Universitatea Transilvania din Brașov

INTERDISCIPLINARY DOCTORAL SCHOOL

Faculty of Electrical Engineering and Computer Science

Bogdan TRĂSNEA (SIBIȘAN)

Integrating sensor fusion and artificial intelligence techniques for advanced path planning of autonomous vehicles

SUMMARY

Scientific supervisor

Prof. Dr. Eng. Claudiu POZNA

BRAȘOV, 2024

Contents

Doctoral thesis subject

The present work proposes a learning-based algorithm that enables efficient and accurate path planning for mobile robots and autonomous vehicles. Autonomous driving represents a transformative shift in the automotive industry, promising to enhance safety, efficiency, and convenience in transportation. As technology advances, it is poised to revolutionize not only personal transportation but also logistics, public transit, and other sectors reliant on vehicular mobility. The evolution towards fully autonomous vehicles encompasses a broad spectrum of technological innovations, including sensor fusion, machine learning, path planning, and real-time system integration.

Recent developments in the fields of deep learning and artificial intelligence have aided the autonomous driving domain's rapid advancement. Autonomous vehicles (AVs) are robotic systems that can navigate without the need for human intervention. The deployment of AVs is predicted to have a major impact on the future of mobility, bringing a variety of benefits to daily life, such as making driving simpler, increasing road network capacity, and minimizing vehicle-related crashes.

Autonomous driving technology integrates a variety of advanced systems to enable vehicles to navigate and operate without human intervention. These systems include sensor fusion, which combines data from multiple sensors (e.g., LiDAR, radar, cameras) to create a comprehensive understanding of the vehicle's surroundings. Machine learning and AI are utilized to interpret sensor data, make decisions, and learn from driving experiences to improve performance over time. Path planning involves developing optimal routes and making real-time adjustments to navigate dynamic environments. Real-time system integration ensures that all vehicle systems work harmoniously and respond instantly to changes in the environment.

Despite significant advancements, several critical challenges impede the widespread adoption of autonomous vehicles. Ensuring the robustness and reliability of perception systems under diverse environmental conditions, such as varying weather, lighting, and road conditions, is a significant hurdle. Designing interfaces and protocols for seamless interaction between autonomous vehicles and human users, including drivers, passengers, and pedestrians, remains complex. Developing comprehensive regulatory standards and legal frameworks that address liability, safety, and ethical considerations is also essential.

Protecting autonomous vehicles from cyber threats that could compromise safety and privacy is a crucial concern. Addressing ethical dilemmas related to decision-making in unavoidable accident scenarios and understanding the broader societal impact of autonomous vehicles are necessary. Additionally, understanding and mitigating the effects of automation on employment, urban planning,

and public policy is critical.

The development of autonomous driving technology is intrinsically linked to advances in mobile robotics. Mobile robots have long served as the experimental bedrock for autonomous navigation systems, providing crucial insights into real-time decision-making, environment perception, and dynamic path planning. Techniques for mapping and understanding complex environments developed for mobile robots are directly applicable to urban driving scenarios. Algorithms for obstacle avoidance and path optimization in mobile robots are foundational to vehicle autonomy. Strategies for processing sensor data and making split-second decisions in mobile robots are essential for safe autonomous driving.

Looking forward, the future of autonomous driving is expected to be shaped by several key advancements. Continued improvements in AI algorithms will enhance the decision-making capabilities and reliability of autonomous systems. Advances in sensor technology, such as higher-resolution LiDAR and more sophisticated radar systems, will improve environmental perception and safety. Enhanced communication between vehicles, infrastructure, and other road users will facilitate more efficient and safer transportation networks.

Autonomous vehicles will become integral components of smart city infrastructures, contributing to optimized traffic management and reduced urban congestion. The ongoing development of regulatory standards will provide clearer guidelines and facilitate the deployment of autonomous vehicles. Greater collaboration between researchers, industry stakeholders, and policy makers will drive innovation and address the multifaceted challenges of autonomous driving.

The path towards fully autonomous driving is both exciting and challenging, with substantial progress dependent on continued interdisciplinary research and collaboration. By leveraging advancements in mobile robotics and addressing inherent challenges, the realization of safe and efficient autonomous vehicles will move closer to becoming a reality. This transformation promises to reshape transportation, improve safety, and create new opportunities across various sectors, heralding a new era in mobility.

Research objectives

The main objective of this research is to develop, implement, and rigorously test a learning-based algorithm that enables efficient and accurate path planning for mobile robots and autonomous vehicles. By leveraging the latest advancements in deep learning and artificial intelligence, this objective aims to enhance the vehicle's ability to autonomously navigate through dynamic and unpredictable environments. The algorithm should be able to process sensory data and generate optimal routes in real-time, considering both static and dynamic obstacles.

Achieving this objective will result in the development of a validated, efficient, and reliable learning-based path planning algorithm. This will significantly improve the autonomous vehicle's ability to navigate complex environments, ensuring safe and optimal routes. The successful implementation of such an algorithm is crucial for the broader adoption and integration of autonomous vehicles in everyday transportation systems, ultimately enhancing road safety, efficiency, and convenience.

Based on this primary objective, the following specific objectives were identified and categorized into four distinct categories:

1. Comprehensive literature review
    - Conduct an in-depth study of existing literature in the fields of autonomous navigation and path planning.
    - Identify current methodologies, technologies, and gaps in the research related to autonomous vehicles.
2. Simulation environment development
    - Create a simulation tool to establish a controlled testing environment for the proposed algorithms.
    - Ensure the simulation tool accurately represents real-world driving scenarios to validate the effectiveness of the algorithms.
3. Algorithm development
    - Combine data from multiple sensors, such as LiDAR, radar, and cameras, to create a comprehensive understanding of the vehicle's surroundings.
    - Ensure the sensor fusion approach enhances the perception accuracy and reliability of the autonomous vehicle.
    - Develop, implement, and test a learning-based algorithm to enable efficient and effective path planning for mobile robots and autonomous vehicles.
    - Develop a secure development environment on the NVidia AGX Xavier board.
    - Focus on integrating deep learning models and artificial intelligence techniques to enhance the robustness and accuracy of path planning.
    - Develop optimal routes and make real-time adjustments to navigate dynamic environments.
4. Evaluation and benchmarking
    - Ensure the path planning algorithms can handle both static and dynamic obstacles efficiently.
    - Include various performance metrics and testing scenarios to ensure the reliability of the system in diverse conditions.

- Ensure the proposed algorithms perform optimally on various datasets and adapt to different driving conditions.

These objectives aim to ensure the development of a robust, reliable, and secure system for advanced path planning in autonomous vehicles, leveraging the latest advancements in sensor fusion and artificial intelligence.

Thesis structure

The thesis is organized in the following way:

Chapter 1 provides an overview of the autonomous driving research topic, which defines the background and explains its scientific significance. Furthermore, the chapter emphasizes the issues to be addressed, the primary goals, as well as the actions and methods utilized throughout the research activity. Finally, the chapter ends by summarizing the overall structure and content of the thesis.

A broad introduction of deep learning technology is presented in Chapter 2. The fundamental artificial neural network designs, such as convolutional and recurrent neural networks, as well as the idea of deep reinforcement learning, are covered in this chapter. In addition, current deep learning trends for path planning and behavior arbitration, as well as those for driving scene perception and localization, are discussed in depth. Furthermore, the programming aspect of deep learning is covered at the end of the chapter, with a focus on two well-known Python frameworks: Tensorflow and PyTorch.

Chapter 3 describes all of the experimental platforms that were used and built during the thesis study, detailing the development of critical tools and systems for autonomous robot research. It begins with an overview of the GridSim simulation environment, which is a self-driving simulator engine that uses a car-like robot architecture to create occupancy grids using simulated sensors. In addition, GridSim is used to investigate the performance of two deep learning approaches: deep reinforcement learning and driving behavioral learning using genetic algorithms. The Raspberry Pi Robot prototype, which is built on a rear-wheel drive and front-wheel steering system, is then detailed, comparable to models in smart vehicle competitions. The steps of the prototype's development, as well as testing performed with the final version with Lidar and camera, are all documented. Finally, the main robotic platform utilized to test and prove the algorithms outlined in this thesis is given. The robot is a skid-steered wheeled mobile robot platform equipped with e-Cam130A quad cameras and a 360-degree, 40-channel Hesai Pandar Lidar. Following that, a timeline of the robot's development and the experiments which were performed is provided.

The key algorithms created during this thesis are presented in Chapter 4 and Chapter 5, which serve as the thesis's backbone. Chapter 4 begins with a brief description of two-dimensional grid

based representation and continues with a statement of the issue, path planning for autonomous cars, which denotes an autonomous vehicle's capacity to discover a route between two places, namely a starting position and a target location. Following that, the presentation will focus on the idea of behavior arbitration, which will be given from the standpoint of driving scene description, analysis, and modeling, as well as simulation outcomes. In addition, the perspective of trajectory estimation as a cognitive learning problem is discussed. It contains information on how to train a deep neural network to anticipate local ego-vehicle state trajectories via neuroevolutionary training. The solution, coined Neuro-Trajectory, is a multi-objective neuroevolutionary method to local state trajectory learning for autonomous driving, in which a perception-planning deep neural network estimates the intended state trajectory of the ego-vehicle across a restricted prediction horizon. The motion planning problem is sometimes referred to as a sequence to sequence mapping challenge or a sequence creation task.

Chapter 5 details the extension to a 3D representation of the input measurements and sensor fusion, namely OcTrees. Because they feature time-dependent feedback loops, recurrent neural network designs are used to solve such a problem. OctoPath is the outcome, which is a self-supervised encoder-decoder deep neural network that predicts the local optimum course for the ego-vehicle. The installation of OctoPath on an Nvidia AGX Xavier, as well as performance measurement scenarios and results, are also discussed.

Finally, Chapter 6 summarizes the key findings and draws the ultimate conclusions. It also highlights individual contributions, dissemination of the research results, namely patents and papers, and outlines future research directions, focusing on enhancing sensor integration, improving algorithm robustness, and addressing ethical and safety considerations in autonomous driving.

Developing simulation environments

Simulators provide unparalleled scalability, enabling developers to run thousands of test scenarios simultaneously. This scalability is crucial for the thorough validation of path planning algorithms, as it allows for extensive testing across a wide range of conditions and parameters. By leveraging the computational power available in modern simulators, developers can perform large-scale evaluations that would be impractical or impossible in the real world. This capability accelerates the development process and enhances the robustness and reliability of the algorithms.

In real-world testing, reproducing specific scenarios with exact conditions is often challenging. Simulators, however, offer precise control over all aspects of the testing environment, ensuring that scenarios can be replicated consistently. This reproducibility is vital for debugging and refining algorithms, as it allows developers to repeatedly test specific conditions and reliably compare results.

Consistent reproducibility ensures that improvements and optimizations can be accurately assessed, leading to more effective and reliable path planning solutions.

Simulators provide comprehensive data collection capabilities, capturing detailed information about every aspect of the vehicle's performance and the environment. This data is invaluable for analyzing the behavior of path planning algorithms, identifying weaknesses, and making informed improvements. Developers can access rich datasets that include sensor readings, vehicle dynamics, environmental conditions, and interaction with other entities. Such detailed data collection is often difficult and expensive to achieve in real-world testing but is readily available in simulation environments.

