Maleficent
CSCI 5302 Advanced Robotics – Final Project Report

Andy Ho
CU Boulder
andrew.ho@colorado.edu

Matt Ward
CU Boulder
mawa6643@colorado.edu

Mohammed Adib Oumer
CU Boulder
mohammed.oumer@colorado.edu

Rameez Wajid
CU Boulder
rameez.wajid@colorado.edu

***Abstract*—Creating a reliable control algorithm for autonomous vehicles has been, and continues to be, a difficult problem to solve for computer scientists around the world. Developing a reliable algorithm that is both computationally inexpensive and easy to implement is very complex. The contributing team members of this project have begun to delve into this world of autonomous vehicles and their associated control algorithms in an attempt to implement some of those that already exists, and gain a better understanding of their intricacies and inner workings via an Amazon AWS Deepracer car. Documenting all findings and experiences that the team faced throughout the project, this paper serves as a record and testament to the trials and tribulations of implementing control algorithms for an autonomous vehicle. For the laps around the course, the team was able to implement two PID controllers: one for wall following, and another for turning. The challenges attempted were implementing volumetric SLAM, driving in the reverse direction, and building a user interface. Literature review surveys are included in the appendices.**

I. Introduction

Algorithms used to facilitate autonomous motion are becoming increasingly relevant and useful as the world evolves toward the mechanization of simple to complex tasks. In response to these developments in the world, many scientists have been creating algorithms for which to operate these autonomous machines. After years of development of these algorithms, multiple different varieties and methods have been developed to perform tasks of an equivalent variety. SLAM prevails as one of the most popular and versatile of such algorithms when used for autonomous driving, making it an excellent candidate for the team to implement.

Therefore, this project team aims to successfully implement and test the SLAM algorithm – along with other autonomous controllers – to navigate a set course in an effort to gain a better understanding of how autonomous driving works, as well as to understand the benefits and drawbacks associated with it. This paper serves as a record of the efforts that the team put forth and the results that were achieved when working with the AWS Deepracer car and various controllers meant to operate the car, including SLAM.

II. System Details

In order to drive around a course autonomously, a vehicle and a course are required. For the purposes of this project, the team was given an Amazon AWS Deepracer car. This vehicle possesses a single-lens front-facing camera, a stereo camera, a 360° planar LiDAR, and a built-in IMU; each of which it uses to retrieve data about its current environment and state. It is powered by two batteries; one for the motors and the other for computational components and sensors. The car locomotes via a single motor which powers the rear wheels. As for software, the car operates on Ubuntu 20.04 for the middleware and the Foxy version of ROS2.

The course that the car traversed consisted of a series of hallways in the engineering building at the University of Colorado Boulder, forming a rectangular shape. As a result of the course being comprised of university hallways, there exist multiple doorways and inconsistencies along the walls and turns. This is to say that the course was far from an ideal, perfect rectangle. Additionally, the sensor data published by each of the Deepracer's sensors were not entirely void of noise and could sometimes lead to undesirable behaviors.

III. Implementation

In the initial stages of the project, the team attempted to work on a simple open-loop controller that would have the capability to navigate the specified course. The purpose of developing this controller was to test the sensors and general mechanics of the car while making use of prior information about the race course. Two variations of driving policy were tested in the process of trying to get a working controller. The first policy that the team attempted was a centering policy. The idea was to center the car based on the difference between the right and left side LiDAR sensor readings. A turn to the opposite side was executed once readings from one side were larger than the other side. For example, if there were a much larger right reading than the left, the car would initiate a right turn. A race line range was set for the robot to stay within that line, however, the robot was not able to stay within that line due to the controller's configuration. The team attempted to fix the issue by increasing the line range, but this did not seem to help much. The team also tried to accommodate for false turns and cases where there would be a wall in front of the vehicle. Upon reaching a false turn or a corridor, the car would assume that it was taking a turn and attempt to adjust back to the center line; however, it would overcompensate and make too wide of a turn, causing it to crash into the wall. A potential reason for these issues may have been latency in receiving and publishing messages, resulting in the robot not being able to react fast enough to correct itself. Due to the controller not performing as expected after multiple hours of testing, the team decided to pivot to another method.

