Universitatea
Transilvania
din Brașov

INTERDISCIPLINARY DOCTORAL SCHOOL

Faculty of Electrical Engineering and Computer Science

Cosmin George GINERICĂ

Navigation of autonomous robots using artificial intelligence techniques

SUMMARY

Scientific supervisor

Prof.Dr. Eng. Sorin Mihai GRIGORESCU

BRAȘOV, 2024

Table of contents

Doctoral thesis subject 3
Research objectives 4
Thesis outline 5
Research methodology 6
Sensorial observations processing 6
Mobile robot localization 7
Sensory observations model 10
Local trajectory planning 11
Mobile robot control 11
ObserveNet Control particularities for RGB-D data 12
Experimental results 14
GridSim simulated environment 14
CARLA simulated environment 15
CARLA simulated environment – Autonomous Driving Learderboard 17
AgileX Scout mobile robot 17
Unitree A1 mobile robot 19
Control algorith ablation study 20
RGB-D data prediction ablation studies 20
Conclusions and future work 21
Original contributions. Published work 22

Doctoral thesis subject

The present work proposes an autonomous navigation system for mobile robots named ObserveNet Control. It is based on the perception-plan-control paradigm to achieve specific tasks of autonomous navigation: perceiving and understanding the surrounding environment, planning the trajectory for the robot, and controlling it to follow the generated trajectory. The fundamental field covered by this thesis is Mechatronics and Robotics. Additionally, notions and techniques specific to the fields of Computer Vision and Artificial Intelligence are utilized.
In this paper, an artificial intelligence algorithm called ObserveNet Control is proposed, based on a recurrent neural network (RNN), which receives sequences of sensory observations from a vehicle as inputs and based on which it models and predicts the working environment. Based on this model, a mobile robot autonomously navigates its surroundings. All three necessary submodules for autonomous navigation have been defined: the perception module, the trajectory planning module, and the control module.
The local trajectory planning module, responsible for generating the path that the mobile robot should navigate while avoiding collisions with obstacles in the scene, is based on the Dynamic Window Approach, modified to take into account the predictions of sensory observations from the ObserveNet architecture.
The actual control of the robot is performed using an NMPC (Nonlinear Model Predictive Control) controller. The algorithm has been tested both with the help of a simulator used to validate various autonomous navigation technologies and in a real environment, using a quadruped robot and a differential-drive autonomous vehicle.
Localization data (the current state of the autonomous vehicle) and observations recorded by sensors on the vehicle are stored in an augmented memory. These are then partitioned into sequences of predetermined length, to be used both for the inference of the recurrent neural network, which models the dynamics of the surrounding environment, and as inputs for the NMPC controller, which generates the desired trajectory.
Although there are systems that can perform the task of autonomous driving already equipped on vehicles on public roads, they are not fully functional. On the other hand, autonomous driving refers, in addition to the scenario of vehicles navigating on roads, to the field of robotics, where the fulfillment of this task is also required.
At a high level, both for mobile robots and vehicles, the problem of autonomous navigation refers to the intelligent agent's ability to navigate from the starting point to the destination point, while respecting a set of physical and imposed constraints, simultaneously avoiding collisions with static or dynamic obstacles present in the agent's working environment.
Evidently, the task of autonomous navigation involves a series of other tasks that need to be solved by the intelligent agent, grouped into three fundamental components: the perception of the intelligent agent, which consists of understanding the surrounding environment in which it operates, trajectory planning, both global and reference, and local, which allows the incorporation of a certain level of logic necessary for movement, such as obstacle avoidance, and the control component, which is responsible for carrying out the movement of the agent based on command signals generated by the intelligent component of the agent.

The three main components of an autonomous system must be able to perform their respective functionalities under various operating conditions, depending on the application for which the autonomous navigation system is implemented. For example, an autonomous vehicle navigating in traffic is subject to many disturbing factors, such as varied lighting conditions, poor weather conditions. At the same time, this system must take into account many variables from the surrounding environment, such as other vehicles, pedestrians, cyclists, traffic lights, faded road markings, etc. On the other hand, a mobile robot navigating in an unstructured environment faces specific challenges of this scenario, such as the unpredictable dynamics of obstacles in the surrounding environment, constant changes in the environment, and various impediments that hinder safe navigation.

Returning to the high-level description of the problem of autonomous navigation, it can be formulated as follows: the intelligent agent must be able to follow the global reference trajectory $z_{ref}^{<t-\infty,t+\infty>}$ as closely as possible, but without causing collisions with other agents or with static obstacles in the surrounding environment. In order to avoid collisions, a local trajectory must be generated for the agent to follow - $z^{<t+1,t+\tau>}$.

Limit situations can cause problems for the intelligent agent, especially if the dynamics of the scene evolve strongly. For example, it is much more difficult for the autonomous vehicle in situations where other vehicles are moving at high speeds or exhibiting dangerous behavior in traffic. For such situations, it is useful for an intelligent agent to have additional details available. In this sense, the intelligent component of the autonomous driving system can make use of sensory information predicted by an artificial intelligence algorithm.

Summarizing the problem addressed in this paper, the intelligent agent navigating autonomously, whether it is a vehicle in traffic or a robot in an unstructured working environment, must be able to navigate between the starting point and the destination point, simultaneously avoiding dynamic obstacles in the scene.

Also, scenarios that are dangerous for an autonomously navigating agent, represented in the form of dynamic obstacles in the surrounding environment, are of critical importance for the successful completion of the tasks of this agent.

Research objectives

The main objective of the research activity is represented by the utilization of artificial intelligence techniques to achieve autonomous navigation of a mobile robot, while also considering boundary situations generated by the unpredictable evolution of the surrounding environment in which the robot navigates. In order to accomplish this algorithm for autonomous navigation, activities such as studying specialized literature, developing concepts, implementing them, as well as testing in simulated and real environments using mobile robots, have been carried out.

Derived from the main objective of the work, specific objectives have also been outlined:

- Studying specialized literature in the field of autonomous navigation.
- Studying works on environmental prediction.
- Studying works on intelligent control.

- Development of the ObserveNet Control architecture (an artificial intelligence algorithm used in the autonomous navigation task).
- Training the ObserveNet artificial intelligence algorithm with specific data for simulated and real scenarios.
- Implementation and testing of control algorithms for mobile robots.
- Implementation of perception algorithms for mobile robots.
- Implementation of sensor data processing algorithms.
- Implementation of a localization module for the mobile robot.
- Implementation of a local trajectory planning module.
- Implementation of the necessary interfaces for all subsystems responsible for robot navigation: actuators, inter-module communication, sensor data acquisition, etc.