The ability to rapidly iterate and test changes in a simulated environment significantly accelerates the development cycle. Developers can quickly implement new features, test their impact, and refine the algorithms based on feedback from the simulation results. This rapid iteration is crucial for keeping up with the fast-paced advancements in autonomous vehicle technology. Simulators enable developers to experiment with innovative ideas and incorporate cutting-edge techniques into their path planning algorithms more efficiently.

Using simulators for developing path planning algorithms offers numerous advantages, including a controlled environment, enhanced safety, cost efficiency, scalability, reproducibility, comprehensive data collection, and an accelerated development cycle. These benefits collectively contribute to the development of robust, reliable, and effective path planning algorithms for autonomous vehicles. By leveraging the power of simulation, developers can overcome many of the challenges associated with real-world testing and advance the capabilities of autonomous driving systems.

GridSim

The developed driving simulator is coined GridSim and it is described as an autonomous driving simulator which uses the non-holonomic robot car kinematics. It has been developed from scratch to support development and validation of autonomous driving systems. It contains a menu which allows the switching between multiple scenarios which are easily represented and loaded into the simulator as backgrounds.

The simulated sensors have a field of view (FOV) of 120 degrees. They react when an obstacle is sensed, by marking it as an occupied area. The static obstacles are a priori mapped to the backgrounds as lists of polygons. The simulated sensors continuously check if the perception rays are colliding with the given polygons. The dynamic obstacles are represented by traffic cars, which have their trajectory randomly generated from a uniform distribution of the possible free spaces inside the given scenario. The longitudinal velocity and the steering angle's rate of change is included in the trajectory. All GridSim features were built using Python and the PyGame library.

The single-track kinematic model is further described. The positions of the front and rear wheels are p_f and p_r, respectively, while α is the heading angle describing the facing direction of the vehicle, defined by angle between vectors $\hat{e}_x$ and $p_f - p_r$. For the sake of clarity, a no-slip assumption for the wheels on the driving surface has been considered. The slipping of the wheels can be modelled by adding the inertial effects generated by the ground on the vehicle's tires, similar to Pacejka's tyre

model.

The wheels rotate freely about their axes of rotation, while the steering is modeled through angle δ as an extra degree of freedom on the front wheel. The vehicle obeys the "non-holonomic" assumption, expressed as a differential constraint on the motion of the car. The non-holonomic constraint restricts the vehicle from making lateral displacements, without simultaneously moving forward.

The forward speed is:

$$v_r = \dot{p}_r \cdot \frac{(p_f - p_r)}{||p_f - p_r||}, \tag{1}$$

where v_r is the magnitude of $\dot{p}_r$ with the correct sign to indicate forward or reverse driving.

The differential constraints written in terms of the motion of p_f, where the front wheel forward speed v_f is used:

$$\dot{x}_f = v_f \cos(\alpha + \delta), \tag{2}$$

$$\dot{y}_f = v_f \sin(\alpha + \delta), \tag{3}$$

$$\dot{\alpha} = \frac{v_f}{l} \sin(\delta). \tag{4}$$

The speed of the front wheel v_f is related to the speed of the rear wheel v_r by:

$$\frac{v_r}{v_f} = \cos(\delta). \tag{5}$$

The planning and control problems for this model involve selecting the steering angle δ within the mechanical limits of the vehicle $\delta \in [\delta_{min}, \delta_{max}]$ and forward speed v_r within an acceptable range $v_r \in [v_{min}, v_{max}]$.

Continuity of the steering angle can be imposed by augmenting the previous model with the steering rate of change:

$$\dot{x}_f = v_f \cos(\alpha + \delta), \tag{6}$$

$$\dot{y}_f = v_f \sin(\alpha + \delta), \tag{7}$$

$$\dot{\alpha} = \frac{v_f}{l} \sin(\delta). \tag{8}$$

$$\dot{\delta} = v_\delta. \tag{9}$$

In addition to the limit on the steering angle, the steering rate can now be limited to:

$$v_\delta \in \left[\dot{\delta}_{min}, \dot{\delta}_{max}\right]. \tag{10}$$

GridSim was used to study the performance of two simulation-based autonomous driving approaches based on occupancy grids: deep reinforcement learning and neuroevolutionary driving using genetic algorithms. The synthetic data utilized as input for both control systems, the DQN agent and the Neuroevolutionary agent, is represented by the simulated Occupancy Grids (OGs). The algorithms are evaluated on the following scenarios: highway (with two different sensors models), curved road, inner-city, and seamless model, each with its own set of constraints and increasing difficulty. The desired behavior of the proposed Neuroevolutionary agent is encoded in a two elements fitness function describing a maximum traveled distance (defined as the remaining distance to the previously defined goal) and a maximum forward speed, bounded to a specific interval.

We have built in GridSim a user interface suitable for creating and training various deep learning topologies. The interface also supports data preprocessing, labeling and annotation. For implementation, we have evaluated three different AI libraries, namely Caffe2, Cognitive Neural Toolkit (CNTK) and TensorFlow. We decided to use TensorFlow, because it has the advantage of supporting fine grain network layers that allow users to build new complex layer types without implementing them in a low-level language. This back-end for the neural networks representations is mixed together with the Keras API and it is written in Python. GridSim can be used in both CPU and GPU configurations.

In order to reduce the time necessary for the training process, we have used the following hardware setup: a desktop computer equipped with an Intel Core i7 7700K CPU, 64 GB RAM, and a high-performance NVIDIA GeForce GTX 1080 Ti graphics card. A script implementation was also necessary for saving the output coordinates of the artificially generated input data. This data is is used in replay mode for training.

The user interface was integrated into the GridSim environment menu, such that the modes can be switched between replay, record, and training, with each one having access to the five different scenarios. There is a large number of configurable parameters, such as the resolution of the simulator, occupancy grid precision, number of traffic participants, ego car's maximum speed and turning radius, etc.

DNNs are usually trained via gradient-based learning algorithms, such as backpropagation. Neuroevolutionary training strategies can rival backpropagation-based algorithms, such as Q-learning and policy gradients, on difficult DRL tasks. The idea which is explored in this work is to evolve the weights of a deep neural network by using a population-based genetic algorithm, with altered breeding rules.

Because the deep network's response is quantified using a multi-objective loss function, its weights are learnt using evolutionary computation. The training aims to compute optimal weights for a collection of deep neural networks $\varphi(\cdot;\mathbf{\Theta})$ by simultaneously optimizing their fitness functions. This learning procedure was first proposed by the authors for training a generative one-shot learning classifier.

Traditional training approaches use algorithms such as backpropagation and a scalar loss function, in order to compute the optimal weight values of a single network. In the case of evolutionary training, $\varphi(\cdot;\mathbf{\Theta})$ represents a collection of K deep networks, each network having a corresponding set of weights Θ_i:

$$\boldsymbol{\varphi}(\cdot;\boldsymbol{\Theta}) = \Big[\varphi_1(\cdot;\Theta_1), \varphi_2(\cdot;\Theta_2), ..., \varphi_i(\cdot;\Theta_i), ..., \varphi_K(\cdot;\Theta_K),\Big]^T \tag{11}$$

The weights of a single deep network are stored in a so-called solution vector $\Theta = \Big[\theta_1, \theta_2, ..., \theta_n,\Big]^T$, composed of n decision variables θ_i, with $i = 1, ..., n$ and $\theta \in \mathbb{R}^n$. θ_i represents a weight parameter in a single network.

Algorithm 0.1 Neuroevolutionary agent in GridSim

```
procedure Train(G)
    encoded_w ← encode_w(net_topology)
    ppl ← init_gen(encoded_w)
    while not end of all generations do
        while not end of generation do
            weights ← decode_weights()
            dist, vel ← fitness_eval(weights)
            gen_score[i] ← dist, vel
        end while
        ppl ← tournament_sel(gen_score, ppl)
        ppl ← uniform_mate(α, ppl)
        ppl ← gaussian_mutate(β, ppl)
    end while
    elite_first ← k_best_selection(1, ppl)
    validate_elite(elite_first)
end procedure
```

Using genetic algorithms, the weights $\boldsymbol{\Theta}$ of a population of $\boldsymbol{\varphi}(\cdot;\boldsymbol{\Theta})$ deep networks, where an individual Θ is a solution vector containing the weights of a network $\varphi(\cdot;\Theta)$, were evolved. The first step in the training consists of running a forward pass through the population of networks, thus obtaining values for the multi-objective fitness function $L(\cdot)$. After completing forward passes for all networks population, we use the tournament selection algorithm to select the best individuals.

The tournament selection algorithm assures that a number of elite individuals, with the best accuracy, carry on to the next generation unmodified. For exploring the decision space S, we used an uniform crossover between 2 individuals from a batch of individuals, with an independent probability α of happening. Furthermore, a mutation operation is applied on the same batch, but with an independent probability β. The next generation of individuals' genes are subjected to an additive Gaussian noise σ:

$$\theta' = \theta + \sigma; \sigma \in [-3, 3] \tag{12}$$

The $[-3, 3]$ interval for the Gaussian noise was chosen with respect to the sigmoid function used for activating neurons in the deep networks. Namely, the interval aims to not oversaturate the values

of the weights by reaching maximum or minimum values returned by a sigmoid. The new population is evaluated and the process repeats itself for G generations.

Firstly, the decision space of the Neuroevolutionary Agent was reduced to three actions (turn left, turn right and no action), starting from the original decision space of eight actions (accelerate, turn left, turn right, brake, accelerate + turn left, accelerate + turn right, reverse/negative acceleration and no action).

This first simplified version of the environment also maintains the velocity of the simulated car at a constant rate, and uses a seamless road model which has no t-junctions or intersections. By using the mentioned environment, the genetic algorithm selects the best combination of weight parameters for a feed forward neural network.

In the beginning, the fitness function was defined with only one metric, which is the traveled distance. The neural network performs an action in the simulator and the fitness function gets updated with the new value. This distance is computed as the Euclidean distance (measured in pixels) between the starting position and the new position, obtained after taking the predicted action. The selection criteria of the genetic optimization was based on the traveled distance, having the objective to maximize it:

$$\rho = f(\delta) \tag{13}$$

To build the next generation of individuals, the following equation was used:

$$\Theta_{i+1} = max(\rho) + rand(\Theta_i) + Cx(\Theta_i) + \mu(\Theta_i) \tag{14}$$

The input of the neural network is a vector described by the values of the occupancy grid generated by the synthetic beams of the radar sensor model. The number of sensor beams is also configurable and can be increased to any resolution necessary.

After the desired behavior was met, and the car was able to navigate the seamless generated model by itself, we performed incremental updates to the decision space, until reaching five actions (accelerate, turn right, turn left, brake and no action). We have also removed the constant velocity, and starting from this model we included a new objective to be maximized in the fitness function, the velocity of the car:

$$\rho = f(\delta, \theta) \tag{15}$$

The internal metric used by the simulator is ppu (pixels per unit). This defines how many pixels go into a single unit, and we use this value to correlate the simulator velocity to the real world measurement of meters/second. The unit can be defined as a fixed number of frames, or a fixed number of seconds. We have defined our unit as a single frame, with the simulator running as 30 frames per second, thanks to PyGame's internal renderer.

We imposed a reachable top speed threshold of $30ppu$, and then we trained the model to adapt the brake action to the environment, in order to not crash in the steep curves of the generated seam-

less environment. After 26 generations the model could navigate without crashing with an average speed of $15ppu$.

After passing each incremental step of complexity, the action space up was increased up to the dimensionality of eight actions, together with the vector size of the sensor input, thus reaching the original complexity of the DQN environment.

