Navigating the Future: an approach of autonomous indoor vehicles

Chandu Bhairapu, Christopher Neeb, Fatima Mohamed, Julian Tilly, Indrasena Reddy Kachana, Mahesh Saravanan, Phuoc Nguyen Pham, Arndt Balzer

Technical University of Applied Sciences Würzburg-Schweinfurt
Faculty of Computer Science and Business Information Systems

Abstract

Reinforcement learning (RL) has proven its good performance in solving challenging decision-making problems. Therefore, RL can be a promising solution for autonomous car to deal with complex driving scenarios. In this project, we explored the ability of RL in driving an indoor car autonomously. We also utilized SLAM for environment mapping and Wi-Fi technology for localization. The car was built using Waveshare JetRacer Kit, Nvidia Jetson Nano board, RPLidar A3, and Intel D455 camera. Data collected from the Lidar is considered the agent's state. The data is fed to the RL model to generate the best action the car should take in each state. The RL model is trained with over 200 episodes in a simulation environment. The experiment showed that the model can perform very well in simulation. In real-world environment, although the model lacks the ability to drive the car smoothly along a predefined path, the car is still able to avoid obstacles and walls.

Keywords— Autonomous vehicles, SLAM, indoor localization, reinforcement learning

1 Introduction

For several years, autonomous vehicles, also known as self-driving cars, have been an active area of scientific research. Every year, approximately 1.3 million people worldwide die as a consequence of traffic accidents [1]. The development of autonomous vehicles has the potential to revolutionize the entire transportation system and increase safety on the roads. The goal of this research sector is to develop vehicles that can drive autonomously from a starting point to a destination without human intervention. Numerous studies and experiments have already been carried out in this area, making an important contribution to the development of autonomous vehicles [2]. The Society of Automotive Engineers (SAE) has categorized autonomous driving into six levels. The lowest category, level 0, describes the state in which there is no automation. The top category, level 5, is reached when driving is fully automated. Many commercial cars nowadays reach level 2 in this classification [3].

Despite the progress made in the field of autonomous vehicles, the need for further research and development remains as high as it has always been. There are still multiple challenges to overcome, associated with autonomous driving. One of these challenges is developing vehicles, methods and techniques that can be used effectively in complex indoor environments, for example, a campus. Companies such as Waymo and Cruise have already driven several 100,000 km autonomously [4]. However, this experience only applies to outdoor environments. The majority of corporate research and development, has focused on the outdoor environments, because economically, the most benefit is available there. Indoor environments are a niche area of research.

The motivation for using automated driving in complex indoor environments is particularly high in manufacturing and distribution enterprises. Many use cases involve the transport of materials, not people. The purposeful transport of goods and commodities takes a lot of time and resources. Automation of logistics would increase productivity and reduce costs.

The development of autonomous vehicles for an indoor environment, such as a campus, is expected to contribute to filling the research gap. The system consists of a JetRacer Pro equipped with a propulsion control system and a Jetson computer for data processing. The vehicle was also enhanced with an RP LIDAR and an Intel Real Sense camera. Our assembled car is shown in figure 1. These sensors allow the vehicle to collect data about its surroundings. This data is used to create a map of the environment and to locate the vehicle within the map. The vehicle uses this information to plan the best route to its destination using reinforcement learning. Safety is an extremely important aspect that must be taken into account. The vehicle should ensure the safety of people or objects in its vicinity. Therefore, the autonomous vehicle is equipped with an object detection and avoidance system that allows it to detect and avoid any objects that are in its immediate surroundings. This work fills the niche by offering a unique prototype for the indoor autonomous driving challenge and providing new insights into the field of autonomous vehicles. The prototype is adapted to the SHL campus and reaches level 3, of the SAE classification.

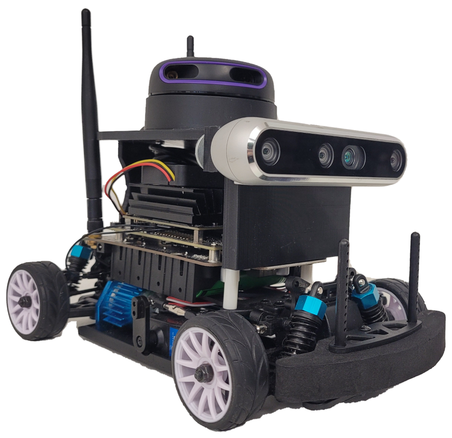


Figure 1: Autonomous car with the JetRacer Pro platform, advanced sensors RP LIDAR and Intel Real Sense camera

2 The car's hardware

Before the work on algorithms can begin, the first topic, that needs to be discussed is hardware. Obviously, a carrier unit is needed, which can drive indoors, carry all sensors needed onboard and which is capable of computing and/or communicating with algorithms to make it drive the way it is supposed to. Beside of that, we need to have a look at the question, what sensors will be needed to fulfill the task efficiently and what software architecture can be used to combine all this. Since factors as time, money and know-how are limited in our project, we chose to use the robot operating software (ROS, 3.1) to handle our codes as well as the internal hardware communication. With this decision in place, the following hardware choices have been made.

