CAIRO Center for Artificial Intelligence and Robotics

thws Technical University of Applied Sciences Würzburg-Schweinfurt

Navigating the Future: an approach of autonomous indoor vehicles

Chandu Bhairapu, Christopher Neeb[1], Fatima Mohamed, Julian Tilly[2], Indrasena Reddy Kachana, Mahesh Saravanan, Phuoc Nguyen Pham

Technical University of Applied Sciences Würzburg-Schweinfurt (THWS)
Faculty of Computer Science and Business Information Systems
Würzburg, Germany

christopher.neeb@study.thws.de[1], *julian.tilly@study.thws.de*[2]

Abstract

In this project, we explored the ability of Reinforcement learning (RL) in driving an indoor car autonomously. 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. As hardware a model car eqipped with sensors and powerful computational unit has been used. We also utilized SLAM for environment mapping and a combination of lidar data and Wi-Fi technology for localization. The experiment showed that the model can perform very well in simulation. Although the model lacks the ability to drive the car as smoothly along a route, the car is still able to avoid obstacles and walls in an unknown real-world environment.

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.

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 RPLidar 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 RPLidar and Intel Real Sense camera

2 The car's architecture

Before the work on algorithms can begin, the topic of architecture have to be discussed first. 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, 2.4) 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 (see 2.4) 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 Sensors

Two main sensors have been used in this project. An Intel RealSense camera as close to mid range depth sensor and a lidar for more exact long range measurements.

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. 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.

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.

2.4 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. The fundamental concept of ROS is the communication of nodes. A ROS node can be any algorithm which sends or receives data. Sensors and their software drivers for example can be included into a node, which will publish their measurements in a predefined syntax, the messages. To make sure, that the published data finds its ways to the right receiver, the so called ROS master or ROS core is used to distribute that data. A node does not contact another node directly. Instead they inform the ROS core about their capability of sending or receiving data of a certain kind, the so called topics. Only if there is at least one sender (a ROS publisher) and one receiver (a ROS subscriber) on the same topic, the core connects those directly and data can be send. Alternatively, service nodes can be used, when specific calculations are needed only momentarily and not during the whole run. Instead of connecting two nodes permanently, a service will be called once by a special message and it will return its results only when called [8].

In addition to those communicational core capabilities of ROS there are includable packages, which expand the functional scope of actions. All kinds of software packages are available for free, for example sensor drivers,

example code for hardware or software tools such as path planing nodes for mobile robots. Beside of installable packages, ROS already included some of the most useful packages into their own framework. ROS provides a transformation tool, which allow the user to easily construct robot models with many movable parts, which do not act individually but depend on each other. The visualisation tool RViz allows to not only visualize such a model itself but their sensor data and environment as well and with Gazebo ROS even included their own simulation software. These software packages cover a lot of problems which commonly occur in the field of robotics, but their use is optional and can be replaced by own solutions. In this project, ROS enabled our teams to work independently on our tasks and merge it effortlessly afterwards.

3 SLAM

SLAM is short for simultaneous localization and mapping and describes a well known problem in robotics. For localization on its own a map is needed to point out where in it something (f.ex. a robot) can be found. In contrast, during the whole mapping process one needs to know the exact position and orientation said thing. The problem arises especially in robotics, where both techniques are needed at the same time, although each of these processes relies on data generated from the other one. There are many approaches of different kinds to solve the problem of SLAM with extended kalman filtering, particle filtering and graph based approaches being the most popular ones [9].

In this paper the open source tool Hector SLAM has been used which follows the extended kalman filter approach. The choice of Hector SLAM among the many available solutions results from two reasons. First of all, the vehicle described can only move in a plane environment, so only a 2D map is needed to compute its path. Although having sensors on board capable of creating a 3D map as well, 2D mapping is sufficient and easier to compute, so we chose to save computational capacities by using Hector SLAM. Secondly, most of the SLAM approaches use odometry data as mandatory input. Due to software limitations this was no option for our case. Hector SLAM is not only very efficient regarding computational resources. It publishes a 2D map and is one of the rare open source options which only needs lidar data without odometry [10].

3.1 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 autonomous vehicle will be unable to plan its route safely and efficiently, resulting in delays as well as increased energy consumption. Therefore, the availability 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 enables the autonomous vehicle to understand its position and plan its movements within the environment to efficiently perform the tasks at hand [11]. 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. For our use case, the system is accurate enough and no loop closure procedures are needed [12]. 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 [13].

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

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 (Fig. 2, bottom right) to the end point (Fig. 2, 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 figure 2.