The second policy of interest was a right-wall following policy. The advantage of this policy is that it makes use of the prior information regarding the race course to only tune with respect to one variable, the right side readings, instead of accounting for both the left and right side. The team anticipated this policy to be more effective at completing the track than the centering policy. It turned out that during straight hallways, the car was able to perform extremely well. The only issue was figuring out how to handle turns. Much like the centering controller, the robot would overcompensate for turns and end up crashing into the walls. It was difficult to pinpoint the correct tuning parameters which could lead to reliably completing the course. The policy ultimately kept failing due to inconsistencies in the course such as alcoves or hallways with varying widths. This was still an open-loop controller, but it helped reinforce the understanding of creating a better controller.

These failures led to the decision to make use of a closed-loop feedback PID (proportional, integral, and derivative) controller. While the previous methods required manually setting the steering angles and speed with respect to the LiDAR readings, the PID intuitively determines the parameters that are needed for steering based on the characteristics seen while driving the robot. The impacts of the three PID controller parameters, typically denoted by K_p, K_i, and K_d respectively, towards the anticipated movement of the robot, were better understood by consulting resources such as an MIT Aerospace Controls Lab video [1].

Each of the parameters of the PID serves its own distinct purpose. K_p pushes the robot proportionally towards the reference point using the error (difference between the current position and the reference point). K_i helps push the steady-state error (the error after the robot has found the reference point) towards 0, thereby readjusting the robot back to the reference point in case it deviates away from it. K_d adjusts the rate at which the robot reaches the reference point, slowing down the impact of K_p when the robot is close to the desired reference point.

Based on observations on the track, the team decided to implement two PID controllers: one for the wall following through a hallway and another one for turning right at appropriate intersections.

A. First PID - Wall Follow

In the centering policy, the desired reference point is 0 (i.e., the difference between left and right side readings is 0). In the right wall following policy, the desired reference point was 70 centimeters away from the right wall which was determined based on prior knowledge of the course dimensions. The wheel throttle was set based on the battery level while the PID output was used to steer the robot. The PID output for the orientation was limited to be within 30% of the maximum steer ($\approx 30°$) both to the right and to the left. The set of initial controller parameter values $\{1.0, 0.0, 0.1\}$ were used, but this exhibited marginal stability characteristics, causing the car to snake through the hallways, weaving back and forth away from the desired reference input. This would also lead to some of the turns becoming too tight to pass through and the robot crashing into a wall (which was seen even more frequently in the centering policy). The centering policy led to a lot more crashes and, as mentioned earlier, tracking the two sides of the robot was an unnecessary and avoidable overhead. As such, the team decided to stick with the wall following policy. After researching and experimenting with different tuning values, the set of values $\{1.0, 0.0, 0.95\}$ were found to be sufficiently optimal for achieving the desired trajectory with this policy.

The main challenge here was dealing with the inconsistencies within the architecture of the building, particularly the alcoves. To counteract this problem, a mechanism alongside the PID controller was implemented which would sense a wall in front of the car and reverse at the appropriate angle to navigate out of the alcove. During the team's test trials, it successfully navigated out of the alcove approximately 80% of the time. The other 20% of the time, it would get stuck in a back-and-forth movement in the alcove indefinitely. Moreover, the success of this process appeared to depend significantly on the motor battery level at the time of the trial. It was found that as the battery's power level dropped, the car became slower. While this increased the time spent performing the lap, the slower speeds were found to be the safest way to complete the course.

B. Second PID - Turn

In order to make turns, the team programmed a second PID controller. This choice was made because the first PID was able to wall follow with minimal adjustments to the steering angle but the robot required a lot more steering to make the turns swiftly. The reference point for this controller was set at 0.85 and the steering angle limits were set within 40% of the maximum steer. The wheel throttle was set based on the battery level, however, this was not always reliable. Even with this dynamic throttle setting, the robot's speed was inconsistent between each run.