2.1 Nvidia Jetson Nano

First of all we chose to use a mobile computing unit, which can do both. Efficiently compute algorithms on the car itself as well as granting remote access and data transfer, so that the computation of more complex algorithms can be outsourced if needed. The Jetson Nano by Nvidia is a highly compact platform with a lot of computational resources, which makes it a great tool for autonomous driving. It combines CPU and GPU processors and supports a wide range of sensors. In addition, a custom Linux Ubuntu system is provided as software, so that the usage of ROS is covered as well.

In detail, the Jetson Nano Dev Kit B01 is based on the same architecture as more powerful Jetson platforms like the Jetson Xavier AGX, which are widely used in robotics and in the autonomous driving vehicle industry. A quad-core ARM A57 CPU with 1.43 GHz and 4 gigabytes of DDR4 RAM are combined with 128-core Maxwell GPU [5]. Especially the GPU lets the Jetson Nano be more useful than a Rasberry Pi and it provides the parallel processing power needed to handle large amounts of data created by sensors such as lidars or cameras. In addition, the Jetson nano provides multiple interfaces for connecting a wide range of sensors and peripherals, for example two CSI slots for cameras and digital pins for optional sensors like super sonic [5].

2.2 Waveshare JetRacer Pro

The JetRacer Pro by Waveshare is an autonomous racing car kit that is designed to be used with the Nvidia Jetson Nano (see 2.1) and it includes a battery pack, a power management system, a motor controller for steering and throttle, four wheels and tires embedded in an all-wheel undercarriage, a simple camera as well as a small display. The special Ubuntu version for this car includes all important software packages and example code, with which the car can be trained via supervised learning to follow a line autonomously. Some adjustments had to be made from our side, since we do not simply follow some tutorial. To fit in a lidar and a depth camera, we removed the included camera and its holder. Instead a custom 3D-printed rig has been installed, which offers mounting points for a camera and a lidar while offering storage for cables and keeping the Jetson Nano´s cooling intact.

2.3 Intel D455 camera

The Intel RealSense D455 is a depth camera that uses stereo vision to capture a three-dimensional image of the world. By combining two infrared cameras with a colour camera, not only pure depth perception is possible but the computation of coloured point clouds as well. The depth perception can be used for close range collision avoidance up to a distance of 6 meters while the coloured point clouds open up the possibility of generating coloured 3D-maps. The infrared cameras have a global shutter, which is special in comparison to other systems which usually have rolling shutters and therefore tend to miscalculate some distances while in motion. Also build in is an inertial measurement unit (IMU), a combination of a three-axis gyroscope and accelerometer which detects changes in position and rotation of the camera in dependence of time. IMU´s are frequently used for SLAM-algorithms (see 3). The calculation of the depth image stream with up to 90 frames per second and a resolution of 1280 to 720 pixel takes place onboard ([6]). This way some computational resources of the Jetson Nano (see 2.1) can be saved other uses. Additionally, the manufacturers' software offers full compatibility with Python, our mainly used programming language, as well as an full inclusion into ROS.

2.4 Slamtec RPLidar A3

The RPLidar A3M1 is a low-cost laser rangefinder device used in robotics and autonomous driving. Commonly used in devices like indoor vacuum cleaner robots, lidar sensors stand out do to there high accuracy and 360 degree measurement angle. The RPLidar used in this project is lightweight and offers a maximum range of 25 meters with 16000 samples per second ([7]). It comes with a software embedding for ROS.

3 SLAM

Autonomous Robots can move through both indoor and outdoor environments without running into any obstructions. A technique called simultaneous localization and mapping is used to create and update a map as well as to determine the location of the robot. Particle filter, Extended Kalman filter, FastSLAM, covariance intersection, and Graph based SLAM are a few of the algorithms used to solve it. Mapping, sensing, kinematic modelling, multiple objects, multiple cameras, moving objects, loop closure, exploration, and complexity are all components of the SLAM algorithm. A range measurement device that is used to observe the environment is the main component of SLAM. The range measurement tool is dependent on a number of factors. The robot uses sensors and measuring equipment to use landmarks to determine its location. The robot extracts the input and recognizes the environment when it detects a landmark. There are several different kinds of SLAM algorithms, including CoreSLAM, Gmapping, KartoSLAM, Lago SLAM, and HectorSLAM. We used ROS (Robot Operating System) to implement SLAM.

3.1 ROS

ROS (Robot Operating System) is an open-source software framework for building and controlling robots. It offers a collection of libraries and tools for creating robot software, including communication, simulation, and visualization tools.

ROS is used for SLAM (Simultaneous Localization and Mapping) in several ways. ROS offers a set of communication tools to facilitate the exchange of information between the various components of a SLAM system, such as laser rangefinder readings and map updates. For testing and debugging SLAM algorithms, ROS offers tools for simulating robots and environments. In order to comprehend and debug the behavior of the system, ROS offers tools for visualizing maps(Rviz), robot poses, and other data produced by a SLAM system. ROS provides a large library of open-source packages that can be used for building SLAM systems, including packages for sensor drivers, mapping algorithms, and navigation.