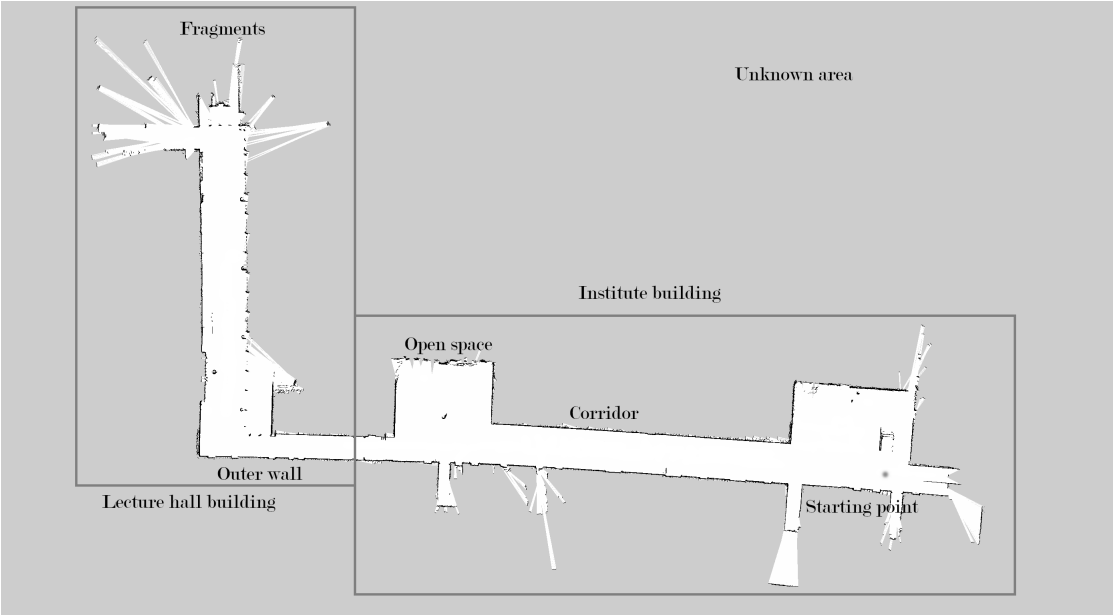

Figure 2: The map shows the first basement of the SHL campus. The horizontal corridor is part of the Institute Building and the vertical corridor belongs to the Lecture Hall Building. The black dots or lines represent obstacles and walls. The outer walls of the campus are clearly visible. The free space 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.

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.

3.2 Localization

Accurate navigation of autonomous vehicles in indoor environments is a crucial requirement for their effective functioning. Therefore, the implementation of a reliable localization system is essential. This system determines the position and orientation of the vehicle, enabling it to navigate in a building or indoor environment effectively.

Various techniques have been developed to achieve accurate localization in different environments. There are five common techniques in total, namely Global Positioning System (GPS), visual-based systems, lidar-based systems, inertial measurement units (IMUs), and Wi-Fi-based positioning. The GPS is a commonly used localization method, but its application in indoor environments is limited due to several reasons. First, GPS signals in indoor environments are weakened by the presence of walls, roofs and objects, which interfere with the signal [14]. Second, the use of GPS in indoor environments often results in low-accuracy position information. Visual-based systems employ computer vision algorithms to determine the precise location of a mobile robot based on visual information in an indoor environment or building [15]. Lidar-based systems, on the other hand, use lasers to measure distances and provide position information for the mobile robot. Inertial measurement units (IMUs), which typically include accelerometers and gyroscopes, are also used for indoor localization and orientation of mobile robots [16]. Finally, Wi-Fi-based positioning utilizes the strength of Wi-Fi signals to determine the location of a mobile robot in an indoor environment.

In this project, we relay on Wi-Fi based localization to provide reliable real-time location information for indoor autonomous vehicles. We proposes a Wi-Fi positioning system for localizing autonomous vehicles to navigate in

indoor environments. The system relies on the signal strength of pre-installed Wi-Fi access points to determine the vehicle's location. A real-time Wi-Fi scanner was implemented in Python to detect Wi-Fi access points and their relative signal strength. The scanner compares the signal strength of the Wi-Fi access points to determine the car's position and direction. The proposed algorithm provides medium accuracy, but uncertainties can arise due to environmental factors or objects. These uncertainties should be considered when using the algorithm for real-time positioning of the autonomous vehicle. Specifically, Wifi localization is used as a coarse localization method, which provides an accuracy of approximately 9 meters. To achieve fine localization, the existing lidar system is utilized. This enables precise measurement of the distance between the vehicle and surrounding objects, facilitating accurate determination of the vehicle's position.

4 Object detection

Object detection is a widely studied and rapidly evolving computer vision technique that allows the detection of meaningful objects in digital images or videos. It has numerous real-world applications, such as self-driving cars, security and surveillance systems, and medical imaging. With the recent advancement of deep learning techniques, object detection has become an essential task in the field of computer vision [17].