Raspberry PI robot

The Raspberry Pi robot is based on the ER-SER85080C chassis. This features a rear-wheel-drive front-wheel steering mechanism, similar to the models in smart car competitions. The floor plates are made of aluminum alloy, which greatly improves the strength of the body. It is composed of two 1500 RPM DC Motors for each rear wheel, with a maximum speed of 4.5 m/s and a motor deceleration ratio of 1:10. The steering is performed via a DS3119 switch motor, with a steering torque of 20kg. It also comes with separate encoders for each rear motor, with the following precision settings: the wheel turns one revolution, the 30-speed reduction ratio can output 1560 jump edges, whereas the 10-speed reduction ratio can output 520 jump edges.

Testing path planning algorithms on a mobile robot controlled by a Raspberry Pi involves a series of carefully designed experiments to evaluate the robot's ability to navigate and perform tasks efficiently. These tests should cover various aspects of path planning, including obstacle avoidance, navigation accuracy, real-time processing, and adaptability to dynamic environments. Below are some key tests that can be performed to assess the path planning capabilities of such a mobile robot.

The basic navigation test aims to verify the robot's ability to navigate from a starting point to a destination point using the planned path. To conduct this test, a simple environment with a predefined start and end point is set up. The path planning algorithm is then programmed to generate a path between these points. Upon execution, the robot's ability to follow the path is observed, with measurements taken for the time required and the accuracy of reaching the destination. This test ensures that the robot can reach the destination accurately, with minimal deviation from the planned path, and within a reasonable time frame.

Obstacle avoidance is crucial for autonomous navigation. In this test, obstacles of various shapes and sizes are randomly placed in the robot's environment. The path planning algorithm must account for these obstacles and re-plan the path as necessary. By executing the algorithm and observing the robot's behavior in real-time, developers can assess whether the robot successfully avoids obstacles while reaching the destination. The evaluation criteria include the robot's ability to avoid collisions, smoothly reroute around obstacles, and maintain a reasonable time to reach the destination.

Assessing the robot's ability to adapt to moving obstacles is the focus of the dynamic obstacle test. Moving obstacles, such as other mobile robots or manually controlled objects, are introduced into the environment. The path planning algorithm must include real-time path re-planning capabilities to adapt to these changes. By executing the algorithm and observing the robot's real-time adaptations, the test evaluates how effectively the robot avoids moving obstacles and maintains its

course. Key factors include the effectiveness of avoidance maneuvers, real-time re-planning efficiency, and minimal interruptions or delays in reaching the destination.

The precision navigation test evaluates the robot's ability to navigate through narrow passages and complex environments. A test environment with narrow passages, tight corners, and complex pathways is created. The path planning algorithm is programmed to navigate through this challenging environment. Upon execution, the robot's performance is observed, focusing on its ability to maintain the planned path within narrow and complex spaces. Evaluation criteria include accurate navigation through narrow passages, minimal collisions with boundaries, and the smoothness and continuity of motion.

The sensor reliability test verifies the reliability and accuracy of the sensors used for path planning. The robot is equipped with various sensors, such as ultrasonic and infrared for environment detection. Each sensor is tested individually to ensure accurate readings and reliable performance. These sensors are then integrated into the path planning algorithm. By executing the algorithm in different environments and lighting conditions, the test assesses sensor reliability. Consistent and accurate sensor readings, reliable performance across different environments, and proper integration and utilization of sensor data in path planning are the key evaluation criteria.

Measuring the energy consumption of the robot during path planning and navigation is the objective of the energy efficiency test. The power usage of the Raspberry Pi and the robot's motors is monitored during operation. A series of path planning tasks of varying complexity are executed, and the energy consumption for each task is recorded. The analysis focuses on the correlation between path complexity, obstacle avoidance, and energy usage. The evaluation criteria include efficient energy consumption during navigation, the impact of path planning complexity on energy usage, and the optimization of path planning for energy efficiency.

The real-world scenario test evaluates the robot's performance in a real-world environment with real-life obstacles and scenarios. A test environment that mimics a typical operational scenario for the robot, such as a household setting or office environment, is set up. The path planning algorithm is programmed to navigate through this environment. By executing the algorithm and observing the robot's behavior in real-world conditions, the test assesses the robot's robustness and adaptability. Successful navigation and task completion in a real environment, effective handling of unexpected obstacles, and adaptability to changes are the key evaluation criteria.

Testing path planning algorithms on a mobile robot controlled by a Raspberry Pi involves a comprehensive set of experiments designed to evaluate various aspects of the robot's navigation capabilities. These tests ensure that the path planning algorithm is robust, efficient, and reliable in diverse environments and conditions. By systematically conducting these tests, developers can refine and optimize their algorithms, ultimately enhancing the performance of autonomous mobile robots.

Agile Scout AMTU

The Agile Scout robot is an advanced mobile robotic system designed for a variety of applications, including exploration, surveillance, search and rescue, and environmental monitoring. Its de-

velopment focuses on combining agility, adaptability, and robustness to operate effectively in diverse and often challenging environments. The following sections provide a detailed description of its key features and capabilities.

At the core of the Agile Scout robot's design is its exceptional mobility. It is typically equipped with a set of highly versatile wheels or tracks that allow it to navigate over uneven terrain, obstacles, and steep inclines with ease. Some models may also feature articulated legs or hybrid designs that combine wheels and legs to further enhance maneuverability. This flexibility enables the robot to move swiftly and efficiently in both indoor and outdoor settings, making it suitable for tasks that require rapid deployment and extensive area coverage.

The Agile Scout robot is outfitted with a comprehensive suite of sensors that provide it with a rich understanding of its environment. These sensors often include LIDAR, ultrasonic sensors, infrared cameras, and high-resolution visual cameras. LIDAR is particularly useful for creating detailed 3D maps of the surroundings, which is crucial for navigation and obstacle avoidance. The combination of these sensors allows the robot to detect and recognize objects, measure distances, and identify potential hazards in real-time.

One of the standout features of the Agile Scout robot is its advanced autonomy. Effective communication is essential for the Agile Scout robot, especially in applications like search and rescue or surveillance. The robot is equipped with robust communication systems that enable it to transmit data back to a control center in real-time. This data includes video feeds, sensor readings, and status updates, allowing operators to monitor the robot's progress and make informed decisions. The control interface is typically user-friendly, offering both manual control options and autonomous operation modes. This dual capability ensures that human operators can intervene when necessary, while still benefiting from the robot's autonomous functions.

The versatility of the Agile Scout robot makes it suitable for a wide range of applications. In exploration and environmental monitoring, it can be used to gather data in hazardous or inaccessible areas, providing valuable insights without putting human lives at risk. In search and rescue missions, its agility and sensor capabilities allow it to locate and assist victims in disaster-stricken areas. Surveillance and security applications benefit from the robot's ability to patrol large areas autonomously, detect intrusions, and relay real-time information to security personnel.

The Agile Scout robot represents a significant advancement in mobile robotics, combining agility, advanced sensing, autonomy, and robust communication to perform a variety of critical tasks. Its ability to navigate complex environments, coupled with its autonomous capabilities, makes it an invaluable tool in fields ranging from exploration and environmental monitoring to search and rescue and surveillance. As technology continues to evolve, the Agile Scout robot is poised to become even more capable, further expanding its range of applications and effectiveness in mission-critical scenarios.

Agile scout is a SSWMR (skid-steer wheeled mobile robot). The following model assumptions are taken into account:

1. The robot's mass center is at the geometric center of the body frame;

2. Each side's two wheels rotate at the same speed;

3. The robot is operating on a firm ground floor with all four wheels in contact with it at all times.

We define an inertial frame (X, Y) (global frame) and a local (robot body) frame (x, y). Presume the robot movies in a plane with linear velocity $v = (v_x, v_y, 0)^T$ and rotates with an angular velocity $\omega = (0, 0, \omega_z)^T$, both expressed in the local frame. If $q = (X, Y, \theta)^T$ is the state vector defining the robot's generalized coordinates (position X and Y, as well as the orientation θ of the local coordinate frame with respect to the inertial frame), then $\dot{q} = (\dot{X}, \dot{Y}, \dot{\theta})^T$ is the vector of generalized velocities.

The relationship between the robot velocities in both frames is then calculated as follows:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix}. \tag{16}$$

Because it only specifies free-body kinematics, Equation (16) places no limits on the SSWMR plane movement. As a result, the relationship between wheel velocities and local velocities must be analyzed. For simplicity, the thickness of the wheel is neglected and is assumed to be in contact with the plane at point P_i, as according to the initial model assumption nr. 3. In comparison to other wheeled vehicles, the SSWMR has a non-zero lateral velocity. This property stems from the SSWMR's mechanical structure, which necessitates lateral skidding if the vehicle's orientation shifts. As a result, the wheels are only tangent to the path when $\omega = 0$, that is, when the robot travels in a straight line. It is important to consider all wheels together when developing the kinematic model.

Let ω_i and v_i, with $i = 1, 2, 3, 4$ denote the wheel angular and center linear velocities for front-left, rear-left, front-right, and rear-right wheels, respectively. Thus, we have:

$$\omega_L = \omega_1 = \omega_2, \qquad \omega_R = \omega_3 = \omega_4. \tag{17}$$

We can use the previous equation to state the direct kinematics on the plane:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = f \begin{bmatrix} \omega_l r \\ \omega_r r \end{bmatrix}, \tag{18}$$

where $v = (v_x, v_y)$ is the vehicle's translational velocity with respect to its local frame, ω_z is its angular velocity, and r is the radius of the wheel.

The instantaneous centers of rotation (ICR) of the left-side, right-side, and robot body are denoted as ICR_l, ICR_r, and ICR_G, respectively, while the mobile robot moves. ICR_l, ICR_r, and ICR_G are all known to lie on a line parallel to the x-axis. We define the x-y coordinates for ICR_l, ICR_r, and ICR_G as (x_{ICR_l}, y_{ICR_l}), (x_{ICR_r}, y_{ICR_r}), and (x_{ICR}, y_{ICR}), respectively. The sides' angular velocity is equal to the velocity of the robot body ω_z. We further obtain the following geometrical relations:

$$x_{ICR} = x_{ICR_l} = x_{ICR_r} = -\frac{v_y}{\omega_z} \tag{19}$$

$$y_{ICR} = \frac{v_x}{\omega_z}, \tag{20}$$

$$y_{ICR_l} = \frac{v_x - \omega_l r}{\omega_z}, \tag{21}$$

$$y_{ICR_r} = \frac{v_x - \omega_r r}{\omega_z}. \tag{22}$$

From Equations (19)–(22), the kinematics relation (18) can be represented as:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = J_\omega \begin{bmatrix} \omega_l r \\ \omega_r r \end{bmatrix}, \tag{23}$$

where the elements of matrix J_ω are determined by the ICR coordinates on the left and right sides:

$$J_\omega = \frac{1}{y_{ICR_l} - y_{ICR_r}} \begin{bmatrix} -y_{ICR_r} & y_{ICR_l} \\ x_{ICR} & -x_{ICR} \\ -1 & 1 \end{bmatrix}. \tag{24}$$

Since the SSWMR is symmetrical in our case, we can obtain a symmetrical kinematics model. As a result, the ICRs are symmetrically distributed on the x-axis, and the matrix J_ω can be written as follows:

$$J_\omega = \frac{1}{2y_{ICR_0}} \begin{bmatrix} y_{ICR_0} & y_{ICR_0} \\ 0 & 0 \\ -1 & 1 \end{bmatrix}, \tag{25}$$

where $y_{ICR_0} = y_{ICR_l} = -y_{ICR_r}$ represents the side ICR values. Considering that, for our symmetrical model, $v_l = \omega_l r$ and $v_r = \omega_r r$, the relations between the angular wheel velocities and the robot velocities are as follows:

$$\begin{cases} v_x = \frac{\omega_l r + \omega_r r}{2} = \frac{v_l + v_r}{2} \\ v_y = 0 \\ \omega_z = \frac{-\omega_l r + \omega_r r}{2y_{ICR_0}} = \frac{-v_l + v_r}{2y_{ICR_0}} \end{cases}. \tag{26}$$

Based on Equation (26), the control signal u can be written as:

$$u = \begin{bmatrix} v_x \\ \omega_z \end{bmatrix} = r \begin{bmatrix} \frac{\omega_l + \omega_r}{2} \\ \frac{-\omega_l + \omega_r}{2y_{ICR_0}} \end{bmatrix}. \tag{27}$$

The last equation shows that the pair of angular velocities ω_l and ω_r, as well as velocities v_x and ω_z, can technically be viewed as a control kinematic input signal. The accuracy of relation (27), on the other hand, is heavily reliant on longitudinal slip, and it can only be used if this phenomenon is not dominant. Furthermore, the parameters r and y_{ICR_0} can be calculated experimentally to ensure that

the angular robot velocity is accurately estimated in relation to the angular velocities of the wheels.