We developed the communication between ROS and a LIDAR sensor. Created a ROS node to handle communication between the LIDAR driver and the rest of the ROS system. Made the node subscribe to the data produced by the LIDAR and republish it as a ROS topic. The node will receive the LIDAR data as messages, which we can then process or visualize as needed using Rviz tool.

3.2 RViz

The Robot Operating System (ROS) has a 3D visualization tool called RViz (ROS Visualization) that enables users to view and interact with robotic data. It can show reference frames, robot models, sensor data (including lidar data), and more.

Lidar data can be seen in Rviz by launching it and adding LaserScan display in the RViz visualization window and by selecting the appropriate topic that contains the lidar data in the "Topic" field.

3.3 Hector SLAM

The SLAM algorithm used in this study is HectorSLAM. In particular, Hector SLAM is based on the Extended Kalman Filter-based SLAM (EKF-based SLAM) algorithm. Based on laser rangefinder measurements, the EKF algorithm is used to estimate the robot's pose and the environment's map. The algorithm is based on a mathematical framework called a Kalman filter, which is used to estimate a system's state from noisy measurements.

Hector SLAM is a ROS package and is installed using the ROS package manager. We made Hector SLAM node subscribe to a ROS topic to which the LIDAR is publishing data and started Hector SLAM node by launching a ROS launch file. The launch file specifies the parameters for the algorithm, such as the type of LIDAR sensor being used, the frequency of the LIDAR scans, and the topic names for the LIDAR data and the generated map. Now the vehicle is made to move to collect the LIDAR data and send it to Hector SLAM node. The Hector SLAM node publish the generated map to a ROS topic, which can be visualized using a ROS tool such as Rviz. The map is saved for later use by exporting it as a file in a common format, such as a .yaml or .pgm file. Hector Mapping is a part of Hector SLAM which has this key capability of Mapping.

4 Mapping

In this section, mapping of an unknown indoor environment is explained in more detail. The legitimate question arises, why do we need a map? An unknown environment is a big problem for a human, as well as for an autonomous vehicle. The autonomous vehicle may not be able to detect and react to unexpected obstacles or environmental conditions, which could lead to accidents or malfunctions. In addition, the autonomous vehicle will be unable to plan its route safely and efficiently, resulting in delays as well as increased energy consumption. Therefore, the creation of an accurate and detailed interior map is essential for the successful operation of an autonomous vehicle. The map serves as a reference for the localization and navigation algorithms and enable the autonomous vehicle to understand its position and plan its movements within the environment to efficiently perform the tasks at hand [8].

An autonomous vehicle needs a large amount of data to understand its environment and to react to it, if this is not the case it leads to big problems. To capture the environment, we use a LIDAR sensor mentioned earlier. Only the sensor data of the LIDAR is used, no Inertial Measurement Unit (IMU) or other sensors. The sensor data must be processed after acquisition to generate a map. For this we use the tool called "Hector map". Hector map uses grid mapping as a mapping method and thus creates a 2D map of the environment. The indoor environment can be mapped in real time. For our use case, the system is accurate enough and no loop closure procedures arises [9]. Because the world is complex, we use approximations that require certain assumptions. One assumption we make, is a discretization of the environment into independent cells. In this way we obtain a cell structure of the environment. This has advantages for our further steps.

We estimate the state of each cell using a binary Bayes filter. Each cell is a binary random variable estimate whether the cell is occupied or free. The map m is created from the measured sensor data z and the positions of the car x. Calculations are performed in log odds notation to improve efficiency. The log odds notation calculates the logarithm of the ratio of the probabilities. This ratio can also be expressed as a sum. Formula 1 shows the occupancy mapping in log odds notation. The complete derivation and further details can be found in the book "Probabilistic Robotics" [10].

$$l\left(m_i \mid z_{1:t}, x_{1:t}\right) = \underbrace{l\left(m_i \mid z_t, x_t\right)}_{\text{inverse sensor model}} + \underbrace{l\left(m_i \mid z_{1:t-1}, x_{1:t-1}\right)}_{\text{recursive term}} - \underbrace{l\left(m_i\right)}_{\text{prior}} \tag{1}$$

For the creation of the map, we needed several attempts to find the right settings in Hector map that we could achieve the best quality. For this we took the car in our hands and walked from the starting point (bottom right) to the end point (top left). In our experience, we walked with the car because it is a smoother movement than driving the car remotely. Due to the steady movement, the quality of the map was better and contained fewer errors. However, we made the assumption that the map would look the same at 0.2 m height when the car is moving as it does at 1.2 m height when we are carrying it. This assumption is acceptable for our map. The map was created in real time while the car was moving. The final result is shown in the Fig. 2.