Object detection algorithms typically rely on machine learning or deep learning approaches to identify relevant objects in an image. These algorithms learn to recognize objects by analyzing large data sets of training images and identifying unique patterns or features that are characteristic of each object. They then use this learned information to detect instances of these objects in new images or videos.

There are two main categories of object detection methods: 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 objects in an image and produce a bounding box around each object.

Several state-of-the-art object detection methods have been proposed in recent years, including Faster R-CNN, You Only Look Once (YOLO), Single Shot MultiBox Detector (SSD), and RetinaNet. These methods utilize deep learning networks to detect objects accurately and efficiently [18]. For example, YOLO has achieved impressive performance on real-time object detection tasks [19], while RetinaNet has been shown to produce more accurate detections of small objects than previous methods [20].

Overall, object detection has proven to be a crucial tool in computer vision with many practical applications . Ongoing research efforts aim to improve the accuracy and efficiency of object detection algorithms, further expanding the range of applications for this important technology [21].

The object detection approach used in this paper is the 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.

5 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 Figure 3 [22].

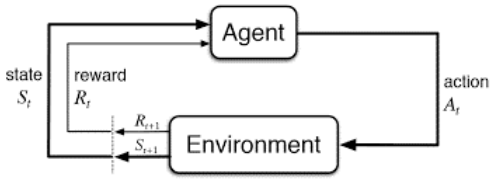

Figure 3: Markov decision process of Reinforcement Learning

5.1 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.

5.1.1 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 state. 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 the 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}) [23].

$$B_t = (S_t, A_t, R_t, S_{t+1}) \tag{2}$$

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

5.1.2 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

$$q_*(s,a) = E\left[R_{t+1} + \gamma \cdot \max_{a'} q_*(s',a')\right] \tag{3}$$

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.

5.1.3 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 corelation 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 [24].

$$q_*(s,a) = E\left[R_{t+1} + \gamma \cdot \max_{s',a'} q_*(s',a')\right] \tag{4}$$

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 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.

5.2 Reinforcement learning in simulation

Using simulation for training reinforcement learning model is essentially efficient in this project. Simulation enables the car to avoid collisions 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).

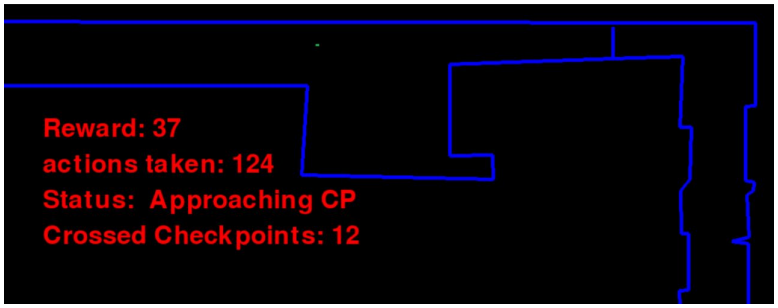

Figure 4: Pygame visualization of the buildings floor plan used for simulation

5.2.1 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 5 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.

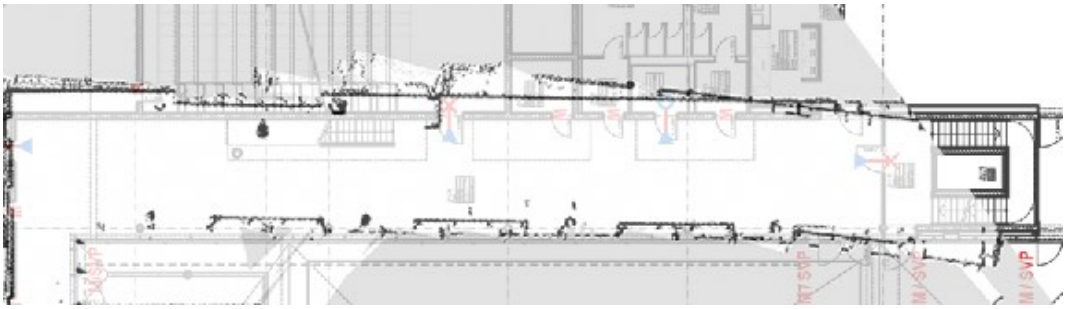

Figure 5: This figure shows the combined map created by combining lidar indoor mapping and the blueprint. This map enables them to provide better navigation and path planning for autonomous vehicles operating in this environment.