Thesis outline

The thesis is structured into seven chapters, as follows:
Chapter 1 presents some ideas about the impact of artificial intelligence on the field of autonomous navigation, as well as a high-level description of the main components of an autonomous navigation system. Additionally, the main objective and secondary objectives of the research are presented, along with the structure and content of the thesis.
Chapter 2 provides an overview of artificial intelligence technologies, as well as neural networks and their application in intelligent control techniques for autonomous vehicles. The main types of neural networks used by researchers are described, the technique used for training neural networks is presented, several cost functions used in training are described, as well as various activation functions used.
Chapter 3 presents general notions about the perception component of an autonomous navigation system, starting from the types of sensors used for perception, as well as explanations about the processing of this sensor data before being sent to various interpretation algorithms. Then, theoretical notions about the trajectory planning of an intelligent agent are presented. Also, the types of trajectory planners are discussed, and the generation of local and global trajectories is discussed. The main control algorithms used for intelligent agents are also presented.
Chapter 4 presents the current state of research on artificial intelligence algorithms for the task of autonomous navigation, describes the main components of navigation systems, and presents perception, planning, and control algorithms developed for this thesis.
The simulated platforms, as well as the differential mobile robot (Agile Scout-X) used for testing the algorithm, along with the experimental results and corresponding conclusions, are presented in Chapter 5.
The second robotic platform used (mobile robot Unitree A1) for testing, along with the experimental results and corresponding conclusions, are presented in Chapter 6.
Chapter 7 presents the final conclusions, as well as the dissemination of results and summarization of future research directions.

Research methodology

The proposed autonomous navigation system in this work is structured in the form of a perception-plan-control architecture, a model that involves separating the components responsible for the navigation task into dedicated modules. These modules include perception of the surrounding environment, localization, trajectory planning, and control. This navigation system has been implemented according to a modularized architecture, so that it can be used with minimal modifications on multiple platforms, both in simulated environments and with real mobile robots.

A mobile robot navigating autonomously must possess knowledge about its surrounding environment at all times, in order to evaluate certain risky situations and make decisions accordingly. The perception component is responsible for the acquisition and interpretation of sensor data from the robot. The most important aspect of perception is the ability to assess the dynamic evolution of obstacles in the working environment.

Sensorial observations processing

The perception module of the mobile robot uses information from a multitude of sensors. Among them, the most used are the RGB cameras, based on which the understanding of the viewed scenes is achieved with the help of Artificial Vision or Artificial Intelligence algorithms. Another category of sensors frequently used in autonomous navigation tasks are those that provide topological information about the environment in which mobile robots are navigating, such as LiDAR or RADAR. The sensors used by the mobile robots (simulated or real) in this work are distance sensors: LiDAR, RGB-D camera, respectively ultrasonic.

LiDAR data comes in the form of point clouds with a fixed acquisition frequency and will be referred to hereafter using the term observation (sensor) interchangeably. Thus, a LiDAR sensory observation will be described as follows:

$$I^{<t>} = \{(X_i, Y_i, Z_i), \forall i \in [0, N]\}$$

Where I is the notation for the observation, the exponent $< t >$ refers to the current time instant t, and (X_i, Y_i, Z_i) are the three-dimensional Cartesian coordinates of point i. N is the size of the point cloud.

The point cloud obtained from the sensor first goes through a coordinate transformation so that it has the points relative to the mobile robot's coordinate system.

$$I^{<t>} \leftarrow T_{VEH}^{L} \cdot \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}, \forall i \in [0, N]$$

Where T_{VEH}^{L} is the homogeneous transformation matrix from the coordinate system of the sensor to that of the vehicle. It has the form:

$$T_{VEH}^{L} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

Where *R* is the rotation component and *t* is the translation.

The next step of processing the point cloud is to change its representation in the OcTree structure. This is a tree structure for representing three-dimensional objects by recursively dividing them into subdivisions with eight children each. Thus, sensory observations will become:

$$I^{<t>} \leftarrow f_O(I^{<t>}, r)$$

Where f_O is the function for converting point clouds into OcTree representation and *r* is the desired resolution.

Another preprocessing step in the data flow is the distance based thresholding, where the sensorial observations have their dimensionality reduced by:

$$p_r \leftarrow p[i] \leftrightarrow d(s, p[i]) < T_d, \forall i \in [1, N]$$

Where p_r is the selected base unit, $p[i]$ is the analyzed base unit, $d(s, p[i])$ is the Euclidean distance between the origin of the vehicle and the unit, T_d is the (empirically chosen) threshold value, and N is the observation size.

The next processing operation is the elimination of the points located in the floor plane, respectively those located at a higher height than that of the mobile robot:

$$p_r \leftarrow p[i] \leftrightarrow V_L < p[i]_Z < V_H, \forall i \in [1, N]$$

Where $p[i]_Z$ is the point height, V_L and V_H are the minimum and maximum threshold values for a unit to be considered.

Points which lie outside the robot's field of view are also eliminated:

$$p_r \leftarrow p[i] \leftrightarrow |\angle(p[i], V_O)| \leq T_V$$

Where V_O is the mobile robot origin and T_V is the threshold value (empirically chosen) to define the maximum field of view of the robot.

After performing the pre-processing operations described previously, the current sensory observation represented in the three-dimensional OcTree structure is projected in a top-down perspective:

$$I^{<t>} \leftarrow f_{2D}^{3D}(I^{<t>})$$

Where $f_{2D}^{3D}(\cdot)$ is the 3D-2D projection function.

$$f_{2D}^{3D}(p_i) = \{(X_{p_i}, Y_{p_i}), \forall i \in N\}$$

Where (X_{p_i}, Y_{p_i}) are the Cartesian coordinates of the OcTree point.

Mobile robot localization

The localization component uses geolocation data recorded by the mobile robot as well as measurements from the inertial sensor to calculate the current state of the robot. The state transition model is defined as follows:

$$s^{<t+1>} = s^{<t>} + v^{<t>} \begin{bmatrix} \cos(\phi^{<t>}) \\ \sin(\phi^{<t>}) \\ 1 \\ \frac{1}{L}\tan(\delta^{<t>}) \end{bmatrix} dt$$

Where $s^{<t>}$ is the current state of the robot and $\delta^{<t>}$ is the steering angle of the robot. The current state is defined as:

$$s^{<t>} = [x, y, v, \phi]^{<t>}$$

Where x, y Are the Cartesian coordinates of the robot, *v* is it's speed and ϕ it's orientation. *L* is the vehicle length and *dt* is the sampling time.