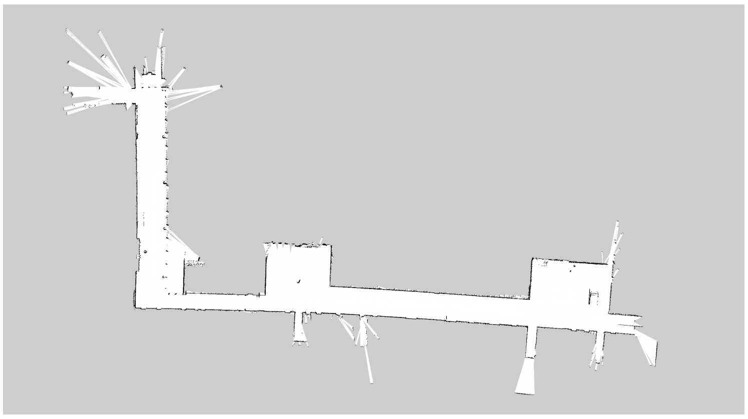


Figure 2: The map shows the first basement of the SHL campus. The horizontal corridor is part of the Institute Building and the vertical corridor belongs to the Lecture Hall Building.

The black dots or lines represent obstacles and walls. The outer walls of the campus are clearly visible. The free space in between, shown in white, are the corridors and rooms inside the building. The unknown area is shown in gray, which is the outside of the basement in our case. There are fragments where the map is not completely accurate. This is the case where, for example, doors of rooms were open or corridors were not fully explored. In summary, we can say that the map corresponds to our imagination and represents the reality well.

Once the map is created, it can also be used for training in a simulation. In this way, possible problems can be identified and corrected before the vehicle is used in a real environment. In the simulation, the map was revised and the fragments were removed to use an ideal map. In addition, the map can also be applied to validate the performance of various algorithms used by the autonomous vehicle and test diverse capability in a simulation. Thus, various scenarios such as unexpected obstacles or changes in the environment can be specifically tested, evaluated, and improved without causing a hazard in the real environment.

In summary, the creation of an accurate and detailed interior map is a crucial step in the development of autonomous vehicles and has a critical impact on ensuring and operating them safely and efficiently.

5 Localization

Since GPS is widely used for localization, but it is not used for localization in an indoor environment due to the following reasons. First, GPS indoor signals are weak because of the walls, roofs, and indoor objects interfere with the signal[11]. Second, if GPS is used in an indoor environment, it will give low-accuracy position information. Third, the GPS has a slow update rate, and since indoor environments require real-time localization the GPS is not suitable for usage. Therefore, in this project Wi-Fi based localization is utilized to provide the autonomous car with reliable and real-time location information in an indoor environment.

5.1 Indoor Localization

It is very important for the autonomous car to navigate indoors correctly, therefore a localization system is needed. Indoor Localization is used to determine the position and orientation of the autonomous vehicle to navigate in a building or an indoor environment. Localization has many techniques including:

1. Visual-based systems: which use computer vision algorithms to find out the exact location of the mobile robot given the visual information in a building or an indoor environment [12].

2. Lidar-based systems: Lidar stands for Light Detection and Ranging and it utilizes laser to measure distances for mapping and providing position information for the mobile robot.

3. Inertial Navigation Systems (INS): these are usually accelerometers and gyroscopes, and are also called inertial sensors. They measure acceleration and angular velocity and are used for indoor localization and orientation of mobile robots [13].

4. Wi-Fi-based positioning: which uses the Wi-Fi's signal strength to find the location of a mobile robot in an indoor environment.

In this project, Wi-Fi-based positioning is used and is detailed in the following section.

5.2 Wi-Fi based Localization

To localize the autonomous vehicle and orientate it to navigate along a predefined efficiently and safely a Wi-fi positioning system is used. Wi-Fi stands for wireless fidelity, it is one of the wireless network protocols of IEEE 802.11 standards [14], and it is commonly used for Internet access through wireless routers. The Wi-Fi's signal strength is usually measured in decibels (dBm), a higher decibel number indicates a stronger signal [15].

RSSI (Received Signal Strength Indicator) is a widely used measurement, but it differs from one adaptor to another, and it is not standardized. Some adapters use a scale of 0-60, and others 0-255. Basically, the most efficient way to measure signal strength is with dBm, which stands for decibels relative to a milliwatt. RSSI is usually converted to dBm to make it readable for humans.

Formula to convert dDm to milliwatts :

$$PmW = 10^{(P_{dBm} \div 10)}$$

Formula to convert milliwatts to dDm :

$$P_{dBm} = 10 * log_{10}^{(P_{mW})}$$

Formula to calculate the percentage of signal strength from dDm :

$$percentage = 100x(1–(P_{dBm_max}–P_{dBm}) \div (P_{dBm_max}–P_{dBm_min}))$$

Wi-Fi-based positioning (WPS) uses the Wi-Fi signal's strength of the Wi-Fi access points in the building to give the orientation and position of the autonomous vehicle. Wi-Fi localization was chosen for indoor localization because of its high accuracy, low cost, fast update rate, and its availability in the building.

