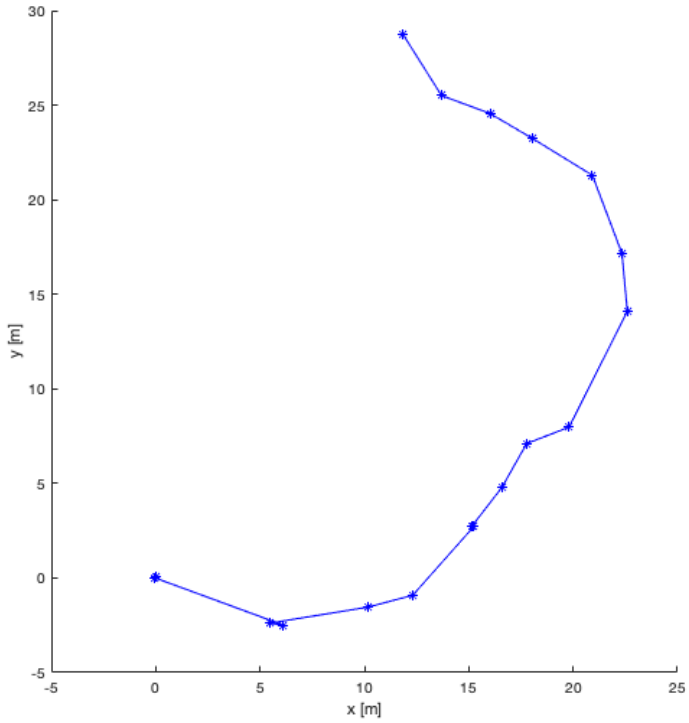


Fig. 5: EKF Estimated Trajectory Over One Loop of the Course

mated trajectory. Finally, we had to make the assumption for the values of the covariance of the prediction and correction noises, R_t and Q_t. Because we did not feel confident in our LiDAR data, we selected a large sensor noise covariance, setting it equal to the identity matrix. Since we felt more confident in the prediction step of the algorithm, we set the prediction noise covariance to be 10 times smaller than the sensor noise. These values were not measured nor verified, and likely lead to error in our estimates.

V. 2 POINT: REVERSE DRIVING

A. Problem Description

Similar to forward driving, the goal of this challenge is to make the car autonomously complete one lap around the track; however, the car must be moving in reverse. This results in the steering wheels of the car being located behind the center of force on the car, creating much less stable driving dynamics than the forward driving case.

B. Implementation

To implement this behavior, we simply reused the driving node for forward driving. To do this, we created a second mode that reversed the throttle, varied the control gains, and changed angle of the observed LiDAR readings. This means that the car was using a proportional controller in an attempt to center itself between the walls and align its orientation to match the walls orientation. We reduced the gains from their forward driving values in an effort to counteract the smaller steering radius that reverse driving results in. We also had to change the range of observed LiDAR measurements so that the car was looking backwards (forwards in this scenario).

Algorithm 1 Stop Sign Control Algorithm

```
function GET_CONTROLS(image)
    Detect if a stop sign is visible in the current image
    Check if there is a stop sign visible and conditions for stopping are met
    if stop and stop_count < 75 and go_count > 75 then
        Stop the vehicle
    else
        Continue driving
        if stop_count > 0 then
            Reset stop_count and go_count
function STOP_SIGN_VISIBLE(image)
    Extract red, green, and blue layers from image
    Apply color thresholds to identify potential stop sign areas
    Blur image and apply additional thresholding to refine stop sign detection
    Detect blobs using SimpleBlobDetector
    Check if any blobs (potential stop signs) were detected
    if any blobs detected then
        Indicate stop sign is visible
    else
        Indicate no stop sign detected
```

C. Performance and Results

The car was able to successfully complete a loop around the track, and car was able to maintain stability when located between two walls. However, the car did have a few crashes into the wall, similar to the forward driving challenge. The car also had some difficulty performing the 90 degree

turns around the corners, although it could occasionally successfully perform this behavior.

D. Discussion

Achieving stability when centering between the walls was one of our main goals for this challenge. This is difficult to achieve due to the unstable reverse driving dynamics. Using the same position and orientation control laws as forward driving with lower gains appears to have been an effective solution to this issue. The crashes could likely be reduced using many of the same methods as for the forward driving, primarily by performing better filtering of windows and other indentations in the wall in the turn detection algorithm. The turning behavior could likely be improved by further refining the observed LiDAR angles and the steering angle that is commanded when a turn is detected.