The process of estimating the state of the robot is performed using the Kalman Filter, as described next (for the case of a mobile robot with two motors). First the number of pulses given by the wheel encoder during the sampling period dt is calculated:

$$\delta_S^{<t+1>} = E_S^{<t+1>} - E_S^{<t>}$$

respectively

$$\delta_D^{<t+1>} = E_D^{<t+1>} - E_D^{<t>}$$

Where δ is the number of pulses recorded during the sampling period and E is the value returned by the encoder. Exponents $< t+1 >$ and $< t >$ refer to the time points for which the values are calculated, and the indices D and S refer to the right and left wheels, respectively. Afterwards, the number of rotations for the left and right wheel is calculated:

$$R_S^{<t+1>} = \frac{\delta_S^{<t+1>}}{p_{RS}}$$

respectively

$$R_D^{<t+1>} = \frac{\delta_D^{<t+1>}}{p_{RD}}$$

Where R_S and R_D are the number of rotations for the left and right wheel, respectively and p_{RS} and p_{RD} are the number of pulses per revolution for the left and right wheel, respectively. Given the number of revolutions, the distances traveled by the two wheels are calculated:

$$d_S^{<t+1>} = R_S^{<t+1>} C$$

respectively

$$d_D^{<t+1>} = R_D^{<t+1>} C$$

Where d_S and d_D are the two distances traveled and C is the circumference of the wheels. The result is the speed of the robot:

$$v^{<t+1>} = \frac{1}{2}(d_S^{<t+1>} + d_D^{<t+1>})dt$$

The robot's orientation is given by the compass:

$$\phi^{<t+1>} = \phi_{IMU}^{<t+1>}$$

Where ϕ_{IMU} is the orientation provided by the inertial measurement unit (IMU).
Given this estimate of the robot's state, the prediction step of the Kalman Filter is performed:

$$\hat{s}^{<t+1>} = F^{<t+1>} s^{<t>}$$

Where $\hat{s}^{<t+1>}$ is the predicted state and $F^{<t+1>}$ is the state transition matrix.
As soon as GPS measurements are available, the update step of the Kalman Filter is performed:

$$s^{<t+1>} = \hat{s}^{<t+1>} + K^{<t+1>} \hat{y}^{<t+1>}$$

Where $\hat{y}^{<t+1>}$ depends on the measurement vector provided by the GPS, *K* is the Kalman gain and $s^{<t+1>}$ is the post measurement estimated state.
For the situation where GPS measurements are not available (navigation in closed spaces), the Kalman Filter is used together with measurements from the visual odometry system based on the detection of ArUco markers.
Thus, the problem of determining the position, respectively the orientation of the intelligent agent is transformed in an Computer Vision problem. Visual markers are computer generated templates, having the property of being easy to detect in images. Such a marker presents enough key points (corners) so that its position and rotation in the environment can be estimated. In addition to being easy to detect in images due to their construction, these markers encode a unique identifier, depending on the template generated for each one, thus facilitating their identification in the viewed scene.
The ArUco-based localization method works as follows:
1. Certain reference points are chosen in the map where the mobile robot navigates.

2. Visual markers oriented to the general direction from which the robot will approach are placed at those previously chosen points.
3. The agent's localization module stores in a dictionary the correspondences between the identifier coded by each marker, respectively the coordinates in the global reference system of the marker.
4. During navigation, the intelligent agent detects using the camera one marker at a time and then performs visual odometry based on these markers.
Next, the visual odometry process based on ArUco is described. Starting from the matrix of intrinsic parameters of the camera attached to the vehicle, at the same time knowing the distortion coefficients, respectively the physical size of the ArUco markers, the homogeneous coordinate transformation is built for each marker:

$$T_M^W = \begin{bmatrix} R_M & T_M \\ O_{1\times 3} & 1 \end{bmatrix}$$

Where T_M^W s the homogeneous transformation that describes the position of the marker in the global coordinate system, R_M is the rotation component, T_M is the translation and $O_{1\times 3}$ is a null vector of size (1×3).
Afterwards, the marker is detected and identified using the *detectMarkers* function from the OpenCV library, if such a marker is present in the scene. Using this OpenCV function, the homogeneous transformation matrix which describes the marker coordinates in the camera reference system is obtained: T_M^C. Afterwards, the transformation matrix which describes the camera coordinates in the reference system of the marker is calculated:

$$T_M^C = {T_M^C}^{-1}$$

Since the camera coordinate system is not in the ISO format, two rotations must be made: one around the X axis, of $90°$and one around the Z axis, of $90°$:

$$T_{YZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(90°) & -\sin(90°) & 0 \\ 0 & \sin(90°) & \cos(90°) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

respectively

$$T_{XY} = \begin{bmatrix} \cos(90°) & -\sin(90°) & 0 & 0 \\ \sin(90°) & \cos(90°) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Given these, the homogeneous matrix is defined for the change of coordinates from the camera coordinate system to the global one:

$$T_{CM}^W = T_{XY} \cdot T_{YZ} \cdot T_C^M$$

Then the coordinates of the camera in the global coordinate system are obtained:

$$T_W^C = T_M^W \cdot T_{CM}^W$$

By obtaining this homogeneous matrix, one can extract the Cartesian coordinates of the camera in the global coordinate system, as well as the rotations on the three axes. These can then be integrated into the measurement vector of the Kalman Filter:

$$z_A^{<t+1>} = [x_A, y_A, 0, \phi_A, a_x, a_y, r_z]$$

The prediction step of the Kalman Filter is performed with the state estimates given by the IMU module, together with the movement model itself, while the update step is performed using the positioning components, depending on their availability. Measurements are either

calculated using visual odometry, based on the camera mounted on the mobile robot plus the ArUco visual marker detection algorithm, or by the GPS module coupled to the compass.

The alternation of the two localization systems is done both at the initialization step of the system and during the autonomous navigation task. This is due to the fact that sometimes the mobile robot enters certain areas where the signal of the GPS system is not stable enough, in which case there is the visual odometry system as a redundant system.