5.3 WPS Implementation

This Wi-fi positioning (WPS) algorithm that was created in this project depends mainly on the Wi-Fi access points' signal strength that was already installed in the building. In the building, every area has a known Wi-Fi access point.

Therefore, we implemented in python a real-time Wi-Fi scanner that scans Wi-Fi access points and their relative signal strength. When a specific assigned Wi-Fi access point is found that indicates that the autonomous car is in the same position where the access point is located in the building. Moreover, the Wi-Fi access points' signal strength is compared to provide the orientation or the direction of the autonomous car's movement, and a checkpoint that the car is navigating in the correct direction.

Despite its high accuracy, some uncertainties can occur due to some distortions of the Wi-Fi signals by access points misplacement, doors, or indoor objects.

6 Object detection

Object detection is a computer vision technique for detecting instances of meaningful things in digital photos or videos. With the advancement of deep learning, it has become a significant task in the field of computer vision, with several real-world applications such as self-driving cars, security and surveillance systems, and medical imaging.

Object detection algorithms often use machine learning or deep learning to deliver relevant findings. These algorithms learn to identify items by examining training data and recognizing patterns or features that are unique to individual objects. They then utilize this information to recognize instances of such things in new photos or videos. Object detection algorithms are classified into two categories: Image-level detection and Instance-level detection.

Image-level detection methods classify an entire image into a fixed set of predefined classes without identifying the location of the objects in the image. In contrast, instance-level detection techniques discover and classify numerous instances of things in an image and produce a bounding box around each object. Among the most widely used object detection methods are: Faster R-CNN, You Only Look Once (YOLO), Single Shot MultiBox Detector (SSD) and RetinaNet.

6.1 Mask R-CNN

Mask R-CNN is a computer vision architecture based on deep learning that is commonly used for segmentation. It is an extension of the popular Faster R-CNN network for object detection.

Mask R-CNN is built to handle two tasks at once: object detection and semantic segmentation. The task of recognizing and localizing things of interest in an image is known as object detection. The task of classifying every pixel in a picture into a preset set of categories is known as semantic segmentation. Mask R-CNN expands Faster R-CNN by including a network branch that generates binary masks for each instance of an item in an image.

Mask R-architecture CNN's is made up of three major components: a backbone network, a Region Proposal Network (RPN), and a mask branch. The backbone network captures information from the input image and sends them to the RPN, which provides a set of region suggestions. The RPN-identified regions of interest are then sent to the mask branch, which builds binary masks for each instance of an object in the image.

6.2 Steps to be followed

Follow these steps to detect objects using Mask R-CNN on an Nvidia Jetson Nano:

1. Install the Jetson Nano: Install the necessary software and libraries, such as OpenCV, and PyRealSense2, on Nvidia Jetson Nano. For this, we can use the Nvidia Jetson Nano Developer Kit setup tutorial.

2. TensorFlow and Keras must be installed: TensorFlow and Keras are deep learning frameworks that enable Mask R-CNN. These can be installed with the pip package manager.

3. Downloaded a Mask R-CNN model that has already been trained: we can either train your own Mask R-CNN model on our own dataset or utilize an online pre-trained model.

4. Load the pre-trained model onto Jetson Nano by doing the following: Using the TensorFlow and Keras libraries, loaded the pre-trained model onto Jetson Nano.

5. Use PyRealSense2 to capture depth data: The PyRealSense2 library can be used to capture depth data from a Realsense depth camera connected to the Jetson Nano.

6. Recognize objects: Using the imported Mask R-CNN model, we can now detect objects in depth images produced with PyRealSense2. The model's predict() method can be used to recognize objects and produce instance masks.

7. Show the results: Using OpenCV, we can see the results of object detection, including bounding boxes and instance masks. We can visualize the discovered objects by superimposing the findings on a depth image or video.

6.3 Equations

Among the important equations utilized in the procedure are:

1. Region Proposal: The region proposal step generates region proposals based on the input image using a convolutional neural network (CNN). The equation used to calculate region proposals is as follows:

$$R_{proposal} = f(I, w)$$

where I is the input image, w denotes the CNN parameters, and f denotes the activation function used to generate the region recommendations.

2. Object Classification: The object classification step classifies the objects in the image using the region proposals obtained in the previous stage. The object classification equation is as follows:

$$C_{obj} = g(R_{proposal}, w)$$

where $R_{proposal}$ is the region proposal, w are the CNN parameters, and g is the object classification activation function.

3. Object Segmentation: The object segmentation stage generates a binary mask for each object in the image using the region proposals and object categorization.

$$S_{obj} = h(R_{proposal}, C_{obj}, w)$$

The above equation used for object segmentation, where $R_{proposal}$ denotes the region proposal, C_{obj} denotes the object classification, w denotes the CNN parameters, and h is the activation function used to construct the binary mask.