VI. 1 POINT: STOP SIGN

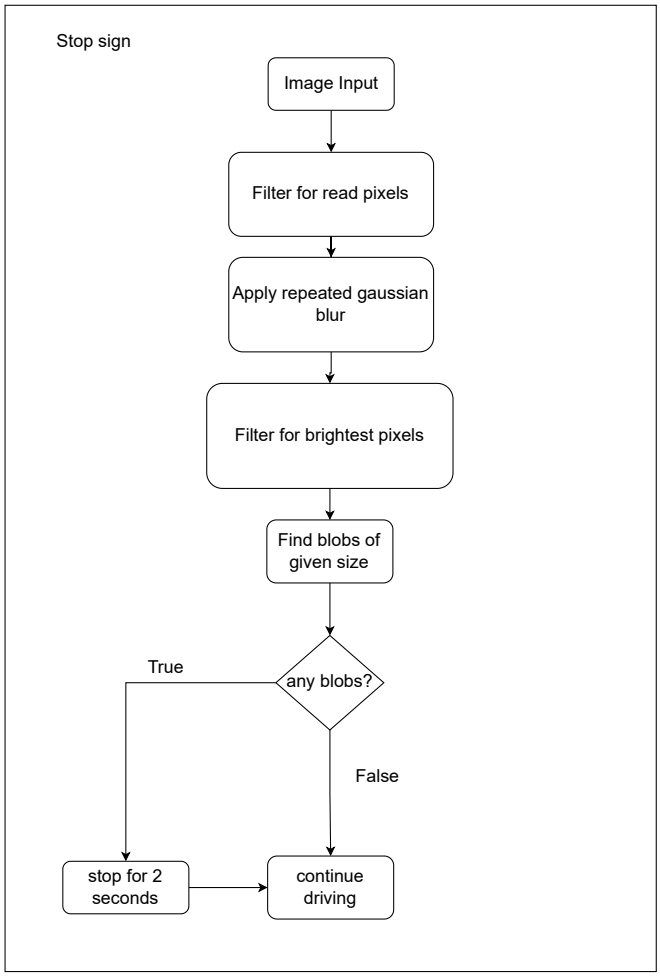


Fig. 6: Driver Node Code Flow

A. Problem Description

For the stop sign problem, the vehicle needed to use its onboard cameras to detect a stop sign, briefly stop moving, and then continue forward. This is largely a computer vision problem, and we elected to solve it using a blob detection algorithm.

B. Implementation

As stated previously, blob detection was used to detect the stop sign in the cameras images, specifically with the library OpenCV. First, the RGB image is filtered for red pixels. This was achieved by setting all pixels with a red value greater than 150 and green and blue values less than 100 equal to 0, and all other pixels to 255. Often, like in the case of the text on a stop sign, this process can produce many small red blobs, instead of one larger one. Since the algorithm is searching for a stop sign of a specific size, and attempting to stop a certain distance away, the blob finding process was filtering out all blobs below a certain area. A repeated Gaussian blur was applied to the image to cause the small blobs on the stop sign to merge in to one larger blob that could be detected. Finally, all of the pixels with a value below 100 were set to zero, and the remaining pixels were set to 255. By this point, all red objects have been transformed into uniform dark blobs on a white background. After this, the image was passed to the OpenCV blob detection algorithm. If any blobs were detected, the algorithm activated a flag to indicate that the stop sign was in view. If this flag was seen for more than 3 iterations, the car would stop for 2 seconds. The car would then continue forward for two seconds without searching for the stop sign. This was intended to allow the car to pass the stop sign after it has already stopped once, instead of stopping multiple times. A flow chart of this algorithm can be seen in Figure 6.

C. Performace and Results

The stop sign algorithm was able to successfully detect and stop at a stop sign; however, there were a few issues with the behavior in its current iteration. For one, the car would often stop when the stop sign was not in view of the camera. This would typically happen when another object with a color close to red was in view. The car also tended to stop multiple times at the same stop sign.