Sensory observations model

The main component of the algorithm proposed in this thesis is called ObserveNet Control and is composed of an artificial intelligence algorithm designed to predict a pattern of sensory observations of an autonomously navigating mobile robot. It uses data from the system's perception and localization components to make predictions of future sensory observations. Sensory observations recorded by the robot are stored in the augmented memory of the artificial intelligence module, together with the generated local trajectories, respectively the global reference trajectory:

$$f_{MA}(z_{ref}, z, I) = \{I^{<t-i>}, \forall i \in [0, \tau_0]; z^{<t-i>} \forall i \in [0, \tau_i]; z_{ref}\}$$

Where z_{ref} is the global reference trajectory and $z^{<t-i>}$ are generated trajectories for historic time steps.

Historical sensor observations $I^{<t-\tau_i,t>}$ are successively passed through two convolutional layers. Thus high-level features are extracted from sequences of occupancy maps returned by the module's augmented memory. From a mathematical perspective, convolutional layers perform the following operation:

$$G(m,n) = f * h = \sum_j \sum_k h[j,k] f[m-j, n-k]$$

Where G is the convolution result, f is the data structure on which the convolution is applied and h is the convolutional kernel.

$$I_{CONV}^{<t-\tau_i,t>} = I^{<t-\tau_i,t>} * K_i, \forall i \in [0, M]$$

Where M is the number of kernels for the respective convolutional layer.

After passing through the convolutional layers, the sensorial observations are passed through two completely connected layers, plus one flatten layer.

$$I_f^{<t-\tau_i,t>} = f_f(f_{FC}(I^{<t-\tau_i,t>}))$$

The flattened resulting observations are passed through the LSTM recurrent layers:

$$\hat{I}^{<t+1,t+\tau>} = f_{LSTM}(I_f^{<t-\tau_i,t>})$$

In addition to processing historical observations, ObserveNet also handles local trajectories generated at time steps $[t-\tau_i, t]$ associated with observations at those same time steps, as well as the global trajectory z_{ref}.

Apart from trainable differentiable layers (fully connected, convolutional, respectively LSTM) included in the ObserveNet algorithm, it also includes certain non-differentiable layers, designed to associate each predicted observation with the most probable future state of the intelligent agent. The Planning and Simulation layer uses recursively the time scheduler to simulate future states of the robot in the same time replanning the target trajectory z. Trajectories generated using predicted observations simulated within the planning and simulation layer are filtered using the Kalman Filter.

The neural network at the heart of the ObserveNet algorithm was trained for 10000 epochs in a self-supervised manner using the Adam optimizer and a learning rate of 0.0003. The cost function minimized in training was Mean Squared Error.

Local trajectory planning

The temporal trajectory planner is based on DWA, modified to iteratively calculate trajectories for the controlled vehicle based on current and historical sensor observations. DWA is a collision avoidance strategy for mobile robots that uses their dynamic motions as well as various constraints on velocities and accelerations respectively to calculate the optimal route in the 2D plane.

DWA was implemented based on the scheduler found in ROS. It receives as inputs information about the obstacles present in the scene, in the form of the distance between the controlled vehicle and the obstacles. In this work, DWA was modified to be able to accept both current vehicle observations and predictions.

The desired state of the vehicle is iteratively predicted at each sampling period in the future, based on the temporal dynamics of the vehicle, respectively observations. The input data is both historical data $< t - \tau_i >$, as well as predicted data $< t, t + \tau_0 >$:

$$z^{<t+i>} = f(\hat{I}^{<t-\tau_i, t+i>}, z^{<t+i-1>})$$

Where *i* is the time step for which the prediction has been made and $f(\cdot)$ is the temporal planning function. The function is used to interpolate trajectories specific for time steps $[t - \tau_i, t + \tau_0]$ and can be decomposed in:

$$f(\cdot) = \sum_{t=-\tau_i}^{0} DWA(I^{<t>}, z^{<t>}) + \sum_{t=1}^{\tau_0} DWA(\hat{I}^{<t>}, \hat{z}^{<t>})$$

Where the first term represents the trajectories interpolated by DWA for historical observations and the second represents the interpolation of trajectories based on predicted observations. The states are integrated within the generated local trajectory, and this is used as input to the control module of the algorithm.

Mobile robot control

The trajectory generated by the temporal planner is the one used by the MPC controller to execute the movement of the mobile robot. MPC computes an iterative solution for optimal control problems over a finite prediction horizon. To be able to run MPC, first the necessary quantities are defined, starting with the system inputs (command inputs):

$$u = [a^{<t>} \quad \delta^{<t>}]$$

Where $a^{<t>}$ is the robot's acceleration and $\delta^{<t>}$ is the steering angle.

Afterwards, the number of steps for which the optimization is performed is defined (e.g. *T=10*). The variables based on which the system is being constructed are the current state of the mobile robot:

$$s^{<t>} = [x, y, v, \phi]^{<t>}$$

Next, the quantities are chosen on the basis of which the cost function is built, which will be optimized using the MPC algorithm. These are the reference speed, obtained from the global reference trajectory (v_{ref}), the orientation error (e_ϕ), respectively the deviation from the reference trajectory (e_{ct}). A set of weights is then defined for these error quantities:

$$w = [1.0 \quad 3.0 \quad 10.0]$$

Where these weights have been empirically chosen as to enable MPC to have a satisfactory behaviour.

Afterwards the state matrix X_{T,N_S} is being intialized, as well as the control matrix U_{T,N_U}, where N_S is the state vector size and N_U is the control vector size.