2D path planning

The ability of an autonomous car to find a route between two points, that is, a start position and a desired location, represents path planning. According to the path planning process, a self-driving car should consider all possible obstacles that are present in the surrounding environment and calculate a trajectory along a collision-free route. As stated before, autonomous driving is a multi-agent setting where the host vehicle must apply sophisticated negotiation skills with other road users when overtaking, giving way, merging, taking left and right turns, all while navigating unstructured urban roadways. The literature findings point to a non trivial policy that should handle safety in driving. Considering a reward function $R(\bar{s}) = -r$ for an accident event that should be avoided and $R(\bar{s}) \in [-1, 1]$ for the rest of the trajectories, the goal is to learn to perform difficult maneuvers smoothly and safe.

Traditionally, DNNs are trained using differentiable methods based on single-objective cost functions. Although multiple losses can be aggregated into a single cost function via weighting, the gradient descent step in the backpropagation algorithm will adjust the network's weights based only on the single-objective loss. Weighting multiple losses also introduces additional hyperparameters (typically manually defined) required to weight each individual loss.

Another approach to multi-objective optimization in deep learning is Multi-Task Learning (MTL), where the objectives are given as tasks. The network model either shares its layers between weighted tasks (hard parameter sharing), or each single task is used to train a separate model (soft parameter sharing). In the latter case, the parameters between the models corresponding to the given tasks are regularized in order to encourage the parameters to be similar. As stated before, the hard parameter sharing paradigm is still pervasive for neural-network based MTL. In contrast to MTL, we use a Pareto multi-objective optimization technique which independently optimizes a population of DNNs, where the training of each individual is not influenced by the other individuals. In this way, we ensure a better exploration of the parameters space during training, while avoiding additional weighting hyperparameters.

Problem definition

Given a sequence of 2D occupancy grids (OG) $\vec{X} : \mathbb{R}^2 \times \tau_i \rightarrow \mathbb{R}^2 \times \tau_o$, the position of the ego-vehicle $\vec{p}_{ego}^{<t>} \in \mathbb{R}^2$ in $\vec{x}^{<t>}$ and the destination coordinates $\vec{p}_{dest}^{<t>} \in \mathbb{R}^2$ in occupancy grid space at time t, the task is to learn a local trajectory for navigating the ego-vehicle to destination coordinates $\vec{p}_{dest}^{<t+\tau_o>}$. τ_i is the length of the OGs input sequence, while τ_o is the number of time steps for which

the trajectory of the ego-vehicle is estimated.

In other words, with $\vec{p}_0^{<t>}$ being a coordinate in the current OG observation $\vec{x}^{<t>}$, we seek a desired local navigation trajectory of the ego-vehicle from any arbitrary starting point $\vec{p}_0^{<t>}$ to $\vec{p}_{dest}^{<t+\tau_o>}$, with the following properties:

- the traveled path $||\vec{p}_0^{<t>} - \vec{p}_{dest}^{<t+\tau_o>}||$ is minimal;
- the lateral velocity, given by the steering angle's rate of change $v_\delta \in \left[\dot{\delta}_{min}, \dot{\delta}_{max}\right]$ is minimal, signifying a minimal value for $v_\delta^{<t,t+\tau_o>}$;
- the forward speed, also known as longitudinal velocity, $v^{<t,t+\tau_o>}$ is maximal and bounded to an acceptable range $[v_{min}, v_{max}]$.

The vehicle is modeled based on the single-track kinematic model of a robot, with position state $\vec{y}^{<t>} = (p_x^{<t>}, p_y^{<t>})$ and no-slip assumptions. p_x and p_y represent the position of the vehicle in the 2D driving plane, respectively. The heading is not taken into consideration for trajectory estimation.

We observe the driving environment using OGs constructed from fused LiDAR and radar data. Green and red pixels represent free-space and obstacles, respectively, while black signifies unknown occupancy. A single OG corresponds to an observation instance $\vec{x}^{<t>}$, while a sequence of OGs is denoted as $\vec{X}^{<t-\tau_i,t>}$. These observations are axis-aligned discrete grid sequences, acquired over time interval $[t - \tau_i, t]$ and centered on the sequence of vehicle states $\vec{Y}^{<t-\tau,t>}$.

AI Behavior Arbitration (AIBA)

Autonomous Vehicles (AVs) are robotic systems that can guide themselves without human operators. Such vehicles are equipped with Artificial Intelligence (AI) components and are expected to change dramatically the future of mobility, bringing a variety of benefits into everyday life, such as making driving easier, improving the capacity of road networks and reducing vehicle-related accidents.

Most likely, due to the lack of safety guarantees and legislation, as well as the missing scalability of Autonomous Driving (AD) systems, fully autonomous vehicles will not travel the streets in the near future. Nevertheless, over the past years, the progress achieved in the area of AI, as well as the commercial availability of Advanced Driver Assistance Systems (ADAS), has brought us closer to the goal of full driving autonomy.

The Society of Automotive Engineers (SAE) has defined standard J3016 for different autonomy levels, which splits the concept of autonomous driving into 5 levels of automation. Levels 1 and 2 are represented by systems where the human driver is required to monitor the driving scene, whereas levels 3, 4 and 5, with 5 being fully autonomous, consider that the automated driving components are monitoring the environment.

An AV must be able to sense its own surroundings and form an environment model consisting of moving and stationary objects. Afterwards it uses this information in order to learn long term

driving strategies. These driving policies govern the vehicle's motion and automatically output control signals for steering wheel, throttle and brake. At the highest level, the vehicle's decision-making system has to select an optimal route from the current position to the destination.

The main reason behind the human ability to drive cars is our capability to understand the driving environment, or driving context. In the followings, we will refer to the driving context as the scene and we will define it as linked patterns of objects. In this paper, we introduce AIBA (AI Behavior Arbitration), an algorithm designed to arbitrate between different driving strategies, or vehicle behaviors, based on AIBA's understanding of the relations between the scene objects.

A driving scene consists of objects such as lanes, sidewalks, cars, pedestrians, bicycles, traffic signs, etc., all of them being connected to each other in a particular way (e.g. a traffic sign displays information for a driver). Driving behavior prediction is a key part of an AV's decision-making process. The task of identifying the future behavior of the scene's objects is not a trivial one, since this type of information cannot be directly measured or communicated, being considered latent information. To be able to perform behavior prediction for the surrounding vehicles, an AV can use mathematical models which consider the variation in the objects' movement and describe the driving scenario from the view point of the ego-car.

Such kind of models use several types of information, i.e. vehicle kinematics, the relationship between the ego-car and the surrounding entities, the interactions with other vehicles and a-priori knowledge. Vehicle kinematics and the relations with road entities were considered by almost all existing studies.

Usually, most models for behavior arbitration are tailored for one specific scenario. Nevertheless, AVs must drive through dynamically changing environments in which a diversity of scenarios occur over time. Multiple scenario-specific models activate a corresponding model according to the characteristics of the scenario. AIBA is a system for behavior arbitration in AVs, which constructs a description and understanding model of the driving scene. Our idea is to model the human driver (HDr) understanding process of the driving scene, in order to achieve an optimal behavior arbitration solution. We describe the mechanism of the HDr thinking and transpose it to an approximate model.

The driving scene description is given from a human driver's perspective, and it formulates properties derived from the definitions of classes, subclasses and objects which represent the core of an abstraction model, based on the authors' previous work. The main idea behind AIBA is to model, or formalize, the HDr understanding process and afterwards transform it into a formal model for behavior arbitration in AV.

A human driver is able to perceive the scene's objects and observe them. This means that the HDr identifies the concepts and the different properties of the objects. The different scene objects are linked between them, and the links definition is a first step in the knowledge process, which means it establishes the subjects of interest and also the importance level of each subject. The HDr scene understanding synthesis contains the following steps: identifying the link between the objects, allocating models to each link, running the models and afterwards finding a strategy to act. In fact, the HDr creates an implicit system and simulates it.

Two types of links can be observed: internal links, considered to be the set of links between the

observer (the HDr) and all the observed objects, and external links, represented by the set of links between the objects present in the scene. From the HDr point of view, the links have different meanings and importance, or significance. Specifically, a human driver knows the traffic rules and how to get to his destination. These rules will determine traffic priorities, thus making the HDr to attach a greater importance to those links which are more important to his strategy.

In the next step, a description is established for each link. During the driving, the HDr adapts, or refines, the mentioned description by observation. The driver, by using the importance of the links, simulates a scene description. If we analyze the aim of driving scene understanding, we will observe that its origins are the stability in time and space. More precisely, the human driver has a driving task which can be accomplished if the possibility of locomotion is preserved in the current position and during a specific time. Intuitively, the stability is related to the objects in the scene and can be reached by understanding the scene. This understanding does not solve the driving problem, but offers the information upon which the human driver decides to act on a certain behavior.

The previous analysis of the driving scene understanding can be described like a process which consists of the following steps:

1. Perceiving the objects (cars, traffic signs, lanes, pedestrians, etc.) and obtaining the object properties (the pedestrian intention is to cross the road).

2. Defining the links (the scene network) between the objects and their importance (i.e. the car in front is important, the pedestrian which intends to cross the road is very important, etc.).

3. Adding models to the mentioned links.

4. Simulating the model of the most significant links, and proposing driving behaviors which will be verified with the other significant links until the appropriate behavior (which allows the driving task) is found.

Entities like objects, links, or networks, which have been introduced in the previous section, have correspondences in the modeling process. Our intention is to approximate the HD understanding process through a formal representation. More precisely, this assignment mimics an input/output process: using a perceived scene, the AIBA model must output a description which offers all the information needed within the AD system to arbitrate the driving behavior.

Within AIBA, the first action is to transform the scene in to a collection of objects. This operation is accomplished by the following generative function:

$$obj\_gen : \Sigma \times K \rightarrow \Omega \tag{28}$$

where Σ is the perceived scene; K is the set of known object classes and Ω is the set of objects.