D. Discussion

As stated previously, the stop sign algorithm had two major errors: stopping when the stop sign was not in sight and stopping multiple times for the same stop sign. In order to fix the first issue, various parameters of the code could be further refined, in order to better filter out objects that are not the stop sign. For one, the red maximum red threshold and minimum green an blue thresholds needed to pass through the filter could be better refined for the color of the stop sign. Additionally, an HSV color format could be used to allow the algorithm to be more resistant to varying lighting conditions. The OpenCV blob detection algorithm also has various filters that could be used to more accurately detect the stop sign. Two such filters that could be helpful in the future are the circularity and convexity filters, both of which could be used to only select blobs that are relatively round. There are two main changes needed to fix the second issue. For one,

the minimum area of blobs detected could be increased, so that the car only stop when it is much closer to the sign. Additionally, the duration that the car is unable to stop after stopping once should be increased, so that the car is able to get the stop sign outside of its field of view. Both of these fixes would prevent to car from stopping multiple times at the same stop sign.

VII. CONCLUSIONS

Based on our result and performance analysis, we believe that we can improve the performance for forward driving of the car in two ways. One, we would need a properly functioning LiDAR since the one on our bot is faulty it does not give the entire range of readings (We don't get LiDAR readings between -15 deg to 15 deg) and we get very large readings in this region. Two, Implementation of derivative control this gives the resistance to changes thereby making the bot more stable. For improvement in EKF performance, again a properly functioning LiDAR would make our reading more reliable since, having a faulty LiDAR also affected out implementation of EKF. Another improvement could be that we only considered pose and orientation in our model, having Landmarks in our environment would help in model our Jacobian more accurately thereby improving our uncertainity estimate. In case of the stop sign, we can improve the current methodology used by incorporating the use of neural networks this would help detect the stop sign as it is in actuality instead of approximating it to a blob. Overall, we attempted to use what worked best for our bot but there definitely are ways that the performance can be improved.

VIII. ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Professor Jake Brawer, whose guidance and expertise were invaluable throughout this project. Their support and encouragement helped us navigate through challenges and achieve our goals.

We are also grateful to Alec Reed, our teaching assistant, for their assistance in clarifying concepts and providing valuable feedback on our work.

Finally, we extend our thanks to the college (University of Colorado, Boulder) for providing the necessary resources and facilities for conducting this research.

APPENDIX: GROUP B CHALLENGES

Literature Review on Learning from Demonstration

Check references [2], [3] and [4]

Learning from Demonstration (LfD) is a technique in which an agent learns a task by observing demonstrations provided by a teacher or expert. This approach has gained significant attention in the field of robotics and autonomous systems due to its ability to enable robots to learn complex tasks efficiently.

Introduction to Learning from Demonstration: Learning from Demonstration encompasses a variety of techniques, including imitation learning, apprenticeship learning, and inverse reinforcement learning. The key idea behind LfD is to leverage human expertise to teach robots how to perform tasks without explicitly programming them. By observing demonstrations, robots can learn task-specific policies or models that generalize to new situations. Reinforcement learning can be modelled as a Markov decision process and using and the algorithms can be divided into three classes: value-based, policy gradient, and actor-critic algorithms. Value-based algorithms (e.g. Q-learning) estimate the value function V(s), which represents the value (expected reward) of being in a given state. Policy gradient algorithms (e.g REINFORCE) do not estimate a value function, but instead parametrise the policy and then update the parameters to maximise the expected rewards and actor-critic algorithms (e.g. A3C), are hybrid methods which combine the use of a value function with a parametrised policy function.

Applications of Learning from Demonstration: Recent research has demonstrated the effectiveness of LfD in various domains, including manipulation tasks, navigation, and human-robot interaction. In the context of autonomous vehicles, LfD can be used to teach vehicles how to navigate complex environments, follow traffic rules, and interact with pedestrians and other vehicles safely.

Challenges and Future Directions: While LfD shows promise for autonomous vehicles, several challenges remain, such as robustness to noise and uncertainty, and the need for large-scale demonstrations. Computation is a challenge due to the large amount of data required to train deep learning models. Architectures, due to the difficulty of choosing the optimal network architecture for a given task. Goal specification is a challenge for reinforcement learning techniques due to the importance of designing a reward function which promotes the desired behaviour and adaptability and generalisation is a challenge in the autonomous vehicle domain due to the highly complex nature of the operational environment. Future research directions include developing algorithms that can learn from sparse or imperfect demonstrations, incorporating safety constraints into learning algorithms, and enabling robots to adapt autonomously to changing environments.