After the variable initialization step, the system optimization step follows. The closest T states on the reference trajectory to the mobile robot, in terms of Euclidean distance, are chosen:

$$P_R = \{z_i, z_i \in z_{ref}^{<t-\infty, t+\infty>}, \forall i \in [0, T]\}$$

where z_i is the current state on the global reference trajectory and z_{ref} is the global reference trajectory.
After the Cartesian coordinates belonging to the selected states within the reference trajectory were selected, they were translated to the origin of the vehicle and respectively rotated by its angle:

$$P_R = \{(x_i - x_{veh}, y_i - y_{veh}), \forall i \in [0, T]\}$$

respectively:

$$P_R = R_{-\phi} P_R$$

Where x_i, y_i are the Cartesian coordinates of the state i chosen from the global reference trajectory, x_{veh}, y_{veh} are the mobile robot's Cartesian coordinates and $R_{-\phi}$ is the rotation matrix necessary for rotating the points with the inverse of the vehicle orientation.
The rotated and translated points are then used for applying polynomial regression for a fourth degree polynomial, using the Least Square method. As such, the polynomial is defined as:

$$\hat{y} = X\hat{\beta}$$

where $X = [1 \quad x \quad x^2 \quad x^3 \quad x^4]$ and $\hat{\beta}$ is the polynomial coefficient matrix. It is then obtained:

$$\hat{\beta} = (X^T X)^{-1} X^T \hat{y}$$

Afterwards, the cross-track error and orientation error e_{ct}, e_ϕ are being defined, as well as the MPC cost function:

$$J_{MPC} = K_{ct} e_{ct} + K_\phi e_\phi + K_v v_{ref}$$

Where K_{ct}, K_ϕ, K_v are weights associated with the previously defined error quantities.
The following constraints have been defined:

$$a_{min} < a < a_{max}$$

respectively:

$$\delta_{min} < \delta < \delta_{max}$$

Also having these constraints, the current state of the vehicle is assigned as the initial state of the MPC algorithm and the optimization is performed, using the IPOpt algorithm.

ObserveNet Control particularities for RGB-D data

Although the general structure of the ObserveNet Control algorithm is similar for both LiDAR and RGB-D camera sensory data, there are some differences.
Another implementation of the ObserveNet Control algorithm that performs the prediction for sensory observations of an intelligent agent is based on depth images acquired using an RGB-D sensor, unlike the previously presented implementation, which
uses data from the LiDAR sensor. In the following, the particularities of the prediction algorithm of the sensory data coming from the RGBD sensor will be described, as well as the particularities of the complete autonomous navigation system built around this prediction algorithm.
In principle, the ObserveNet Control architecture operating using RGB-D data is similar to the LiDAR variant. So, the problem that this architecture deals with is the following: giving a series of historical observations $\Theta^{\{<t-\tau,t-1>\}}$, current observation $\theta^{<t>}$, provided by the RGB-D sensor, the current state of the mobile robot $s^{<t>} = [x, y, v, \phi]^{<t>}$, as well as the global reference trajectory $z_{ref}^{<t-\infty,t+\infty>}$, the goal is to build a recurrent neural network that can encode within its hidden layers the dynamics of the environment and based on historical observations presented to the network, it can predict future observations for a finite time window $[t+1, t+n]$. This AI model aims to build an agent's belief of the environment, which is then used by the local trajectory planning algorithm to generate an optimal collision-free trajectory for the agent $z_L^{<t+1,t+n>}$.

Encoding the dynamics present in the scene involves processing the data coming from the RGB-D sensor to create a representation of the current scene, also to predict the evolution of these dynamics over a finite time horizon. LSTM neural networks are suitable for encoding scene dynamics due to their ability to learn temporal dependencies for the input data and use these dependencies to predict future states of the system.
The embedding of the environment is based on an LSTM recurrent neural network, which contains several cells for processing time series data in the form of historical observations of length τ. The output of this AI model is a sequence of predicted future observations of length n.
The depth information from the RGB-D system is not directly fed into the network but is first pre-processed. Thus, the data acquired using the RGB-D sensor must be used to recreate their three-dimensional structure from the global coordinate system.

$$P_{3D} = \{[X_i, Y_i, Z_i], \forall i \in D_{img}\}$$

Where X_i, Y_i, Z_i are the coordinates of the three-dimensional point P and D_{img} is the depth image. For every point from the depth image, the points' coordinates in 3D are calculated:

$$\begin{cases} X = (j - c_x)\dfrac{z}{f_x} \\ Y = (j - c_y)\dfrac{z}{f_y} \\ Z = z \end{cases}$$

Where j, i are the coordinates of a pixel in the depth image, f_x, f_y are the coordinates of the focal length, and z is the value of the pixel in the depth image at coordinates (i, j).
Then, like the case of LiDAR information processing, the thresholding operations are performed:

- Points that are too far away or to close to the vehicle origin are eliminated
- Points located at a greater height or on the ground plane are eliminated
- The point cloud structure is converted into a Voxel Grid representation.
- Top-down projection of the Voxel Grid structure is created.
- Two-dimensional voxel grid data is accumulated in historical data sequences, which will be further processed by the recurrent neural network

These data sequences are then restructured into the required form to be processed by the LSTM network. It outputs data sequences of length n, representing predictions of sensory observations n steps into the future.
The predictions of the LSTM network are then passed through successive fully connected layers and sent to the clustering module to be clustered using the DBSCAN method. The clustering module has the role of transforming the predictions of the network in the form of voxels at each time step $t + i$ into lists of obstacle centroids plus their sizes. The Log Odds method is also used to filter these centroids.
The recurrent neural network was trained in a self-supervised manner for 1000 epochs, using the Adam optimizer with a learning rate of η=0.001, using historical sequences of size τ=50, having a prediction horizon size of n=30, equivalent to 5 , respectively 3 seconds. Also, an asymmetric non-standard cost function was used due to the input/output data structure of the network.
The Zero Moment Point (ZMP) algorithm was used to control the mobile robot, together with the DWA local trajectory planner.
Candidate DWA trajectories are then evaluated against the following performance criteria:

- Goal point cost e_G, calculated as the Euclidean distance between the last state of the candidate trajectory and the coordinates of the destination point in the global reference trajectory.

- Orientation cost e_{ϕ}, which represents the difference between the orientation of the final state on the candidate trajectory and that of the destination state.
- Velocity cost e_{v}, which represents the difference between the reference speed recorded on the global trajectory, respectively the speed of the robot.
- Smooth trajectory cost e_{S}, which represents the difference between two successive sets of throttle and steering angle commands.
- Cost of proximity to the global reference trajectory, which denotes the proximity of the generated local trajectory to the global reference.