After a turn is executed using the second PID, the first PID will then take over to continue following the wall to the right of the robot. For this controller, a set of controller parameters $\{2.0,0.0,1.8\}$ were found to be optimal. Occasionally, depending on the battery level that impacts the wheel throttle, the PID would not execute an attempted turn perfectly, overcompensating and driving in circles while looking for the right wall instead of switching to the wall following PID as intended. Relevant parameters for both PIDs are summarized in Table I.

IV. Major Implementation Challenges

Throughout the implementation phase, the team faced constant connection issues with the car. This came in a variety of forms. The first and earliest experienced connectivity issue was that of the car's nodes and topics randomly ceasing to show. Once they disappeared, it was impossible for the team to continue testing without first restarting the car, requiring the team to wait several minutes to reboot. This led to a lot of idle time, waiting for things to work. The other connection issue was in terms of connecting to the robot in certain sections of the course. For instance, Windows 10/11 requires an internet connection to utilize a mobile hotspot, and if the device hosting the hotspot disconnects for whatever reason, the hotspot also turns off and the robot disconnects leading to the robot veering off on its own.

Additionally the LiDAR would occasionally stop working, or its respective ROS node would disappear, which would once again require a reboot. As it turns out, it is impossible to navigate the course while using information from the LiDAR when that same LiDAR is not publishing any information. Lastly, the car would periodically disconnect from the internet. This issue has served as a major barrier for the team. Constantly disconnecting and reconnecting to the car takes a significant amount of time, taking away from the team's overall ability to tune parameters within a semester-long time frame.

In terms of the basic setup and configurations in regards to SLAM. The team had downloaded all of the necessary packages to run them on the robot. One issue was that the team needed to connect local machines and the robot on the same ROS_DOMAIN_ID. The team knew that in order to run anything, that was the first step in doing so. After searching for and finding the configuration file where the domain ID could be changed, nodes were finally running on the set unique domain ID.

Another issue arose, however. The nodes appeared only on the robot, but not on the team's machines despite being on the same domain ID. Even after disabling firewalls, nodes were still not appearing. The team figured that it was an issue with the team's machines. Originally, the connection between the robot and local machines was connected via ssh by WSL. This caused a major issue as a connection was not being established between the robot to local machines. The team's machines were able to ping the robot, but not the other way around. The issue using WSL causes there to be a different IP than the local machine, therefore the team was unable to establish a connection over the same ROS_DOMAIN_ID. Consequently, the team decided that it would be best to create that connection on a Linux machine instead of WSL. This resolved most of these issues in terms of establishing connections.

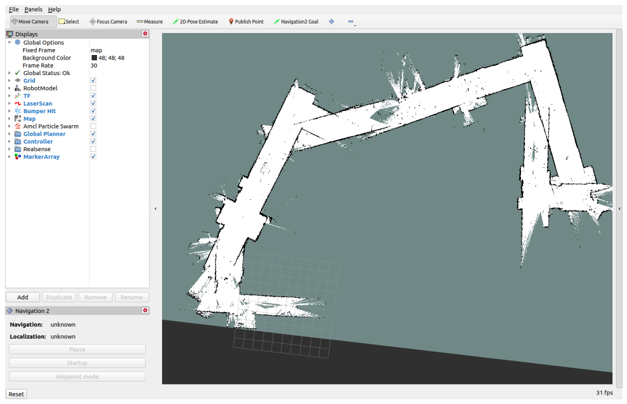
Another issue also arose when launching the Nav2 and Slamtoolbox packages and nodes. They were originally launched on the robot, but the robot's hardware was not able to support the Slamtoolbox specifically. This was due to a lack of CPU computational power and memory to support it causing the robot to crash repeatedly; so the Linux machine was critical for making the system work. It allowed the team to launch the toolbox on the local machine with the capabilities of handling the toolbox as well as Rviz.

V. Performance Analysis

As mentioned in the implementation section, the team was eventually able to arrive at an optimal tuning for the PID controller so that it operated as intended. While the car was able to complete the course using this controller, there were several locations along the course where the car would become stuck. Alcoves and different textured wall materials would create varied sensor readings, causing the car to drift from its course. This would inevitably lead to the car heading straight towards a wall, resulting in a small distance reading from the front of the car below the team's specified stopping threshold. Because the team has implemented a reverse function in the PID though, the car would then reverse and change its steering angle. One can imagine that this would work relatively well in theory, but due to the design of the PID, it is difficult to change the steering angle to such a degree that it is able to become unstuck in these scenarios easily. Additionally, returned noise in the LiDAR sensor readings made steering away from whatever obstacle was inhibiting the car very difficult.