Literature Review of Deep Learning Techniques for Autonomous Control

Check references [5], [6], [7], [8] and [9]

Deep learning techniques have revolutionized many fields, including autonomous control, by enabling end-to-end learning of complex control policies from raw sensor data. In this literature review, we provide an overview of recent research on deep learning techniques for autonomous control and discuss their potential application to improving our controller for the autonomous vehicle challenge.

Introduction to Deep Learning for Autonomous Control: Deep learning methods, such as deep neural networks (DNNs) and convolutional neural networks (CNNs), have

been successfully applied to various autonomous control tasks, including perception, planning, and control. These methods learn hierarchical representations of input data and can capture complex relationships between inputs and outputs. DL techniques are classified into three major categories: unsupervised, partially supervised (semi-supervised) and supervised. Furthermore, deep reinforcement learning (DRL), also known as RL, is in the category of partially supervised (and occasionally unsupervised) learning techniques. The main benefit of CNN compared to its predecessors is that it automatically identifies the relevant features without any human supervision

Applications of Deep Learning in Autonomous Control: Recent research has demonstrated the effectiveness of deep learning techniques in various autonomous control domains, including lane keeping, obstacle avoidance, and path planning. Deep learning models have been used to learn control policies directly from sensor data, bypassing the need for specifically handcrafted features or explicit modeling of the environment.

Trade-offs, Considerations and Future Directions: While deep learning techniques offer significant advantages for autonomous control, they also pose challenges, including the need for large amounts of labeled data, computational complexity, and interpretability issues. Additionally, deep learning models may struggle to generalize to unseen scenarios or handle safety-critical situations and issues such as user acceptance, cost efficiency, machine ethics for artificial intelligence technologies, and lack of legislation/regulation for autonomous vehicles must also be addressed. Future research directions include developing algorithms that can learn from sparse or imperfect demonstrations, incorporating safety constraints into learning algorithms, and enabling robots to adapt autonomously to changing environments.

Comparison of Control Approaches

In this section, we compare two control approaches for the autonomous vehicle challenge: controlling just the centering and controlling the centering and orientation of the bot. We discuss the implementation details of each approach, evaluate their performance on relevant metrics, and provide insights into why we chose one approach over the other for our final controller.

controlling just centering did not work because the when the bot is very close to walls the error correction spikes up and give us huge steering angle we tried saturating the output but that did not help, we used derivative filter but the system unstable to then we realised that bot state is not just represented by x,y even orientation needs to corrected so we added an additional orientation control to make the bot align to walls instead of turning 90^0 this improved the bot stability upon tweaking the gains

REFERENCES

[1] SimpleICP: https://github.com/pglira/simpleICP/tree/master/python

[2] M. Kuderer, S. Gulati and W. Burgard, "Learning driving styles for autonomous vehicles from demonstration," 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 2015, pp. 2641-2646, DOI: 10.1109/ICRA.2015.7139555.

[3] P. M. Kebria, A. Khosravi, S. M. Salaken and S. Nahavandi, "Deep imitation learning for autonomous vehicles based on convolutional neural networks," in IEEE/CAA Journal of Automatica Sinica, vol. 7, no. 1, pp. 82-95, January 2020, DOI: 10.1109/JAS.2019.1911825.

[4] Harish Ravichandar, Athanasios S. Polydoros, Sonia Chernova, and Aude Billard Belmont, CA: Wadsworth, "Recent Advances in Robot Learning from Demonstration" Vol. 3:297-330

[5] S. Kuutti, R. Bowden, Y. Jin, P. Barber and S. Fallah, "A Survey of Deep Learning Applications to Autonomous Vehicle Control," in IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 2, pp. 712-733, Feb. 2021, DOI: 10.1109/TITS.2019.2962338.

[6] Li, Dong, et al. "Reinforcement Learning and Deep Learning based Lateral Control for Autonomous Driving," 2018. arXiv preprint arXiv:1810.12778.

[7] Grigorescu, Sorin, et al. "A survey of deep learning techniques for autonomous driving," Journal of Field Robotics, vol. 37, no. 1, pp. 42-116, November 14, 2019, DOI: 10.1002/rob.21918.

[8] Alzubaidi, L., Zhang, J., Humaidi, A.J. et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data 8, 53 (2021). https://doi.org/10.1186/s40537-021-00444-8

[9] Li, Guanpeng, et al. "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17), Article 8, 1–12, November 2017, DOI: 10.1145/3126908.3126964.