Experimental results

From the perspective of the software platform, the general framework that facilitated the development of the proposed algorithm is the ROVIS platform, both as an operating system for the hardware platforms used, and as a development environment for artificial intelligence algorithms.
The operating system for ROVIS robots is built modularly, having dedicated components for sensors used by autonomous vehicles, for example LiDAR, RGB, RGB-D, ultrasonic cameras, etc. Dedicated sensors for autonomous vehicle location such as IMU, GPS or compass are also supported.
It runs natively in C++, but also allows data acquisition through third-party programs, such as ROS or CARLA, having implemented interfaces for them.
It also provides support for various sensory data processing algorithms for VA or DL operations, such as image filtering, segmentation, data preparation for artificial intelligence algorithms, etc. ROVIS works both in real time, connected to a simulator like CARLA, or to a mobile robot, and in simulated computer mode, to test the developed algorithms off-line. So,
data can be recorded during the operation of a robot, and then this data can be used in simulated mode, without the need to configure the robot. Established databases for testing artificial intelligence algorithms, such as nuScenes or KITTI, are also supported.
The dedicated components for autonomous navigation were implemented using the ROVIS architecture, respecting the general structure, implementing the dedicated interfaces for data acquisition, agent localization, trajectory planning, intelligent control and actuator control of the simulated or real mobile robot.
Thus, except for implementation variations specific to each platform with which the experiments were carried out, the implemented algorithms were the same.
Experiments were conducted within three different test platforms: the GridSim simulated environment, the CARLA simulated environment, and the Agile Scout-X robotics platform. Across all four platforms, the intelligent agent controlled using ObserveNet Control had to navigate autonomously, with an emphasis on difficult situations associated with this task. Thus, dedicated test scenarios were defined for each platform used for the experiments, with the role of highlighting the performance of the intelligent agent in the difficult situations encountered during autonomous navigation.

GridSim simulated environment

GridSim is a simplistic autonomous navigation simulator where a mobile robot architecture is used to generate occupancy maps using simulated sensors. It was implemented in Python and allows the integration of artificial intelligence algorithms for testing purposes. GridSim allows the simulation of distance sensors with a configurable FOV, which can be attached to both the front and rear of the vehicle.

As a simplistic system for simulating autonomous driving situations, GridSim was used to generate synthetic simulated sensor data specific to a highway navigation scenario. The vehicle was driven manually while training data for the algorithm was acquired. The scenario consists of two vehicles traveling parallel in a straight line. The main vehicle moves at a constant speed, while the second vehicle moves at a variable speed.
The lead vehicle simulates sensory data while the second vehicle enters and exits the field of view of the lead vehicle, overtaking and being overtaken, respectively. These sensory data are stored and processed in N-dimensional historical sequences and P-dimensional prediction sequences, respectively.
To gather training data, the intelligent agent was driven manually, while a second vehicle was controlled automatically, varying its speed and distance from the agent.
The agent records data from simulated sensors as another vehicle passes through their field of view, while overtaking or being overtaken. The recorded data is then processed into sequences of length N (the size of the historical observations), and the speed of the intelligent agent is stored into sequences of length P, where P is the size of the prediction horizon.
ObserveNet Control was trained in a self-supervised manner for approximately 1000 epochs, having checkpoints containing the intermediate weights of the model saved, respectively having the reduction of the learning rate, as well as early stopping. To vary the training data, the second vehicle was randomly positioned at the beginning of each data recording session.
To train the ObserveNet Control, 33000 observation sequences were collected, each of length $N = 150$. The model was used to predict for a horizon $P = 10$ consecutive frames from the future, using 30 simulated rays, and was subsequently evaluated using 6700 test frames, yielding statistical error magnitudes, evaluated per-pixel.
The performance of the ObserveNet algorithm was evaluated in terms of percentage statistical error. Thus, the performance of the system for this simulator could be observed in the work.

CARLA simulated environment

The algorithm presented in the thesis was tested in a simulated environment using the platform intended for the evaluation of autonomous navigation algorithms called CARLA. This simulator was designed to allow researchers to implement, train and validate autonomous navigation systems. Additionally, the simulator provides textures for urban scenes, buildings and vehicles for developers. CARLA also has implementations for various sensors used for autonomous navigation, such as RGBD, LiDAR, ultrasonic, RGB cameras, and RADAR sensors. At the same time, CARLA has support for the location of the intelligent agent, generating simulated GPS data.
In terms of connectivity with applications developed by researchers, CARLA exposes an API that allows control of both vehicles and simulation conditions, such as weather conditions, configurable behavior of pedestrians and automated vehicles, etc.
The simulator was built using Unreal Engine and is based on a client-server architecture, where multiple nodes can simultaneously connect to a server running the simulation.
An intelligent agent can be configured to be integrated into the simulation in several ways. The simulator is flexible in this regard, supporting implementations for agents in C++ as well as Python. It also provides the user with a bridge for ROS.
Being a dedicated simulator for autonomous navigation algorithms, CARLA allowed the entire processing chain of the proposed algorithm to run. Thus, interfaces were made for the simulated sensors implemented in CARLA, for the localization module in the simulator, as well as for the vehicle controls. Communication between ROVIS and CARLA was done in a client-server manner, where the client was the intelligent agent.

The autonomous agent was implemented following a perception-plan-control type structure, in which each task associated with autonomous navigation is encapsulated in its own module. Thus, the following modules were used to carry out this experiment: the perception module, the one that includes the artificial intelligence algorithm, the state estimation module, the planning module, respectively the control module.
The perception module, which in this case uses sensory data from LiDAR, has the role of acquiring and processing data sequences organized in the form of point clouds. It connects directly to the API exposed by CARLA to gain access to the point clouds returned by the simulator.
The location module, based on GPS data processing, connected to the corresponding API in CARLA is used to obtain data about the current state of the agent, which it forwards to the trajectory planning component.
The artificial intelligence module processes the sequences of point clouds provided by the perception module, arranging them into historical data sequences, then applying them to the recurrent neural network, which has the role of predicting sensory observations in the future. The predicted observations along with the current sensory observation as well as the location information are used by the trajectory planner, which has the role of generating the optimal trajectory for the intelligent agent.
The controller receives as input the trajectory generated by the planner as well as the current state of the agent and generates the appropriate command signals so that the agent to proceed to its destination. This component is connected via the CARLA API directly to the simulator in order to be able to control the simulated vehicle.
In the experiments in the CARLA simulator, emphasis was placed on modeling the environment of the intelligent agent. Thus, the behavior of the prediction algorithm of sensory observations in the form of LiDAR data, as well as of the entire autonomous navigation system in borderline traffic situations, was verified.
Thus, several scenarios were defined in which the behavior of the intelligent agent was validated: the overtaking scenario, the one with traffic coming from the opposite direction, the one approaching from the side, the one in which the vehicle is overtaken, respectively the one with multiple agents present in the scene .
The overtaking scenario is a challenge, especially at high speed. The other vehicle accelerates less than the intelligent agent, thus creating the need for the overtaking maneuver.
The oncoming traffic scenario assumes that the agent goes forward, and another vehicle approaches from the opposite direction, driving close to the lane demarcation line. This creates a dangerous situation for the agent. Using sensory predictions should generate a safer trajectory for the vehicle.
The side traffic scenario assumes that the agent enters an intersection, and another vehicle enters the intersection from the right or left, respectively, causing an accident.
The use of predictions should make it possible to avoid collisions with other road users.
For the situation where the agent is in turn overtaken, it must use the sensory predictions to plan the trajectory in such a way that it moves away from the overtaking vehicle.
The multi-vehicle scenario is the most complex test environment in the simulator. The vehicle must travel a longer route, during which it interacts with multiple vehicles, overtaking where appropriate, while avoiding collisions. This scenario best replicates complex real-world situations where the autonomous vehicle has limited room to maneuver.
The autonomous navigation algorithm had to incorporate in addition to the collision avoidance logic based on the current sensory observations, the sensory observations predicted by the artificial intelligence algorithm.