Having an initial collection of classes, the generator function from Eq. 45 transforms the scene entities into a collection of objects. The following object classes are taken into consideration: traffic participants, pedestrians and buildings. The set of classes are a priori set within AIBA.

The complexity of this process, even for the set of road classes, is seen in many types of roads which exist around the world. In order to reduce the scene's complexity, we split them in two major classes, static objects (lanes, traffic signs, buildings, etc.) and dynamic objects (cars, pedestrians, etc.).

Two kinds of definitions are here significant: the generic definition where the proximity and the properties are mentioned, and the extensive definition where the definition of the object (or a picture of it) is indicated or shown. This observation enables us to associate image recognition methods (which correspond to the extensive definition) with a generic definition collection.

Eq. 45 can be generalized when measurements (speed of the cars, distance between cars, size of traffic signs etc.) are associated with the scene description:

$$obj\_gen : \Sigma \times K \times M \rightarrow \Omega, \tag{29}$$

where M is the set of measurements. The generated objects Ω contain a structure of class specific properties and methods. The methods reflect the possible interactions of the objects with the ego-car.

The second step in AIBA's workflow defines the links between a HDr and an object, as well as between the objects themselves, respectively:

$$link\_gen : \Omega \times T \rightarrow \Lambda, \tag{30}$$

where T is the set of task trajectory properties and Λ is the set of links' significance.

Because the links are computed in term of stability around a particular point of the task trajectory, the significance is correlated with specific threats. Driving a car is subject to implicit negotiation based on traffic rules. The most important links are those related to agents which, according to these rules, have priority. Eq. 30 can be imagined as an expert system which will analyse all these links from the mentioned point of view, while outputting different marks:

$$obj\_gen : \Sigma \times K \times M \times T \rightarrow \Omega \times \Lambda. \tag{31}$$

Each recognized object provides methods which refer to the ego-car - object interactions. Information about possible threats on the task stability can be obtained by simulating the behaviors, and also how to select the appropriate driving behavior for avoiding these threats:

$$thr\_sim : \Omega_S \times H \rightarrow \Theta \times B, \tag{32}$$

$$\Omega_S = \left\{ O_i | O_i \in \Omega; S_{O_i,O_j} \geq S_{min} \right\} \tag{33}$$

$$H = \begin{bmatrix} t_c & t_c + \delta \end{bmatrix} \tag{34}$$

where Ω_S is the set of important objects O_i, which are linked with other objects O_j with a significance $S_{O_i,O_j} \in \Lambda$, greater than a minimum (a priori imposed) significance S_{min}. H is the time horizon, t_c

is the current time, δ is the simulation time, Θ is the set of the threat levels and B is the set of recommended behaviors for the ego-car.

thr_sim will simulate, for each important object O_i, a collection of behaviors $O_i\_M_{i,k}(P,H)$ and a predicted threat level $\Theta_{i,k}$:

$$O_i\_M_{i,k}(P,H) = \begin{bmatrix} \Theta_{i,k} \\ \beta_{i,k} \end{bmatrix} \tag{35}$$

where P is the set of object properties and $\beta_{i,k}$ is the behavior of the ego-car which will eliminate the threat.

In order to solve all links' threats, several strategies can be chosen. If we adopt the HDr understanding description from the previous section, the behavior which solves the maximum threat is simulated for the other threats, obtaining the optimal behavior of the ego-car $\Theta_{max} = \Theta_{i^*,k^*}$:

$$(i^*, k^*) = \arg\max_{i,k} \Theta_{i,k} \tag{36}$$

where $\Theta_{i,k}$ are the threat levels.

The last step in AIBA's modelling system is the transformation of the optimal behavior Θ_{max} into a natural language explanation:

$$dsc : \Lambda \times B \rightarrow \Delta \tag{37}$$

$$\Delta = \Omega_S \times E \tag{38}$$

$$E = e_1 \times e_2 \times ... \times e_{n_S} \tag{39}$$

where Δ is the set of descriptions, e_i is the explanation of the significance associated to an object and n_S is the number of significant objects.

NeuroTrajectory

This thesis' approach to local state trajectory learning is to reformulate the autonomous driving problem as a cognitive learning task. The above problem can be modeled as a Markov Decision Process (MDP) $M = (S, A, T, L)$, where:

- S represents a finite set of states, $\vec{s}^{<t>} \in S$ being the state of the agent at time t. To encode the location of the agent in the driving OG space at time t, we define $\vec{s}^{<t>} = \vec{X}(\vec{p}_{ego}^{<t-\tau_i,t>})$, which denotes an axis-aligned discrete grid sequence in interval $[t - \tau_i, t]$, centered on the ego-vehicle positions' $\vec{p}_{ego}^{<t-\tau_i,t>}$.
- A represents a finite set of trajectory sequences, allowing the agent to navigate through the environment, where $\vec{Y}^{<t+1,t+\tau_o>} \in A$ is the predicted trajectory that the agent should follow

in the future time interval $[t+1, t+\tau_o]$. A trajectory $\vec{Y}^{<t+1,t+\tau_o>}$ is defined as a collection of estimated trajectory state set-points:

$$\vec{Y}^{<t+1,t+\tau_o>} = [\vec{y}^{<t+1>}, \vec{y}^{<t+2>}, ..., \vec{y}^{<t+\tau_o>}]. \tag{40}$$

- $T : S \times A \times S \rightarrow [0, 1]$ is a stochastic transition function, where $T^{\vec{s}^{<t+\tau_o>}}_{s^{<t>}, \vec{Y}^{<t+1,t+\tau_o>}}$ describes the probability of arriving in state $\vec{s}^{<t+\tau_o>}$, after performing a motion along trajectory $\vec{Y}^{<t+1,t+\tau_o>}$.
- $\vec{L} : S \times A \times S \rightarrow \mathbb{R}^3$ is a multi-objective fitness vector function which quantifies the trajectory quality of the ego-vehicle:

$$\vec{L}^{\vec{s}^{<t+\tau_o>}}_{\vec{s}^{<t>}, \vec{Y}^{<t+1,t+\tau_o>}} = \left[l_1^{<t+\tau_o>} \quad l_2^{<t+\tau_o>} \quad l_3^{<t+\tau_o>} \right]. \tag{41}$$

Each element in Eq. 41 is defined as:

$$l_1^{<t+\tau_o>} = \sum_{i=1}^{\tau_o} ||\vec{p}_{ego}^{<t+i>} - \vec{p}_{dest}^{<t+i>}||_2^2 \tag{42}$$

$$l_2^{<t+\tau_o>} = \sum_{i=1}^{\tau_o} v_\delta^{<t+i>} \tag{43}$$

$$l_3^{<t+\tau_o>} = \sum_{i=1}^{\tau_o} v_f^{<t+i>} \in [v_{min}, v_{max}] \tag{44}$$

Intuitively, $l_1^{<t+\tau_o>}$ represents a distance-based feedback, which is smaller if the car follows a minimal energy trajectory to $\vec{p}_{dest}^{<t+\tau_o>}$ and large otherwise. $l_2^{<t+\tau_o>}$ quantifies hazardous motions and passenger discomfort by summing up the lateral velocity of the vehicle. The feedback function $l_3^{<t+\tau_o>}$ is the moving longitudinal velocity of the ego-vehicle, bounded to speeds appropriate for different road sectors, such as $v^{<t,t+\tau_o>} \in [80kmh, 130kmh]$ for the case of highway driving.

Considering the proposed state estimation scheme, the goal is to train an optimal approximator, defined here by a deep network, which can predict the optimal state trajectory $\vec{Y}^{<t+1,t+\tau_o>}$ of the ego-vehicle, given a sequence of occupancy grid observations $\vec{X}^{<t-\tau_i,t>}$ and the multi-objective fitness vector from Eq. 41.

We learn an optimal state trajectory by combining Convolutional Neural Networks (CNN) with the robust temporal predictions of Long Short-Term Memory (LSTM) networks. The two types of neural architectures are combined as follows. An observation $\vec{x}^{<t>}$ is firstly processed by a CNN, implemented as a series of convolutional layers, aiming to extract relevant spatial features from the input data. The CNN outputs a feature-space representation for each observation in $\vec{X}^{<t-\tau_i,t>}$. Each processed spatial observation in the input interval $[t-\tau_i, t]$ is flatten and passed through two fully connected layers of 1024 and 512 units, respectively. The input sequence into an LSTM block is represented by a sequence of spatially processed observations, denoted as $CNN^{<t-\tau_i,t>}$. The same network topology can be trained separately on synthetic, as well as on real-world data. As trainable network parameters, we consider both the weights of the LSTM networks, as well as the weights of the convolutional layers.

For computing the state trajectory of the ego-vehicle, we have designed the deep neural network, where OG sequences are processed by a set of convolutional layers, before being feed to different LSTM network branches. Each LSTM branch is responsible for estimating trajectory set-points along time interval $[t+1, t+\tau_o]$. The choice for a stack of LSTM branches over a single LSTM network that would predict all future state set-points comes from our experiments with different network architectures. Namely, we have observed that the performance of a single LSTM network decreases exponentially with the prediction horizon τ_o. The maximum value for which we could obtain a stable trajectory using a single LSTM network was $\tau_o = 2$. As shown in the experimental results section, this is not the case with our proposed stack of LSTMs, where each branch is responsible for estimating a single state set-point.

In order to train the deep network on as many corner cases as possible, we have constructed a training dataset based on real-world occupancy grid samples $\vec{X}^{<t-\tau_i,t>}$, as well as on synthetic sequences $\hat{\vec{X}}^{<t-\tau_i,t>}$. Synthetic data is generated in our OG simulator GridSim. As trainable network parameters, we consider both the weights of the LSTM network, as well as the weights of the convolutional layers. The synthetic and real-world data streams are processed by a convolutional network, before being fed to LSTM networks via two fully connected layer of 1024 and 512 units, respectively. The same network topology can be trained separately on synthetic, or real-world data.

OctoPath - extending to 3D representation

Since motion planning can also be viewed as a sequence to sequence mapping problem, or as a sequence generation task, RNNs have been proposed for modeling the driving trajectories. Different from conventional neural networks, RNN contain a time dependent feedback loop in its memory cell. In order to use RNNs for predicting a future trajectory, each separate point is considered a state, which further implies that the whole trajectory is represented as a sequence. The transition from one state to any another is strictly constrained by the topology of the network].

Most of the RNN solutions proposed for solving the task of trajectory estimation need a discrete environment model. In this thesis, the proposed environment model is based on octrees and uses probabilistic occupancy estimation. The main advantages of using this model are that it explicitly represents not only occupied space but also free and unknown areas and that it enables a compact memory representation and configurable resolutions. As opposed to Neuro-trajectory, which has been presented in the previous section, the world is now sensed in 3D using an octree representation, and we no longer use convolutional layers for processing the input sequences, as this intermediate representation has been taken over by the fixed state vector between the encoder and the decoder of our architecture.

In this thesis, the path planning component from the perception-planning-action pipeline is addressed. The currently proposed method, coined OctoPath, is self-supervised and aims to combine the configurable resolution of an octree-based environment model with a classification-based encoder-decoder RNN architecture. It takes as input a sequence of sensor measurements, together with the current segment of a reference trajectory, building upon the RNN encoder-decoder architecture which has shown excellent performance for sequence-to-sequence tasks.