TABLE I
Relevant Information of the PID parameters

	Reference Point	K_p,K_i,K_d	Steering Limit
PID I - forward	0.7	1.0,0.0,0.95	[-9°,9°]
PID II - forward	0.85	2.0, 0.0, 1.8	[-12°,12°]
PID I - reverse	0.8	0.85,0.0,0.8	[-9°,9°]
PID II - reverse	1.0	1.0, 0.0, 0.9	[-18°,18°]



Global Options
Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Global Status: Ok
Grid
RobotModel
TF
LaserScan
Bumper Hit
Map
Amcl Particle Swarm
Global Planner
Controller
Realsense
MarkerArray

Navigation: unknown
Localization: unknown

Fig. 1. Mapping attempt with collisions

VI. Challenges Attempted

A. SLAM

For the SLAM portion of the project, multiple methods were investigated. Several GitHub repositories and packages were examined, but the team eventually settled on using Nav2 for all mapping purposes and Rviz to visualize what the car was reading from the LiDAR. Using a teleop system, the team was able to manually traverse the course as a means of creating a map. During this process, if the car crashed into a wall due to user error or a disconnected signal between the controller and the car, then the map that was actively being created would become disoriented and inaccurate, requiring the team to restart the map (Fig. 1).

After several hours of attempting to map the course manually, the team attempted to use the existing PID controller

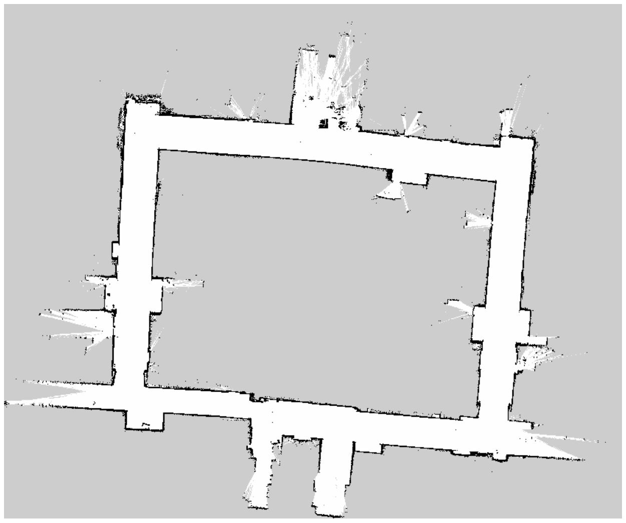
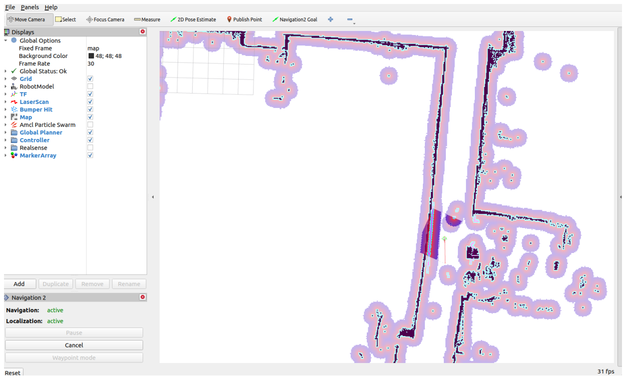


Fig. 2. Successful mapping of the course without collisions



Global Options
Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Global Status: Ok
Grid
RobotModel
TF
LaserScan
Bumper Hit
Map
Amcl Particle Swarm
Global Planner
Controller
Realsense
MarkerArray

Navigation: active
Localization: active

Fig. 3. Localization and navigation using the created map

to drive the car around the course while mapping. Nav2 and the PID controller were not, however, able to operate simultaneously, leaving the team to resort to manually driving the car again. A map (Fig. 2) was eventually produced though after a successful run, free of collisions.