The performance of the autonomous navigation algorithm was compared with the classical DWA algorithm. The evaluation consisted of sequential loading of the five aggressive driving scenarios described previously, within the CARLA simulator. The global reference trajectory was recorded by manually controlling the intelligent agent in the simulated environment.
At the same time, the successive states of the agent were recorded so that they could be used later.
As performance indicators for the algorithm, the number of collisions between the intelligent agent and other traffic participants N_C, the average deviation from the reference path e_{ct}, the vehicle orientation error e_ϕ, as well as the rate of change of the acceleration command quantities, respectively direction angle: e_a and e_δ. The vehicle's ability to complete the test route was also graded C_{100}.
In the first phase, the ObserveNet Control algorithm was compared with the classic DWA, achieving superior performance in terms of the previously described performance criteria. Similarly, the second comparison with two Imitation Learning methods (Learning by Cheating, respectively World on Rails) produced superior results for ObserveNet Control.

CARLA simulated environment – Autonomous Driving Learderboard

The second evaluation method within the CARLA simulator was carried out using the CARLA autonomous leaderboard platform. This is an open-source platform dedicated to testing and comparison of autonomous navigation algorithms, making available to developers more urban scenes, respectively more traffic conditions and weather conditions. Algorithm test tracks simulate several challenging scenarios, such as loss of vehicle control, obstacle avoidance, lane changing, traffic lighted intersections, pedestrians crossing irregularly, etc. These scenarios are used to evaluate performance by building a total driving score, route completion percentage and penalties for offenses (e.g. collisions).
Since the objective of the proposed algorithm is to reduce the number of collisions, i.e. to optimize deviations from the reference trajectory, but also to optimize the rates of change for vehicle commands, due to the concept used for vehicle control, based only on sensory information from LiDAR, there were no considered the scores for offenses involving the use of VA and object detection algorithms, such as road marking violations or violations of the meaning of traffic lights.
The proposed algorithm was evaluated against three well-known algorithms in the field, namely DWA, Learning by Cheating and World on Rails. The first is a classical algorithm used in particular for autonomous navigation in robotics, and the next two are state-of-the-art implementations of the imitation learning paradigm.
The experimental results show improvements in the performance of the autonomous agent in terms of the number of collisions, respectively the quality of the trajectory generation, as well as the control mode of the agent, compared to current algorithms used in the field.

AgileX Scout mobile robot

The AgileX Scout robotic platform is intended for navigation in difficult environments, having all-wheel drive and having a maximum mass that can be carried limited to 50 kilograms. This robot can be controlled by remote control, or it can be adapted to run autonomous navigation algorithms. Its maximum travel speed is 1.5m/s and it is weatherproof according to the IP22 standard. It is equipped with a 30Ah battery, which operates at a voltage of 24V, having an autonomy of 15km (without charge) and comes equipped with four 400W DC motors.
From a communication point of view, it gives developers access to the CAN network, while also providing them with low-level control functionality organized in the form of dedicated APIs written in C++.

To implement autonomous navigation algorithms, the following were attached to the robot:

- A 360° Pandar 40 Hesai LiDAR sensor.
- An IMU VESC unit.
- Four synchronized e-CAM130A cameras to achieve the 360^∘ visual perspective.
- A GPS receiver type unit for location.
- An nVIDIA AGX Xavier compute unit running the Linux operating system.

Due to its versatility and way of construction, the AgileX Scout robot allowed the assembly of the aforementioned computing system as well as the necessary sensors for the artificial intelligence algorithms. Thus, various algorithms could be run, from classic control ones to sophisticated artificial intelligence algorithms.
The ROVIS operating system, over which the autonomous navigation algorithm was implemented, was implemented on the Linux platform running on nVIDIA Xavier. The communication between the agent and the robotic platform is carried out by the operating system of the PC, through the API provided by the manufacturer of the robotic platform. The AgileX Scout robotic platform, together with the sensors, the nVIDIA computing unit and the ROVIS operating system running on it, is called the RovisLab AMTU.
Unlike the experiments carried out in the simulated environment described previously, in the case of experimentation using the RovisLab AMTU platform, the test environment is completely different, both due to the use of a hardware versus software platform used in the simulator, as well as due to the navigation problem itself. Compared to the previously simulated scenarios, where the environment was structured, in the form of a city navigation scenario, in this case the mobile robot has to navigate in an unstructured environment under laboratory conditions. Starting from this premise, the autonomous navigation problem changes due to the environment, especially due to the different types of obstacles that the intelligent agent can "expect".
Thus, although fundamentally speaking the process of autonomous navigation should be identical, the particularities of this environment in which the navigation was carried out for this experiment create difficulties for algorithms based on artificial intelligence.
The main objective for the proposed algorithm is that the intelligent agent, either simulated with special software or implemented on a real hardware platform, can navigate between the starting point, respectively the destination point, avoiding collisions with static obstacles, respectively dynamic obstacles in the environment.
For the purpose of this experiment, two reference trajectories recorded using the manually controlled AMTU system in closed space and open space, respectively, were defined, which the intelligent agent must follow, without making collisions with obstacles present in the environment. Given the fact that static obstacle avoidance is also successfully achieved by the classic DWA algorithm, the focus is on obstacles that have pronounced dynamics, namely human operators in this case.
Training data of approximately 38000 pairs of historical and future LiDAR sequences were acquired both inside and outside the Transylvania University research institute.
Similar to the simulated case based on CARLA, five scenarios were used to validate the behavior of the proposed algorithm in limit situations. The algorithm was compared with DWA, and the AMTU platform had to navigate along the recorded trajectories in the closed space and outside, respectively.
The qualitative results of the performance evaluation of the intelligent agent implemented on AMTU are superior to those based on the classical DWA method.