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. The parameters a, b, c, d, e, f have been determined by many experiments.

$$R_i = ax + by + c \tag{5}$$

$$W_i = dx + ey + f \tag{6}$$

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

$$(x_i, y_i) = \left(\frac{bf - ec}{ae - db}, \frac{cd - fa}{ae - db} \right) \tag{7}$$

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. Figure 6 represents the rays and the intersection points with walls.

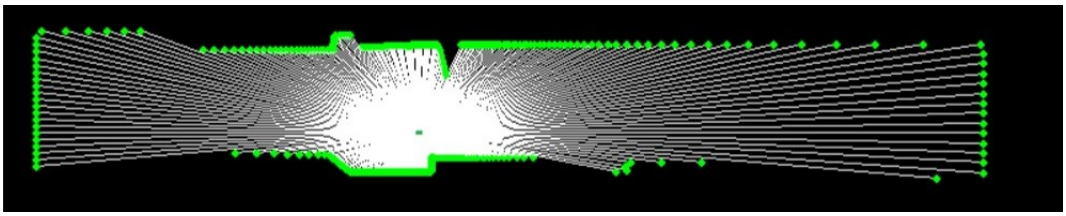

Figure 6: Visualization of the state representation during simulation. The lidar sensor emits laser beams in all directions around the car, which are represented in white. The green dots indicate the end of the beams where they collide with a wall or an obstacle.

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

5.2.2 Reward function

The reward function evaluates 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.

The car was not trained with additional obstacles, but learned to detect and avoid obstacles using lidar data. The vehicle was successfully tested in practice and avoided obstacles based on its skills acquired in the simulation environment. This was successful not only with obstacles encountered in training, such as walls, but also with unknown structures and people. The fact that both learned obstacles and new obstacles were successfully avoided illustrates the performance of the RL model applied.

6 Conclusion

In conclusion, the results of this study provide valuable insights into the feasibility and effectiveness of using RL in combination with SLAM and other techniques to solve the problem of autonomous driving in indoor scenarios. Special attention should be given to the following points.

The creation of the map works great after some tests. The result can be seen in figure 2. It shows that using lidar technology alone can provide highly accurate indoor maps with a resolution of up to 5 cm. The maps appearance is very similar to the original building plan. Compared to the ground-truth measurements, we achieve an average accuracy of 87%. The SLAM algorithm itself was able to handle the whole environment very well. In contrast, the RL algorithm had issues handling black surfaces and glass fronts, due to the sole usage of the live lidar data. There it is necessary to use additional sensors like the depth camera feed or ultrasonic sensors. Beside of such conceptual issues the hardware itself showed some minor difficulties as well. 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.

With regard to the capabilities of the RL algorithm, it can be shown that the trained model was able to successfully navigate through the environment in the simulator after 200 episodes, 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. Positive tests in real-world scenarios were not as frequent and smooth. To increase the safety the simulation for the RL algorithm can be modelled closer to real life conditions, assisted by the addition of collision avoidance using object detection in the live run.

Finally it is quite surprising, how far a team of students can get when challenging the wide topic of autonomous driving. The combination of lots of open source code, the availability of efficient and affordable hardware and wealth of knowledge fostered by our study course in general have driven our approach of autonomous indoor vehicles to our personal success.

Acknowledgements

We would like to express our gratitude to Prof. Dr. Arndt Balzer and Prof. Dr. Magda Gregorová for their valuable contributions to this research. We also thank THWS for their financial support. Additionally, we extend our appreciation to our colleagues and collaborators for their helpful discussions and suggestions throughout the project. Finally, we are grateful to the reviewers for their constructive comments that have improved the quality of this paper.

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] *ROS Concepts*. 2021. URL: http://wiki.ros.org/ROS/Concepts (visited on 03/04/2023).

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[10] S. Kohlbrecher et al. "A Flexible and Scalable SLAM System with Full 3D Motion Estimation". In: *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. 2011.

[11] 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.

[12] 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.

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

[14] 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).

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

[16] 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.

[17] David A Forsyth and Jean Ponce. *Computer vision: a modern approach*. prentice hall professional technical reference, 2002.

[18] Median Hardiv Nugraha and Dina Chahyati. "Tourism object detection around monumen nasional (monas) using YOLO and retinanet". In: *2020 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. IEEE. 2020, pp. 317–322.

[19] J Terven and D Cordova-Esparza. "A comprehensive review of YOLO: From YOLOv1 and beyond. arXiv 2023". In: *arXiv preprint arXiv:2304.00501* ().

[20] Zihao Zhou et al. "Image processing: Facilitating retinanet for detecting small objects". In: *Journal of Physics: Conference Series*. Vol. 1815. 1. IOP Publishing. 2021, p. 012016.

[21] Xiaoyue Jiang et al. "Deep learning in object detection and recognition". In: (2019).

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

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

[24] 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.