Once the map was completed, the team was able to position the robot and localize on the map (Fig. 3). The figure shows the global costmap around the objects and walls, highlighted in purple. Where color contrast is more vibrant than the surrounding areas is where sensor data is currently coming in from i.e the local costmap.

The tools used for referenced in [2] and [3].

B. Display Interface

In order to better visualize the relevant information that the car reads during driving, the team created a simple code block that plots the distance from the right wall in meters and superposes the current speed, angle, and front sensor distance data on the plot. This data is plotted every 10 LiDAR readings (or $\approx$ 0.01 seconds) since too many data points are plotted at the base frequency of the LiDAR output of 10Hz. Some examples of the sample run plots are shown in Fig. 4: (a) and

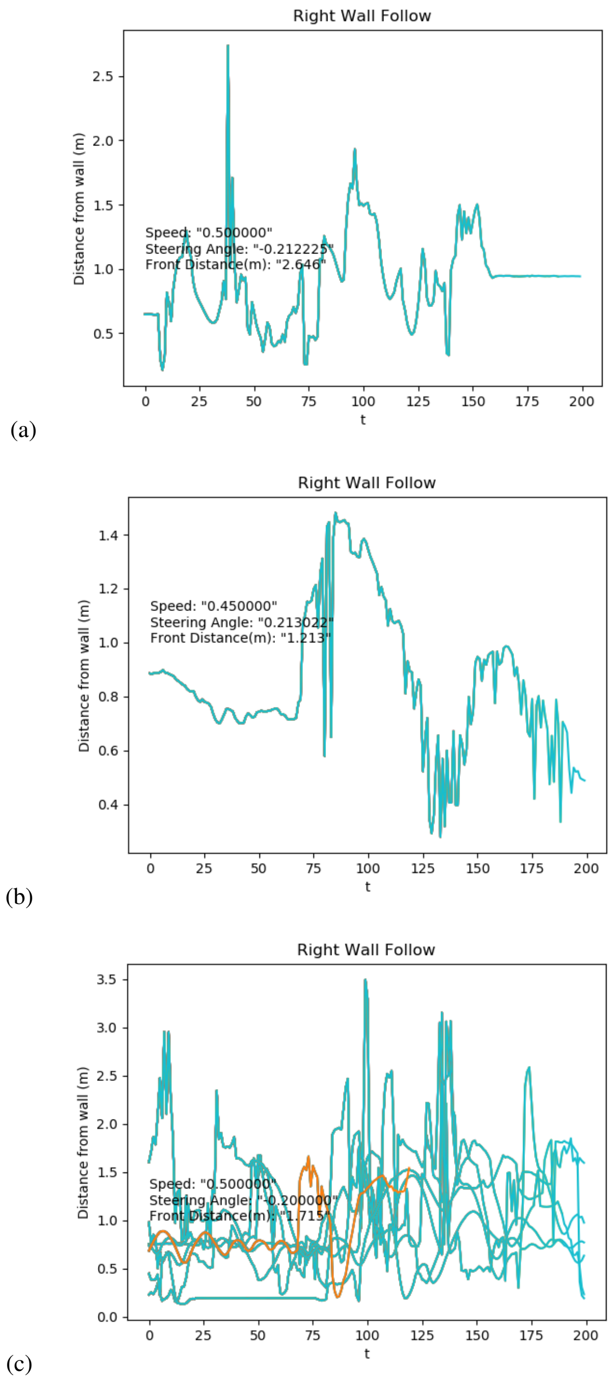


Fig. 4. Vehicle telemetry information from two sample runs

(b) show examples of plots during successful turns while (c) shows an example when manual adjustment was needed to readjust the robot when it is stuck in alcoves.

C. Reverse Driving

The reverse driving was implemented by modifying the script for the forward drive. The modifications made were editing the relevant LiDAR readings to use the true back as the "front" reading and flipping the right and left LiDAR readings. Using the intuition and ballpark values obtained from the original configurations for the two forward driving PID controllers, necessary adjustments were made to the reverse PID controller parameters. The relevant parameter values from both the forward and reverse drive are shown in Table I. In addition, the sign of the wheel throttle and steering angle commands had to be reversed (with a negative sign) and modified. The rear wheels are harder to steer and as such, more time was dedicated to account for how the car would drive differently based on the sensor ranges as well as the calibrations. After a relatively small number of trials, the car was able to make a full lap around the course, albeit very slowly and with a bit of overshoot in both directions. Better safe than sorry :).