7 Reinforcement learning

Reinforcement Learning (RL) is a significant machine learning algorithm that differs from other learning algorithms in that it does not require prior training data. RL is utilized for sequential decision-making tasks and is formulated as a Markov Decision Process (MDP). In MDP, an agent interacts with an environment by following a policy, receiving numerical rewards, training the policy and repeating this process until the policy converges to the optimal policy, which maximizes the cumulative reward. The fundamental components of an MDP include the environment, agent, policy, state, action space, and reward function, as illustrated in Fig. 3 [16].

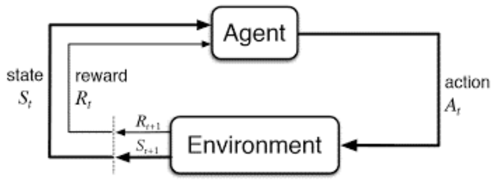
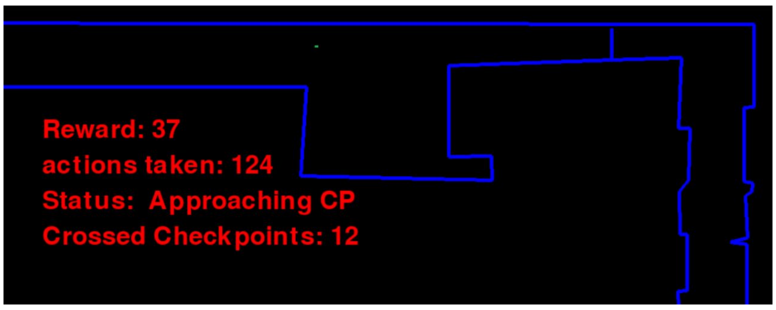


Figure 3: Markov decision process of Reinforcement Learning

7.1 Reinforcement learning in simulation

Using simulation for training reinforcement learning model is essentially efficient in this project. Simulation enables avoiding crashes on the car and reduces training time. Additionally, the car can explore more scenarios in simulation than in hand-designed situations during training. Therefore, our driving model is trained only in simulation with virtual Lidar data.

Our simulator is developed by using a python library "Pygame", which is an open-source game environment. Pygame is extremely useful for training and testing reinforcement learning models with simple visualization (Fig. 4).



Reward: 37
actions taken: 124
Status: Approaching CP
Crossed Checkpoints: 12

Figure 4: Visualization of car model during testing phase using Pygame library

7.1.1 Environment

The environment in the simulator is the virtual space in which a agent(car) operates, which includes walls, obstacles, and other features. This representation is created by combining the floor plan of the building and a 2D map generated from Lidar data using simultaneous localization and mapping (SLAM). The floor plan provides the correct geometric proportions, while the Lidar map ensures a realistic representation of the real-world scenario. The resulting image was scaled to a resolution of 5 cm per pixel using Adobe Photoshop. Figure 4 illustrates a part of merged map of the environment. The starting and ending coordinates of each line in the merged picture are extracted using OpenCV Hough line transform function.

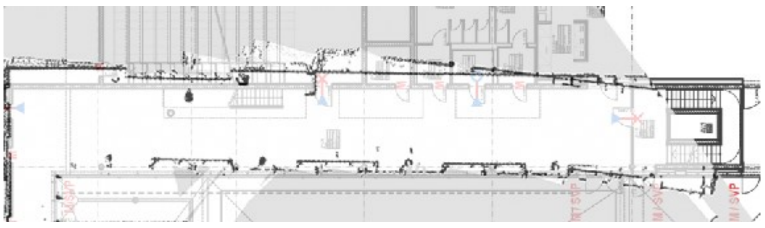


Figure 5: Merged map

The agent in our simulation is a car that moves within the environment, which is defined by lines marking the boundaries of the car. The car is 30 cm by 15 cm in dimension and is scaled to match the resolution of the environment map. The car's state changes in response to action commands, which are realized by updating the pixel values of its corners. The car can move a maximum of 10 cm (2 pixels) in a single time step and can turn up to 10° in either direction. To simulate the effects of real-world uncertainty, Gaussian noise with a standard deviation of 2 pixels (10 cm) is added to the car's position and a noise of 5° is added to its orientation.

The state space represents the car's position and orientation within the environment. This is calculated by projecting 360 rays at different angles around the car and calculating the distances between the car and the points of intersection of the rays and walls or obstacles. This calculation was performed using Cramer's rule. The resulting distance measurements form a geometrical representation of the car's state within the environment. Let R_i and W_i be the equation of line denoting i^{th} Ray and i^{th} wall respectively.

$$\begin{array}{ccccccc} R_i & = & ax & + & by & + & c \\ W_i & = & dx & + & ey & + & f \end{array}$$

The point of intersection (x_i, y_i) can be calculated by equating both the equation.

$$(x_i \ , \ y_i) \ = \ \left(\frac{bf \ - \ ec}{ae \ - \ db} \ \ \frac{cd - fa}{ae - db} \right)$$

The distance between closest intersection point and the car is passed as input to corresponding node. This process is repeated for all the rays and walls. Fig. 6 represents the rays and the intersection points with walls.