OcTrees

Most robotic applications require an environment model that includes free, occupied, and unmapped zones and is efficient in terms of runtime and memory use. Range measurement mistakes are common in sensor models, and reflections or dynamic barriers can generate seemingly random results. When building an accurate model of the environment from noisy data, the underlying uncertainty must be taken into consideration. Multiple erroneous readings can then be merged to produce a credible estimate of the real condition of the environment.

The octree data structure represents three-dimensional objects. An octree is a spatial decomposition in which the tree's root is recursively split by two in each coordinate direction until each cell can contain a maximum number of items. In other words, it is a hierarchical data structure for 3D spatial subdivision that is most frequently used to recursively subdivide a given 3D region into eight octants. The octree organizes its items hierarchically, avoiding the depiction of empty space.

$$O_k : [0,1]^d \rightarrow D_k \subset \mathbb{R}^3 \tag{45}$$

The root node or cell is the initial node of an octree. The root node or cell points to eight elements or cells, each of which may point to another eight elements or cells, and so on. Every node on an octree is the space of a cubic volume known as a voxel, and the minimum voxel size determines the octree's textitresolution. The last level reached is known as the leaf level, and it contains the leaf components or cells. If every cell above the leaf level points to a cell, the octree is said to be complete. If the inner nodes are kept, the tree may be chopped down at any level to get a more coarse subdivision. Octrees avoid one of the major flaws of fixed grid systems in robotic mapping: the fact that the environment should not be known beforehand, and that the environmental model only comprises the measured volume.

When referring to a laser range finder, for example, the endpoints of the sensor generate occupied space, while the detected region between the sensor and the endpoint is considered to be free space. The occupied space is mapped from the point cloud data packets at the corresponding distance in space for our input LiDAR data. As a result, we use LiDAR data to generate an octree environment model, which depicts free-space (driving area) and inhabited areas in three dimensions.

A central property of our approach is that it allows for efficiency of occupied and free space while keeping the memory consumption low, which is essential for our model car hardware. The octrees have fixed sizes, as required by the neural network input, based on the field of view of the LiDAR sensor. The nodes which are neither occupied nor free (these are always beyond the detected obstacles)

are marked as unknown and initialized with zero 0 to prevent them from influencing the inference result. Additionally, we can configure the resolution to a lower value, to reduce the processing times and memory usage even further. Tree structures are the primary methods used in prior point cloud compression algorithms. Numerous approaches store data in an octree and perform entropy coding with hand-crafted entropy models such as adaptive histograms, parent context, and estimations based on planar approximations or neighbour proximity

In its most basic form, octrees can be used to model a Boolean property. In the context of robotic mapping, this is usually the occupancy of a volume. If a certain volume is measured as occupied, the corresponding node in the octree is initialized. Any uninitialized node could be free or unknown in this Boolean setting. To resolve this ambiguity, we explicitly represent free volumes in the tree. These are created in the area between the sensor and the measured end point, e.g., along a ray determined with raycasting. Areas that are not initialized implicitly model unknown space. Using Boolean occupancy states or discrete labels allows for compact representations of the octree: If all children of a node have the same state (occupied or free) they can be pruned. This leads to a substantial reduction in the number of nodes that need to be maintained in the tree.

In terms of data access complexity, octrees require an overhead compared to a fixed-size 3D grid due to the tree structure. A single, random query on a tree data structure containing n nodes with a tree depth of d can be performed with a complexity of O(d) = O(logn). Traversing the complete tree in a depth-first manner requires a complexity of O(n). Note that, in practice, our octree is limited to a fixed maximum depth dmax. This results in a random node lookup complexity of O(dmax) with dmax being constant. Therefore, for a fixed depth dmax, the overhead compared to a corresponding 3D grid is constant.

Octrees can be represented in a hierarchical tree structure or a pointerless linear tree structure. A hierarchical tree structure's cell list includes the root cell, intermediate cells and leaf cells. Each cell includes pointers to parent and child cells. The point location and neighbor search operations would follow pointers to traverse the tree. A linear tree structure's cell list includes only leaf cells. Each cell has a locational code which is used as a search key for cell location. Locational codes are composed of the cells' minimum coordinates.

Octomap

OctoMap uses a probabilistic approach to map the environment. Instead of binary occupancy grids that mark areas as either free or occupied, OctoMap assigns a probability value to each node, indicating the likelihood that the volume is occupied. This probabilistic representation accounts for sensor noise and uncertainty, leading to more robust and accurate maps.

The hierarchical nature of the octree allows OctoMap to represent large environments efficiently. Higher-level nodes cover larger volumes and can be subdivided into smaller, more detailed nodes as needed. This approach enables the framework to adapt its resolution dynamically, providing high detail in areas of interest while maintaining coarse representation in less critical regions.

The mathematical framework of OctoMap is based on Bayesian probability theory, which provides a systematic way to update the occupancy probabilities of nodes based on sensor measurements.

The occupancy probability of a node is updated using the Bayesian update rule. Given an initial probability $P(O)$ and a new measurement with probability $P(M|O)$, the updated probability $P(O|M)$ is computed as:

$$P(O|M) = \frac{P(M|O)P(O)}{P(M)} \tag{46}$$

Where:

- $P(O)$ is the prior probability of the node being occupied.
- $P(M|O)$ is the likelihood of the measurement given the node is occupied.
- $P(M)$ is the probability of the measurement.

To simplify the computation, OctoMap uses the log-odds representation of probabilities. The log-odds value l of a probability p is defined as:

$$l = \log\left(\frac{p}{1-p}\right) \tag{47}$$

Using log-odds, the Bayesian update rule becomes a simple additive operation:

$$l(O|M) = l(O) + l(M|O) - l(M) \tag{48}$$

This representation allows for efficient and numerically stable updates. The implementation of OctoMap involves several key components and algorithms to build and maintain the octree structure. When a new sensor measurement is received, it is inserted into the octree. The measurement is typically a 3D point cloud obtained from sensors such as LIDAR or stereo cameras. Each point in the cloud updates the occupancy probabilities of the nodes it intersects, based on the sensor model.

Raycasting is used to update the occupancy probabilities along the sensor rays. For each point in the point cloud, a ray is cast from the sensor origin to the point. Nodes along the ray are updated to reflect the probability of being free, while the endpoint node is updated to reflect the probability of being occupied. This approach ensures that free space is accurately modeled. OctoMap supports dynamic updates, allowing the map to change as the environment changes. Nodes can be added, removed, or updated based on new measurements. This capability is crucial for environments where objects may move or appear/disappear over time.

For autonomous navigation, OctoMap provides a detailed 3D representation of the environment, enabling path planning and obstacle avoidance. Robots can use the map to navigate through complex and cluttered spaces, avoiding obstacles and finding optimal paths. The hierarchical octree structure allows OctoMap to represent large environments efficiently, using less memory and computational resources compared to voxel grids. The ability to dynamically adjust the resolution further enhances efficiency.

The probabilistic approach of OctoMap accounts for sensor noise and uncertainty, leading to more accurate maps. The use of log-odds for probability updates ensures numerical stability and efficient computations. OctoMap's ability to handle dynamic environments and integrate various sensor models makes it highly flexible. It can be adapted to different types of sensors and scenarios, providing robust performance across a wide range of applications.

OctoMap is a powerful and versatile framework for 3D mapping in robotics and autonomous systems. Its probabilistic, hierarchical representation of the environment enables efficient, accurate, and flexible mapping, making it suitable for a wide range of applications. Whether used for autonomous navigation, robotic manipulation, exploration, or SLAM, OctoMap provides the tools needed to build and maintain detailed 3D maps in real-time. As research and development in robotics continue to advance, OctoMap remains a crucial component for creating intelligent and capable autonomous systems.

RNN Encoder-Decoder

In contrast to traditional neural networks, an RNN's memory cell comprises a time-dependent feedback loop. A recurrent neural network itself can be "unrolled" $\tau_i + \tau_o$ times to produce a loop-free architecture that matches the input length, if we consider an input sequence $[x^{<t-\tau_i>}, ..., x^{<t>}]$ which is time dependant, together with an output sequence $[y^{<t+1>}, ..., y^{<t+\tau_o>}]$. Unrolled networks have $\tau_i + \tau_o + 1$ similar or even identical layers, which means that each layer has the same learned weights.

This architecture is comprised of two models: a stack of several recurrent units for reading the input sequence and encoding it into a fixed-length vector, and a second one for decoding the fixed-length vector and outputting the predicted sequence. The combined models are known as an RNN Encoder-Decoder, which is designed specifically for sequence to sequence problems. Given the input sequence $\vec{X}^{<t-\tau_i,t>}$, a basic RNN encoder computes the sequence of hidden states $(h_1,\ h_2,\ h_3,\ \ldots,\ h_N)$:

$$h_t = \tanh\left(U_{xh}x_t + U_{hh}h_{t-1}\right), \tag{49}$$

where the two matrices U_{xh} and U_{hh} are the weight matrix between the input layer and hidden layer, and the weight matrix of recurrent connections in a given hidden layer, respectively.

The vanishing gradient experienced during training is the major challenge when using simple RNNs. The gradient signal can be multiplied an infinite number of times, up to the number of time steps. As a result, a classical RNN cannot capture long-term dependencies in sequence data. The gradient of the network's output will have a hard time propagating back to affect the weights of the earlier layers if the network is very deep or processes long sequences. The weights of the network will not be successfully modified as a result of gradient vanishing, resulting in very small weight values.

To counter these challenges, in our work, we use a set of Long Short-Term Memory (LSTM) networks for both the encoder and the decoder. LSTMs solve the vanishing gradient problem by adding

three gates that control the input, output, and memory state, as opposed to classical recurrent neural networks.

$\Theta = [W_i, U_i, b_i]$ parametrizes an LSTM network, where W_i embodies the weights of the gates and memory cells multiplied with the input state, U_i represents the weights controlling the network's activations, and b_i contains the bias values of the neurons. A network output sequence is defined as a desired ego-vehicle optimal trajectory:

$$Y^{<t+1,t+\tau_o>} = [y^{<t+1>}, y^{<t+2>}, ..., y^{<t+\tau_o>}], \tag{50}$$

where $y^{<t+1>}$ is a predicted trajectory set-point at time $t+1$. τ_i and τ_o are not necessarily equal: $\tau_i \neq \tau_o$.

The LSTM encoder takes the latest octree samples $\vec{X}^{<t-\tau_i,t>}$, as well as the reference trajectory sequence $\vec{Z}_{ref}^{<t-\tau_i,t+\tau_o>}$ for the current time step t, and produces an intermediate fixed-size vector c_t that preserves the temporal correlation of the previous observations. The hidden state of the LSTM encoder h_t is calculated using the following equations:

$$z_t = \sigma(U_{xz}x_t + U_{hz}h_{t-1}), \tag{51}$$

$$r_t = \sigma(U_{xr}x_t + U_{hr}h_{t-1}), \tag{52}$$

$$\tilde{h}_t = \tanh(U_{xh}x_t + U_{rh}(r_t \otimes h_{t-1})), \tag{53}$$

$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes \tilde{h}_t, \tag{54}$$

where σ represents the sigmoid activation function. z_t, r_t, and $\tilde{h}_t$ are the update gate, reset gate, and candidate activation, respectively. U_{xz}, U_{xr}, U_{xh}, U_{hz}, U_{hr}, and U_{rh} are the related weight matrices. The notation $\otimes$ represents an element-wise multiplication operator.