D. Different Control Approaches

In total, there were 3 classes of controllers tested: open-loop reactive controllers, closed-loop PID controllers and SLAM-based controllers. A centering controller and wall following versions were tested for both the open-loop and closed-loop controllers. The open loop controllers were part of the early development stages. The wall following PID controller was exhibiting better characteristics during run time and was chosen as the final controller. This allowed the robot to dynamically change its steering angle when necessary and the speed was assigned with respect to the motor battery level. There were two PID controllers in the final implementation, one for wall following and one for right turns. This solved most of the issues with the open loop controllers, as they were able to drive straight, but unable to make good turns. Interestingly and perhaps counter intuitive to normal human driving, the speed of the robot during turns was larger by 10-20% of the speed used when driving straight through the hallways.

The last class of controllers attempted was the SLAM-based navigation using waypoints. The software was able to plan the path to the desired goal point. However, due to technical difficulties that could not assign a non-zero throttle to the robot, the team was unable to drive the robot using SLAM. With more time, the team believes that the problem could be fixed to demonstrate a working a SLAM map, path planner and driver.

References

[1] AerospaceControlsLab, "Controlling self driving cars - youtube." https://www.youtube.com/watch?v=4Y7zG48uHRo. (Accessed on 12/05/2023).

[2] "Introduction to the ros navigation stack using aws deepracer evo." https://github.com/aws-deepracer/aws-deepracer/blob/main/introduction-to-the-ros-navigation-stack-using-aws-deepracer-evo.md. (Accessed on 12/10/2023).

[3] "(slam) navigating while mapping — nav2 1.0.0 documentation." https://navigation.ros.org/tutorials/docs/navigation2_with_slam.html. (Accessed on 12/10/2023).

[4] Z. Xie, Q. Zhang, Z. Jiang, and H. Liu, "Robot learning from demonstration for path planning: A review," *Science China Technological Sciences*, vol. 63, no. 8, pp. 1325–1334, 2020.

[5] Z. Zhu and H. Zhao, "A survey of deep rl and il for autonomous driving policy learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 14043–14065, 2021.

[6] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635, JMLR Workshop and Conference Proceedings, 2011.

[7] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end autonomous driving," *arXiv preprint arXiv:1605.06450*, 2016.

[8] A. Y. Ng, S. Russell, *et al.*, "Algorithms for inverse reinforcement learning.," in *Icml*, vol. 1, p. 2, 2000.

[9] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, *et al.*, "Maximum entropy inverse reinforcement learning.," in *Aaai*, vol. 8, pp. 1433–1438, Chicago, IL, USA, 2008.

[10] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[11] F. Behbahani, K. Shiarlis, X. Chen, V. Kurin, S. Kasewa, C. Stirbu, J. Gomes, S. Paul, F. A. Oliehoek, J. Messias, *et al.*, "Learning from demonstration in the wild," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 775–781, IEEE, 2019.

[12] 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.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

APPENDIX1:LEARNING FROM DEMONSTRATION

Learning for robotics has always been a challenging area of research. The problems are generally associated with uncertain environments, complex perception scenarios and domain specific challenges. Learning from Demonstration (LfD), however, offers a promising paradigm for effective learning in challenging robotics task executions [4]. LfD is generally defined as the algorithms where the policies are learned based on demonstrated data.

Trajectory following may be a hard task for many applications as either an optimal trajectory might not be realizable or the robot may find it difficult to follow hand-engineered trajectories when even minor changes occur in the environment. Imitation Learning(IL) is employed to help the robot in mapping optimal actions to states. Whereas, Inverse Reinforcement Learning (IRL) is utilized for intent estimation in case of an expert demonstrator. These are the two main paradigms currently being followed in LfD research.