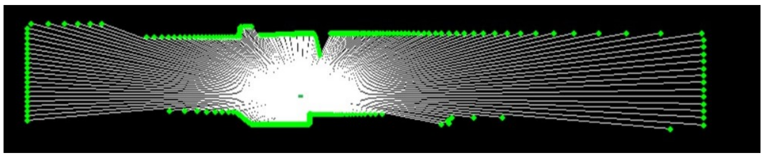


Figure 6: State representation.

The calculated distance data is subject to zero-centered Gaussian noise with a standard deviation of 10 cm (2 pixels) to simulate the real-world measurement uncertainty. Additionally, some random subsets of rays are omitted to further mimic the limitations of real-world sensors. These modifications add realism to the state space representation and help ensure the model's robustness to real-world conditions. The action space is the set of all possible actions set the car can execute which is given by Move Forward, Move Backward, Turn left, Turn right, Stay

7.1.2 Reward function

Reward function evaluate the action taken by the car and returns numerical rewards and penalties. The route followed by the car is divided into a series of consecutive checkpoints. The car receives a reward of +1 for each time step in which it moves closer to the next checkpoint, and -1 for each time step in which it moves away from the checkpoint. When the car successfully crosses a checkpoint, it is rewarded with +10, and when it crosses a checkpoint that it has already passed, it receives a penalty of -10. This reward structure provides incentives for the car to move towards the next checkpoint and avoid backtracking.

A region of 30 cm (6 pixels) from any wall is defined as a danger zone. If the car enters this region, it receives a penalty of -1 for each time step spent in the danger zone. Additionally, if any part of the car intersects with a wall, the car is immediately penalized with a penalty of -10 and that marks the end of an episode. These collisions are calculated by checking for existence of any intersection points between the edges of the car and all walls in the environment. These penalties serve to discourage the car from colliding with walls and from lingering in dangerous regions near walls.

7.2 Training

The policy must be constantly optimized using the knowledge acquired from experiences to maximize cumulative rewards and minimize penalties during simulations. Over time, the policy will improve and converge towards an optimal policy that guides the agent towards maximizing its rewards. The policy is updated after every action taken by the agent. Since the number of possible state spaces is vast and uncountable, the policy must approximate and adapt to similar state spaces.

7.2.1 Policy

The policy for the agent is defined by a simple neural network model consisting of three hidden dense layers. The input layer has 361 nodes, where the first 360 nodes receive the distance measurement at each angle around the car and the last node receives the car's speed (state space). The output layer has five nodes, each mapping to a different action in the action space. TensorFlow library is used to build, train and test the model.The model predicts the logits (Q-values) for each possible action. The car then executes the action with the highest predicted Q-value. In this way, the network guides the car's behavior by determining which action will result in the greatest reward based on the current state of the environment after training.

The epsilon-greedy approach is used to balance exploration and exploitation in the decision-making process of the agent. Exploration refers to the process of randomly selecting an action from the action space in order to gain knowledge about different states and actions taken from that stae. On the other hand, exploitation refers to the process of following the policy and maximizing the reward. The epsilon-greedy policy manages this trade-off by allowing the agent to initially explore more, and then gradually exploiting more as the agent's knowledge of the environment increases.

The degree of exploration versus exploitation is controlled by the exploration-exploitation ratio ϵ. When ϵ is higher, the agent explores more and when ϵ is lower, the agent exploits more. The value of ϵ decreases over time at a rate determined by the epsilon decay rate, ultimately approaching close to zero as the final episode is reached. This approach ensures that the agent gains a thorough understanding of the environment while maximizing its rewards.

Experience memory is a buffer which stores tuples containing history of each time step. The tuple B_t representing the time step t contains state of the car at time t (S_t), action taken by the car from S_t at time t (A_t), Reward received by the car for the action A_t at t (R_t) and next state of the car after executing A_t from state S_t at time t (S_{t+1}) [17].

$$B_t \quad = \quad (S_t, A_t, R_t, S_{t+1})$$

The buffer appends every entry at the end and removes the first entry once it reaches the storage threshold value of 1024 entries.

7.2.2 Bellman Optimality Equation

The training process in reinforcement learning is to approximate the policy to an optimal policy which yields maximum cumulative reward. An Optimal model q_* should follow Bellman's Optimality equation

$$q_*(s,a) \ = \ E\left[R_{t+1} \ + \gamma \cdot max \ q_*(s\prime, a\prime)\right]$$

Expected cumulative reward for an action 'a' taken from state 's' and following the optimal policy q_* thereafter should be sum of immediate reward R_{t+1} and maximum possible future discounted reward $(max \ q_*(s\prime, a\prime))$ when following policy q_*. γ is the discount factor where each consecutive expected reward is discounted by γ^t.

7.2.3 Policy and target network

The batch of data is sampled from the replay memory buffer for training. The data are shuffled in order to break the co-relation between the data. The current state (S_t) is fore-propagated into the policy network and it predicts the q values (logits) for each action. The next state (S_{t+1}) is passed into the new model called "Target network".