The LSTM decoder takes the predicted trajectory sample to produce the subsequent trajectory samples, producing the entire future trajectory $\vec{Y}^{<t+1,t+\tau_o>}$ for the current time step, given the context vector c_t as input. $\vec{Y}^{<t+1,t+\tau_o>}$ is defined as a sequence variable Y with data instances $[y^{<t+1>}, ..., y^{<t+\tau_o-1>}, x^{<t+\tau_o>}]$ in a specific time interval $[t+1, t+\tau_o]$. Each predicted sequence variable's probability is calculated as follows:

$$p(y_t|X, y_{t-1}) = g\left(U_o(Ey_{t-1} + U_s s_t + U_c c_t)\right), \tag{55}$$

where g is a softmax activation function. s_t is the current hidden state of the decoder, and y_{t-1} represents the previous target symbol, while E denotes the embedding matrix.

The earlier target sequence variable y_{t-1} and the context vector ct are also inputs to the decoder, which uses a single unidirectional layer to compute the hidden state st:

$$z'_t = \sigma(U_{yz}Ey_{t-1} + U_{sz}s_{t-1} + C_{cz}c_t), \tag{56}$$

$$r'_t = \sigma(U_{yr}Ey_{t-1} + U_{sr}s_{t-1} + C_{cr}c_t), \tag{57}$$

$$\tilde{s}_t = \tanh(U_{ys}Ey_{t-1} + U_{rs}(r'_t \otimes s_{t-1}) + C_{cs}c_t), \tag{58}$$

$$s_t = (1 - z'_t) \otimes s_{t-1} + z'_t \otimes \tilde{s}_t, \tag{59}$$

where z'_t, r'_t, and $\tilde{s}_t$ are the update gate, reset gate, and candidate activation, respectively. U_{xx} and C_{xx} are the related weight matrices.

The decoder retains the best sequence candidates in the algorithm when creating the future trajectory sample for each time step. As a result, using the octree input framework, the proposed model would predict the most likely hypotheses of the vehicle trajectory. As analogy to machine translation problems, a point coordinate inside an octree is a character, an octree is a word, and the sequence of input octrees represents an sentence. Our experiments show that an encoder-decoder RNN produces an acceptable trajectory and that its prediction accuracy is improved in comparison to traditional prediction methods.

Performance evaluation

OctoPath was compared to the baseline hybrid A* algorithm, to a regression-based approach, and to a CNN learning-based approach. We put the OctoPath algorithm to the test in two distinct environments: (I) in the GridSim simulator and (II) in a real-world navigation environment, both indoor and outdoor, using the the AMTU robot. The AMTU is an AgileX Scout 2.0 platform which acts as a 1:4 scaled car, equipped with a 360° Hesai Pandar 40 Lidar, 4$\times$ e-130A cameras providing a 360° visual perception of the surroundings, a VESC inertial measurement unit, GPS, and an NVIDIA AGX Xavier board for data processing and control. The state of the vehicle was measured using wheels odometry and the Inertial Measurement Unit (IMU).

All experiments aimed at solving the trajectory estimation problem, which was to calculate a trajectory for safely navigating the driving environment without performing the motion control task. To implement motion control, the predicted states were used as input to a model predictive controller, which computed the necessary v_x and ω_z control signals for the the AMTU. The motion controller's design and implementation are beyond the scope of this paper.

The hybrid A* algorithm employs a modified state-update rule to apply a variant of the well-known A* algorithm to the vehicle's octree environment model. The search space (x, y, θ) is discretized, just like in traditional A*, but unlike A*, which only allows visiting cell centers, the hybrid version of the algorithm associates a more continuous state of the car with each grid cell, allowing also trajectory points that are not in the exact center of the octree cell.

In the case of trajectory prediction as a regression problem, the goal is to achieve a direct prediction of continuous future positions without any discretization. Because the average prediction minimizes the regression error, such methods have a bias to output the average of several options, thus rendering it inaccurate.

The Neural RRT* algorithm, is a novel optimal path planning algorithm based on convolutional neural networks. It used the A* algorithm to generate training data, considering map information as input, and the optimal path as ground truth. Given a new path planning problem, the model can quickly determine the optimal path's probability distribution, which is then used to direct the RRT* planner's sampling operation. The performance of the algorithm varies under different values of the clearance to the obstacles and step size. A wider clearance indicates that the planned route is far

from the obstacles, while a smaller clearance indicates that the planned path is closer to them. We have used a fixed step size of 2 and a robot clearance value of 4.

We use the Root Mean Square Error (RMSE) between the predicted and the recorded trajectory in the 2D driving plane:

$$RMSE = \sqrt{\frac{1}{\tau_o}\sum_{t=1}^{\tau_o}\left[(\hat{p}_x^{<t>} - p_x^{<t>})^2 + (\hat{p}_y^{<t>} - p_y^{<t>})^2\right]}, \tag{60}$$

where $\hat{p}_x^{<t>}$, $\hat{p}_y^{<t>}$ are the points on the predicted trajectory, and $p_x^{<t>}$, $p_y^{<t>}$ are the points on the ground truth trajectory, respectively. We set the prediction horizon $\tau_o = 10$.

The workflow of the experiments is as follows:

- collect training data from driving recordings;
- generate octrees and format training data as sequences;
- train the OctoPath deep network;
- evaluate on simulated and real-world driving scenarios.

This experimental setup resulted in 15 km of driving in GridSim, over 1 km of looped indoor navigation and over 2 km of outdoor navigation outside of Transilvania University of Brasov's IHTPSD (Institute of High Tech Products for Sustainable Development). The robot navigated indoor and outdoor environments while avoiding static and dynamic obstacles.

Table 1: Errors between estimated and ground truth trajectories in simulation and real-world navigation testing scenarios.

Scenario	Method	$\bar{e}_x[m]$	max$(e_x)[m]$	$\bar{e}_y[m]$	max$(e_y)[m]$	$RMSE[m]$
GridSim	Hybrid A*	1.43	3.21	2.71	4.01	2.71
simulation	Regression	3.51	7.20	4.71	8.53	5.10
	Neural RRT	1.27	3.01	2.35	2.98	2.48
	Octopath	1.16	2.31	1.72	2.75	2.07
Indoor	Hybrid A*	1.21	4.33	1.33	3.88	1.74
navigation	Regression	1.90	5.73	2.31	4.98	2.75
	Neural RRT	1.01	3.29	0.98	2.16	1.44
	Octopath	0.55	1.08	0.44	0.87	0.69
Outdoor	Hybrid A*	1.35	4.67	1.44	4.44	1.98
navigation	Regression	2.41	8.42	2.77	8.98	3.01
	Neural RRT	1.05	2.52	1.06	3.24	1.17
	Octopath	0.71	1.46	0.57	1.17	0.88

GridSim is a self-driving simulation engine that generates synthetic occupancy grids from simulated sensors using kinematic models, which are then used to produce input octree data. The user

interface was integrated into the GridSim environment menu, such that the modes can be switched between replay, record, and training, with each one having access to the five different scenarios. There is a large number of configurable parameters, such as the resolution of the simulator, occupancy grid precision, number of traffic participants, ego vehicle's size, maximum speed, or turning radius.

The goal is to get from a starting position to a given destination while avoiding collisions and driving at the desired speed. The Z coordinate of all obstacle and free space points is set to zero to adjust the encoder-decoder network's input data to the GridSim environment. The testing scenarios generated using the GridSim simulation environment were not used during the training of the network.

For the various types of roads and traffic environments found in the synthetic testing database, the performance assessment of the benchmarked algorithms is summarized in the top part of the Table 1. The mean position errors ($\bar{e}_x$, $\bar{e}_y$), as well as the RMSE metric from Equation (60) is illustrated.

The indoor navigation experiment was performed using the AMTU skid steer wheeled mobile robot vehicle, with different indoor navigation tasks. The reference routes which the car had to follow were composed of straight lines, S-curves, circles, and a 75 m track on the main hallway of Transilvania University of Brasov's Institute for Research.

The testing room for the indoor experiment was the same as the one that was used for gathering training data, but the reference routes and the obstacles were placed differently. The main hallway was not used for gathering training data.

The first set of 10 trials were performed without any obstacles present on the reference routes, while the second 10 trials set contained static and dynamic obstacles. Fifty-four thousand training samples have been collected in the form of LiDAR data and vehicle states. The path driven when collecting data was considered as a reference trajectory and was created in a self-supervised manner.

The outdoor navigation experiment was performed outside of Transilvania University of Brasov's IHTPSD (Institute of High Tech Products for Sustainable Development). The reference route which the car had to follow was composed of a full loop around the institute and was created using a GPS tool. The route itself is around 500 m long, and we ran it 4 times.

The outdoor reference path which was used for training the network was recorded as the driven path when collecting the sensory data. When testing the network, the reference path was generated using our vehicle mission planner tool. The static obstacles were mainly parked cars, while the dynamic obstacles were moving cars or people.

The mean and standard deviation of the position error (computed as RMSE), left side for indoor navigation, and right side for outdoor navigation. The position errors are shown in Table 1 for all scenarios: simulation, indoor navigation and outdoor navigation. The mean ($\bar{e}_x$, $\bar{e}_y$) and maximum (max(e_x), max(e_y)) position errors, as well as the RMSE metric from Equation (60), are shown. When compared to OctoPath, Neural RRT has the lowest deviations, but, from the non-learning approaches, Hybrid A* performs the best, indicating that it is a good candidate for non-learning trajectory estimation.

In the performed experiments, the hybrid A* algorithm behaved better than the regression ap-

proach, mostly because of the structure of the octree environment model input data. This makes A* strictly dependent on the precision of the obstacle representation in the surrounding environment. Besides, the jittering effect of OctoPath may be a side effect of the decoder output's discrete nature. It will, however, provide a reliable ego-vehicle trajectory prediction over a given time horizon.

The e-CAM130A synchronized quad cameras will be used in future research to perform a full semantic segmentation on the received point cloud and to extend the validation of our approach to more use-cases. Learning-based approaches have proven that they can deliver better results in the long run than conventional methods. This improvement would be achieved by training on more data, which would include a greater number of corner cases.

Personal contributions

Based on the objectives mentioned at the beginning, the personal contributions fall into the same four categories, which are detailed as follows:

1. Comprehensive literature review: particular emphasis on deep learning techniques and highlighting their transformative impact on the autonomous driving field.
2. Simulation environment development: providing a controlled environment for testing and validating the algorithms, ensuring safe and repeatable experimentation.
3. Algorithm development: sensor fusion for enhanced perception and learning-based path planning algorithms.
4. Evaluation and benchmarking: Comprehensive evaluation and benchmarking against established methods, proving the efficacy and reliability of the developed methods.

Comprehensive literature review

The review successfully outlines the transformative potential of autonomous driving, emphasizing its impact on safety, efficiency, and convenience in transportation. It highlights the rapid advancements in technology, such as sensor fusion, machine learning, and path planning, which are critical for the development of autonomous vehicles. The chapter also identifies key challenges, including environmental robustness, human-vehicle interaction, regulatory frameworks, and cybersecurity. By addressing these challenges through interdisciplinary research and collaboration, the chapter sets a solid foundation for the thesis, aiming to advance the field of autonomous driving through innovative path planning algorithms and sensor integration techniques.

To summarize, the main contributions of this category are:

- Conducted a thorough literature review to identify and evaluate existing methodologies and technologies in autonomous navigation and path planning, highlighting key advancements and persistent challenges.
- Identified critical gaps and limitations in current research, providing a foundation for the development of innovative solutions in autonomous vehicle navigation.