General framework

Problem.: A demonstration dataset contains a set of trajectories, where each trajectory is a sequence of state-action pairs, and action is taken by expert at state under the guidance of an expert policy. Given the dataset a common optimization-based strategy is to learn a policy that mimics the expert's policy. This is often ensured by minimization over a selected similarity measure between policies.

Behavior Clone.: In behavior cloning (BC), the problem is taken as a supervised learning process where the objective is to mimic the expert policy. Although L2-loss is most commonly used as the minimization metric, but other metrics are also being utilized recently [5]. This cloning approach generally works well for states present in the original demonstrations, but is fragile when it comes to unseen states. This problem was solved by DAgger (Dataset Aggregation) [6]. DAgger improves on behavioral cloning by training on a dataset that better resembles the observations the trained policy is likely to encounter, but it requires querying the expert online. SafeDAgger [7] is an improvement on DAgger and is query-efficient and more suitable for end-to-end autonomous driving.

Imitation learning for robot path planning

Robot imitation aims to reproduce the expert demonstrations provided. For robot mation imitation, two main approches have been proposed, kinematic demonstrations (like early LfD literature) [4], and Programming by demonstrations (PbD) [4]. Once the emonstratins are available, then various methods can be applied to model, understand, and enhance the data. These include but are not limited to, locally weighted regression (LWR) and dynamic movement primitives (DMP). The former are used to establish baselines for further learning and the latter are used to transform the systems into desired attractors.

IL path-planning approaches are normally classified according to the planning environment.

Free space path planning.: This class of IL problems, as the name suggests, focus on obstacle free trajectory planning. Hidden Markov models (HMMs) are frequently used as a probabilistic framework for modelling human demonstrations and motion generation for the desired robot. The framework requires description of nodes, transition probabilities, the probability density function and initial node distribution. HMM is more famous in human motion modelling and is also shown to inspire dual-arm humanoid robotic manipulation tasks in free space. Another common mathematical framework for imitation learning is the Gaussian mixture model (GMM). It projects the signals that are projected from motion data by principal component analysis (PCA) after data acquisition.

Constrained space path planning.: The contact between the robot and the environment constrains the planned trajectory, so imitation learning equipped with active obstacle avoidance will substantially expand the application field. Dynamic movement primitives (DMPs) framework gives a way to learn various goal-directed movement skills in high-dimensional continuous state-action spaces directly from human demonstration. DMP-based methods focus on mimicking trajectory rather than guiding the optimal motion direction in real time.

Inverse Reinforcement Learning for robot path planning

Inverse reinforcement learning (IRL) was earlier classified as a sub-division of reinforcement learning, but recently there has been an inclination of it being part of imitation learning framework. The IRL problem [8], is to identify a reward function for which the expert behavior is optimal and can be done via maximum margin IRL methods. The reward ambiguity problem was eliminated by introducing the maximum entropy principle [9]. The reward function is learned through maximizing the posterior probability of observing expert trajectories.

Generative Adversarial Imitation Learning

Recently, Generative adversarial imitation learning (GAIL) directly learns a policy from expert demonstrations while requiring neither the reward design in RL nor the expensive RL process in the inner loop of IRL. GAIL establishes an analogy between imitation learning and generative adversarial networks (GANs) [10].

LfD in the wild

LfD has been successful in a variety of challenges but usually depends on either manual demonstrations or specific sensor setups. However, it hasn't been widely used to take advantage of abundant "in the wild" demonstrations, such as those captured by already-existing sensors for different purposes, like traffic cameras recording natural behaviors of cars, bikes, and pedestrians. A novel method called "video to behavior" (ViBe) [11], which learns behavior models from unannotated raw video footage of traffic scenes taken from a single, standard-resolution camera that is initially un-calibrated. This method involves calibrating the camera, identifying and tracking objects over time, and using their movement patterns for LfD, thereby creating models of realistic behavior. ViBe is effective and has ability to learn from raw videos alone, without needing extra expert input.

APPENDIX 2: DEEP LEARNING TECHNIQUES FOR AUTONOMOUS CONTROL

The capability to execute parallel computations has propelled deep learning, enhancing its ability to analyze and resolve intricate challenges. This progression has accelerated research across various disciplines, broadening the scope and efficiency of deep learning solutions in diverse areas.