The target network is the clone of policy network and their weights are locked. The second pass of (S_{t+1}) to the new network is to calculate the target q-value from the bellman's optimality equation which require optimal q value ($max\ q_*(s\prime, a\prime)$). The target q value is calculated using the same equation [18].

$$q_*(s, a) \ = \ E\left[R_{t+1} \ + \gamma \cdot max\ q_*(s\prime, a\prime)\right]$$

The loss between the target q value and the predicted q value from the policy network is calculated using the Mean Square Error (MSE) loss function. The weights of the policy network are optimized using the Adam optimizer, an efficient gradient-based optimization algorithm. To further stabilize the training process, the weights of the target network are updated with the weights of the policy network every 25 episodes. This separation of the target and policy networks is necessary to prevent any instability that could occur if both the target and predicted values were calculated within the same network using the same weights.

8 Conclusion

The hardware was not perfect, the battery condition has a huge impact on the execution of the commands. A fuller battery means more precise movements. The wheels on the axis are not completely parallel. Due to the toe-in, the car does not drive straight even with 0° steering. This interference also affects the performance.

After some tests, the creation of the map works great. The result can be seen in figure 2. The map looks very much like the building map. Black objects were detected by slamming, but the technique of LIDAR fails with glass fronts. There it is necessary to use other sensors like the camera or ultrasonic sensors.

The results after 200 episodes showed that the trained model was able to successfully navigate through the environment in the simulator, reaching the destination while avoiding walls and recovering from unfavorable situations. The model was tested in various conditions, including adding more noise to the input data, to validate its performance. In most of the tests, the car can navigates smoothly in the simulation. However, in the real-world environment, the car can only pass a few tests.

To summarize, object detection is an essential and fast evolving field with numerous applications. With improvements in deep learning and computer vision, object detection is predicted to continue to play an important role in a variety of industries and contribute to the creation of new technologies.

References

[1] *Road traffic injuries.* 2022. URL: `https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries` (visited on 02/04/2023).

[2] Saeid Nahavandi et al. "A Comprehensive Review on Autonomous Navigation". In: *arXiv preprint arXiv:2212.12808* (2022).

[3] *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles.* 2021. URL: `https://www.sae.org/standards/content/j3016_202104/` (visited on 02/04/2023).

[4] David Fickling. *Self-Driving Cars Need to Slow Down After Uber Crash.* 2018. URL: `https://www.bloomberg.com/opinion/articles/2018-03-20/uber-crash-shows-need-for-collaboration-in-self-driving-cars` (visited on 02/04/2023).

[5] *Jetson Nano Developer Kit.* 2022. URL: `https://developer.nvidia.com/embedded/jetson-nano-developer-kit` (visited on 02/04/2023).

[6] *Intel D455 Depth Camera.* 2022. URL: `https://www.intelrealsense.com/depth-camera-d455/` (visited on 02/04/2023).

[7] *Slamtec RPLidar A3.* 2022. URL: `https://www.slamtec.com/en/Lidar/A3` (visited on 02/04/2023).

[8] Syahrul Fajar Andriawan Eka Wijaya et al. "Research Study of Occupancy Grid map Mapping Method on Hector SLAM Technique". In: *2019 International Electronics Symposium (IES).* 2019, pp. 238–241.

[9] Stefan Kohlbrecher et al. "A flexible and scalable SLAM system with full 3D motion estimation". In: *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics.* 2011, pp. 155–160.

[10] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics.* Cambridge, Mass.: MIT Press, 2005.

[11] Yuxiang Sun, Ming Liu, and Max Q.-H Meng. "WIFI signal strength-based robot indoor localization". In: *2014 IEEE International Conference on Information and Automation (ICIA)* (2014).

[12] Ming Liu et al. "The role of homing in visual topological navigation". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2012).

[13] Giovanni Fusco and James M. Coughlan. "Indoor localization using computer vision and visual-inertial odometry". In: *Lecture Notes in Computer Science* (2018), pp. 86–93.

[14] Jonathan Leary Pejman Roshan. *802.11 Wireless LAN Fundamentals.* Cisco Press, 2003.

[15] Yogita Chapre et al. "Received signal strength indicator and its analysis in a typical WLAN system (short paper)". In: *38th Annual IEEE Conference on Local Computer Networks.* 2013, pp. 304–307.

[16] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* Second. The MIT Press, 2018.

[17] Maxim Lapan. *Deep Reinforcement Learning Hands-On.* Packt Publishing, 2018.

[18] Richard S. Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Proceedings of the 12th International Conference on Neural Information Processing Systems.* MIT Press, 1999.

Chandu Bhairapu **Main author:** 3	Christopher Neeb **Main author:** 1, 4	Fatima Mohamed **Main author:** 5
Julian Tilly **Main author:** 2	Indrasena Reddy Kachana **Main author:** 6	Mahesh Saravanan **Main author:** 7
Phuoc Nguyen Pham **Main author:** Abstract	Group work **Contributing:** 8	