Simulation environment development

This section details the experimental platforms developed and utilized during the thesis research. It covers the creation of the GridSim simulation environment, which was instrumental in testing the proposed deep learning approaches for autonomous driving. It also describes the development and testing of the Raspberry Pi Robot prototype and the AMTU platform, highlighting their roles in validating the proposed algorithms. By providing a robust simulation and testing environment, it ensures that the path planning algorithms can be rigorously evaluated under realistic conditions, thereby enhancing their reliability and performance in real-world scenarios.

The primary contributions of this category are:

- Developed a robust simulation tool (namely GridSim) designed to emulate real-world driving conditions, facilitating the controlled testing and validation of proposed algorithms.
- Ensured the simulation tool accurately represents diverse and dynamic real-world scenarios, enhancing the reliability of algorithm testing.
- Development of the prototype Raspberry-PI Robot and of the main robotic platform used for experiments, the AMTU skid-steered wheeled mobile robot.

Algorithm development

This section covers the development of path planning and behavior arbitration algorithms for autonomous vehicles. It introduces the concept of grid-based representation and addresses the challenges of navigating complex environments. It elaborates on the behavior arbitration mechanism, which involves driving scene description, analysis, and modeling. The Neuro-Trajectory approach, a multi-objective neuroevolutionary method, is presented as a solution for local state trajectory learning. By effectively combining cognitive learning tasks with neuroevolutionary training, this chapter contributes significantly to advancing the capabilities of autonomous vehicles in dynamic and unpredictable environments.

Afterwards, it extends the research to a 3D representation using OcTrees, enhancing the spatial perception and path planning capabilities of autonomous vehicles. It discusses the implementation of the OctoPath system, a self-supervised encoder-decoder neural network designed for predicting optimal vehicle paths in 3D environments. By transitioning from 2D to 3D representations, this chapter

demonstrates a significant improvement in the accuracy and reliability of path planning algorithms, contributing to more robust and efficient autonomous driving solutions.

In summary, this category's primary contributions are:

- Designed and implemented a comprehensive sensor fusion framework integrating data from LiDAR, cameras, and IMUs.
- Demonstrated significant improvements in perception accuracy and reliability, contributing to the autonomous vehicle's situational awareness in behavior arbitration.
- Developed, implemented, and rigorously tested a novel learning-based algorithm for efficient and accurate path planning in 2D grids
- Developed OctoPath, a self-supervised encoder-decoder deep neural network that predicts the optimal path for the vehicle, efficiently handling 3D spatial data, allowing for accurate and scalable trajectory predictions in complex environments.

Evaluation and benchmarking

The key contributions of this category are:

- Developed a secure and efficient development environment on the NVidia AGX Xavier board, ensuring optimal performance and security of the implemented algorithms.
- Established various performance metrics and testing scenarios to rigorously evaluate the reliability and efficiency of the developed algorithms.
- Ensured the proposed algorithms perform optimally across diverse datasets and adapt effectively to different driving conditions, demonstrating versatility and robustness.
- Highlighted the practical implications of the research, showcasing its potential to enhance the safety, efficiency, and reliability of autonomous vehicles in real-world applications.

Dissemination of research results

The dissemination of the research results has been pivotal in advancing the field of autonomous vehicles and intelligent systems. Several patents have been filed regarding some of the technologies and methodologies developed during this research:

- Grigorescu, S., Trasnea, B., Vasilcoi A., Training of a convolutional neural network, World Intellectual Property Organization, Patent no. WO2021008798A1, 2020.

- Grigorescu, S., Macesanu, G., Cocias, T., Trasnea, B., Ginerica, C., Generating training images for machine learning-based object recognition systems, European Patent Office, Patent no. EP3343432A1, 2018.
- Vasilcoi A., Radu P., Marina L., Trasnea, B., Grigorescu, S., Convolutional neural network with reduced complexity, European Patent Office, Patent no. EP3343432A1, 2020.
- Trasnea, B., Grigorescu, S., Trajectory estimation for vehicles, European Patent Office, Patent no. EP3839830A1, 2021.

Several publications in international scientific journals and conference proceedings resulted from the work done during the PhD studies. Three research papers have been published as first author, and important contributions to another eight have been made, as follows:

- Trasnea, B., Ginerica, C., Zaha, M., Macesanu, G., Pozna, C. and Grigorescu, S., OctoPath: An OcTree-Based Self-Supervised Learning Approach to Local Trajectory Planning for Mobile Robots. Sensors, 21(11), p.3606, 2021. DOI: 10.3390/s21113606
- Trasnea, B., Pozna, C. and Grigorescu, S.M., AIBA: An AI Model for Behavior Arbitration in Autonomous Driving. In Multi-disciplinary Trends in Artificial Intelligence: 13th International Conference, MIWAI 2019, Kuala Lumpur, Malaysia, Proceedings (Vol. 11909, p. 191). Springer Nature, November 17–19, 2019. DOI: 10.1007/978-3-030-33709-417
- Trasnea, B., Marina, L.A., Vasilcoi, A., Pozna, C.R. and Grigorescu, S.M., GridSim: A vehicle kinematics engine for deep neuroevolutionary control in autonomous driving. In 2019 Third IEEE International Conference on Robotic Computing (IRC) (pp. 443-444). IEEE, February 2019. DOI: 10.1109/IRC.2019.00091
- Grigorescu, S., Trasnea, B., Cocias, T. and Macesanu, G., A survey of deep learning techniques for autonomous driving. Journal of Field Robotics, 37(3), pp.362-386, 2020. DOI: 10.1002/rob.21918
- Marina, L.A., Trasnea, B. and Grigorescu, S.M., A multi-platform framework for artificial intelligence engines in automotive systems. In 2018 22nd International conference on system theory, control and computing (ICSTCC) (pp. 559-564). IEEE, October 2018. DOI: 10.1109/icstcc.2018.8540753
- Marina, L.A., Trasnea, B., Cocias, T., Vasilcoi, A., Moldoveanu, F. and Grigorescu, S.M., Deep Grid Net (DGN): A deep learning system for real-time driving context understanding. In 2019 Third IEEE International Conference on Robotic Computing (IRC) (pp. 399-402). IEEE, February 2019. DOI: 10.1109/IRC.2019.00073
- Grigorescu, S.M., Trasnea, B., Marina, L., Vasilcoi, A. and Cocias, T., Neurotrajectory: A neuroevolutionary approach to local state trajectory learning for autonomous vehicles. IEEE Robotics and Automation Letters, 4(4), pp.3441-3448, 2019. DOI: 10.1109/lra.2019.2926224

- Grigorescu, S., Cocias, T., Trasnea, B., Margheri, A., Lombardi, F. and Aniello, L., Cloud2Edge Elastic AI Framework for Prototyping and Deployment of AI Inference Engines in Autonomous Vehicles. Sensors, 20(19), p.5450, 2020. DOI: 10.3390/s20195450

- Grigorescu, S., Zaha, M., Trasnea, B. and Ginerica, C., Embedded Vision for Self-Driving on Forest Roads. IEEE Computer Vision and Pattern Recognition conference (CVPR), Workshop Demo on Embedded Vision. June 2021.

- Grigorescu, S., Ginerica, C., Zaha, M., Macesanu, G. and Trasnea, B., LVD-NMPC: A Learning-based Vision Dynamics Approach to Nonlinear Model Predictive Control for Autonomous Vehicles. International Journal of Advanced Robotic Systems. May 2021. DOI: 10.1177/17298814211019544

- Ginerica, C., Zaha, M., Gogianu, F., Busoniu, L., Trasnea, B. and Grigorescu, S. ObserveNet Control: A Vision-Dynamics Learning Approach to Predictive Control in Autonomous Vehicles. IEEE Robotics and Automation Letters, 6(4), pp.6915-6922, 2021. DOI: 10.1109/LRA.2021.3096157

Parts of the research results were also presented at the 2019 European Robotics Forum in Bucharest, Romania, where over 50 exhibitors displayed their prototypes, goods, and services and visitors learned about Europe's most sophisticated robotics industry, research institutes, and projects.

Conclusions and future work

The research began with an extensive review of technological advancements and methodologies underpinning autonomous driving systems, focusing on the transformative impact of deep learning techniques. It detailed the application of Convolutional Neural Networks (CNNs) for object detection, image recognition, and scene segmentation, and Recurrent Neural Networks (RNNs) for sequence prediction and behavior modeling. Furthermore, Deep Reinforcement Learning (DRL) is explored for developing policies that enable autonomous vehicles to learn optimal driving strategies through environmental interaction.

Applications of these technologies have been thoroughly examined, particularly in driving scene understanding, path planning, behavior arbitration, and sensor fusion. The practical implementation employs Python frameworks like TensorFlow, Keras, and PyTorch to develop and deploy deep learning models. The study then transitioned to experimental platforms, starting with the development of the GridSim Simulator for testing autonomous driving algorithms in a controlled environment.

The primary robotic platform, AMTU, equipped with e-Cam130A quad cameras and a 360-degree Hesai Pandar LiDAR, validated the algorithms under realistic conditions. The thesis details the iterative development and experimental process, integrating and testing key algorithms on these platforms to bridge the gap between theoretical research and practical deployment.

The research culminated in the advancement of autonomous vehicle trajectory prediction through the NeuroTrajectory and OctoPath algorithms. NeuroTrajectory uses a neuroevolutionary approach with multiobjective Pareto optimization to learn local state trajectories from occupancy grids. OctoPath, employing self-supervised learning with an octree-based environment model, predicts local trajectories efficiently, demonstrated to operate in real-time on the Nvidia AGX Xavier.

OctoPath's adaptability to different environmental resolutions and its capability to handle three-dimensional spatial data highlighted its effectiveness in dynamic environments. Comparative analyses validated OctoPath as a robust method for local trajectory prediction, emphasizing its practical applicability in both simulation and real-world scenarios.

The research projects completed throughout the PhD program show that deep learning has a lot of potential in the automotive and robotics industries. However, based on current research and this thesis' contributions, there are a few challenges and constraints that need to be solved. Path planning remains a critical challenge in autonomous vehicle navigation. Future research should focus on developing more sophisticated algorithms that can handle dynamic and complex environments. Potential areas include:

- Multi-agent path planning: exploring coordination strategies among multiple autonomous vehicles to optimize traffic flow and reduce congestion.
- Enhanced sensor fusion techniques: investigate new methods for combining data from diverse sensors (e.g., LiDAR, radar, cameras) to improve environmental perception and object detection.
- Energy-efficient algorithms: designing algorithms that maximize computational efficiency while minimizing energy consumption, crucial for electric vehicles.
- High-performance computing integration: leveraging advancements in hardware, such as GPUs and FPGAs, to enhance the processing capabilities of embedded systems.
- Latency reduction: developing low-latency communication protocols to ensure timely data exchange between vehicles and infrastructure.
- Robustness in adverse conditions: develop algorithms that can maintain high performance in challenging conditions such as heavy rain, fog, or snow.
- Predictive maintenance algorithms: developing machine learning models that predict and prevent system failures before they occur.

Additionally, greater consideration should be given to combining and optimizing deep neural network designs. In addition, alternative hybrid loss functions and training techniques, such as adversarial setup, should be investigated. In conclusion, the future of autonomous vehicle research is rich

with opportunities for innovation and improvement. By addressing these directions, researchers can contribute to the development of safer, more efficient, and user-friendly autonomous systems.