In the realm of autonomous vehicles, a critical aspect is the interpretation and prediction of the surrounding environment, including other vehicles and pedestrians. This section will delve into the influence of deep learning on various aspects of self-driving car technology. These vehicles typically incorporate five essential modules [12]:

Perception

Mirroring the human visual system, the perception module is tasked with sensing and recognizing environmental elements, such as traffic signs, pedestrians, and road markings. This module is fundamental, impacting other system components. Precision in perception is vital for the overall effectiveness of autonomous systems.

Traditional methods involved manually crafting features to highlight critical areas in sensor data, followed by applying machine learning techniques. These custom features required extensive fine-tuning and often lacked generalizability, especially under varying conditions like different lighting. Deep learning, however, has set new benchmarks in these tasks.

Convolutional Neural Networks (CNNs) have notably surpassed conventional methods in computer vision tasks such as object detection, classification, and segmentation [13]. CNNs, designed to process visual information, introduced the concept of convolutions, applying adjustable filters over image segments to create feature maps. These networks contrast with fully connected neural networks, as they capture local image features while requiring less computational power.

CNNs typically combine convolutional layers with nonlinear activation functions (like ReLU) and pooling layers to reduce spatial dimensions while retaining key features. The architecture progressively captures complex patterns, starting from basic shapes to more detailed representations.

Advanced CNN variants, such as Fast RCNN, have significantly enhanced object detection accuracy. They incorporate a Region Proposal Network (RPN) to share features with classification networks for effective object identification [14]. However, traditional CNNs struggle with temporal features in image sequences, an area where architectures like Transformers have shown promise, often using CNN layers as a foundation.

Prediction

This module anticipates future scenarios based on current environmental observations, crucial for predicting the behavior of other vehicles and pedestrians. Unlike CNNs that focus on single frames, many tasks require analyzing sequences to predict future actions.

Long Short-Term Memory (LSTM) networks, adept at processing sequential data, have become integral in these applications. LSTMs, with their unique gate mechanisms, can retain or discard information based on relevance, enabling them to predict future events effectively. They can be layered to enhance feature extraction from sequences, although they may struggle with long-term dependencies and computational complexity.

Upon detecting objects in the perception stage, LSTMs can process these data to forecast vehicle and pedestrian movements, crucial for anticipating turns, speed changes, or crossings. However, despite their proficiency in handling short-term dependencies, LSTMs are less effective with longer sequences, a limitation addressed by other architectures.

Localization and Mapping

Essential for navigation, localization ascertains the vehicle's position and orientation. This task involves processing vast sensor data, and long-term sequence dependencies are crucial for accurate pose estimation. Here, Transformers have shown potential in understanding long-range sequences, offering efficient and accurate pose predictions. [15]

Initially developed for natural language processing, Transformers are now applied in various fields. Their attention mechanisms enable selective focus on different parts of input sequences, facilitating comprehensive understanding of input relationships. This approach, though computationally intensive, is feasible due to parallel processing capabilities.

Transformers have been integrated into existing architectures, like replacing LSTM in odometry estimation models, to harness their long-term dependency understanding. Their efficacy in processing sequential images for pose estimation has been demonstrated, surpassing traditional methods [16].

Mapping creates a structured environmental representation, aiding in efficient path planning and obstacle avoidance. Addressing mapping challenges, such as sensor data gaps, has led to innovations like PoinTr, which reconstructs point clouds with geometrically-aware attention mechanisms [17].

Planning

With environmental awareness, planning algorithms determine optimal trajectories. They integrate inputs from mapping and prediction, employing traditional algorithms and newer methods like Reinforcement Learning (RL). RL trains agents to maximize rewards through exploration and refinement, proving effective in complex, dynamic environments.

Control

Comprising motion planning and actuation , this module generates detailed plans and executes control commands. Deep learning can directly learn actuator responses, offering advantages over traditional control algorithms [18].

In conclusion, while deep learning has greatly advanced autonomous driving technology, challenges like data dependency and real-time operation persist. Continued research in this field promises to push the boundaries of autonomy further.