Unitree A1 mobile robot

The final series of experiments was performed using the Unitree A1 robotic platform. It integrates an RGB and an RGBD camera as the main sensors for perception. This robot allows a maximum transported mass of 5kg, has a maximum speed of the articulation motors of 21 rad/s and allows individual control of each joint motor, which can be ordered in torque, speed, and position respectively. The maximum movement speed of the robot is 3.3m/s. It has a tough and lightweight construction and can be powered at 5V, 12V or 19V. As external interfaces, it features four USB ports, two HDMI ports, as well as two Ethernet ports.
Unlike the experiments carried out on the basis of the AgileX Scout platform, in the case of the Unitree A1 robot, the depth sensor was used to model sensory observations.
The mobile robot navigated in the unstructured environment, specifically on forest roads, while collision situations with humans were simulated. Meanwhile, training data was acquired in the form of observations from the depth sensor. At the same time, the reference trajectory z_{ref} that he must follow later was also recorded.
For the evaluation of the algorithm using the Unitree A1 robot, data sequences recorded during the navigation of the robot on forest roads were used. The conduct of the experiments can be summarized as follows:

- Training and testing data were acquired during manual control of the robot. During this time, situations of potential collisions with people around the robot were simulated.
- The model of the robot's observations was trained using the previously acquired data, properly preprocessed.
- The local trajectory planner was evaluated based on the acquired test data.
- A safety distance $s = 0.5m$ was defined for the robot. Any violation of this distance, meaning that the robot approached an obstacle at a distance less than s, was scored as a collision. The number of collisions was quantified, thus resulting in the first qualitative measure of the algorithm: N_C.

The second evaluated metric was the cross-track error e_{ct}, defined as the difference between the global reference trajectory z_{ref} and the generated local trajectory.

$$e_{ct} = \frac{1}{n}\sum_{i=0}^{n} |f(x) - y_i|$$

Where $f(x)$ is the polynomial approximation of the trajectory, evaluated in x.
The third quality measure was the orientation error:

$$e_{\phi} = \frac{1}{n}\sum_{i=0}^{n} |\phi^{<i>} - \phi_d^{<i>}|$$

Where $\phi^{<i>}$ is the robot's orientation at time step i and $\phi_d^{<i>}$ is the desired orientation, from the global trajectory.
To simulate collision situations for the robot, experiments were conducted with humans interacting with it during its navigation. The interactions consisted of people approaching the robot from different angles and at variable speed to simulate dynamic obstacles in the scene. Four test scenarios were carried out: frontal collision, left side collision, right side collision, respectively multiple obstacles.
For the first scenario, a single person approached the robot in front of it, while for the left and right side collision, respectively, the person approached from the respective directions. For the multiple obstacle scenario, the scenes contained two people each.
Using the four scenarios described in the previous section, two comparative tests were performed for the proposed algorithm. In the first phase, it was compared with the classical

implementation of the DWA algorithm, specially adapted for the Unitree A1 robot. Comparative results were superior for ObserveNet Control.
In addition to the comparison of the trajectory planning algorithms, the accuracy comparison for the collision detection process between the observation prediction algorithm and a state-of-the-art object detector, YoloV7, was also performed. This comparison was made for accuracy A, false positive rate FP_R and estimated time to collision ttc. The results of this comparison show better performance for ObserveNet Control.

Control algorith ablation study

An ablation study of the control algorithm used on the AgileX Scout mobile robot was performed. NMPC and DWA, respectively, were compared during robot navigation on five test paths. NMPC achieves better control for the mobile robot, as a result it follows the global trajectory better.
The two control algorithms were compared in terms of the average deviation from the reference path e_{ct}, as well as the orientation error e_{ϕ}. In terms of average deviation from the reference path, NMPC showed better values in all five test paths, due to the predictive nature of the respective control algorithm and the better modeling of the system.
Also, a study was carried out on the impact of the size of the prediction horizon T on the performance of the NMPC algorithm, from the point of view of the two sizes previously mentioned: e_{ct}, respectively e_{ϕ}. The conclusion of the study is that increasing the prediction horizon beyond 10 seconds does not bring significant improvements in control system performance.

RGB-D data prediction ablation studies

In order to evaluate the performance of the algorithm components, three ablation studies were performed. In the first study, three cost functions for the neural network were compared: Mean Squared Error (MSE), Mean Absolute Error (MAE), respectively an asymmetric cost function.
It follows that the asymmetric cost function converges the fastest among the three, respectively to the lowest value.

$$L(v, \hat{v}) = \frac{1}{n}\sum(\hat{v_i} - v_i)^2[(1 - 1_+)\alpha + 1_+(1 - \alpha)]$$

The asymmetric cost function was used to train the recurrent neural network due to the possibility of changing it so that, by using a scaling factor, the total cost for training can penalize

situations where the predicted values for the observations take the logical value 1, although they should be 0. Basically, we want the cost function to penalize the network more heavily when it mistakenly marks a cell in the occupancy map as busy when it is actually free.
After experimentation, a value of 0.1 was chosen for the scaling factor of the asymmetric cost function, which achieved the best performance.
The second ablation performed was that of the size of the prediction horizon. System performance was monitored by varying the size of the horizon in seconds, while also measuring the time to collision ttc, respectively the false positive rate FP_R. Predictably, increasing the prediction horizon produces better values for ttc. But simultaneously with this improvement, the FP_R false positive rate also increases. This increase in the false positive rate is expected, however, because expanding the prediction horizon causes more noise in the predicted observations, thus the algorithm signals more collisions.
For the third ablation study, the performance of the proposed algorithm was compared against the size of the historical observations. Thus, the following values (in seconds) were used for the

size of the observational history: $[1,3,5,7,10]$ (s). It can be seen that increasing the history size does not bring significant performance improvements.

Conclusions and future work

The research carried out during the doctoral studies aimed to implement an autonomous navigation system based on artificial intelligence techniques, for an autonomous agent. Such a system must be able to perform several tasks in the field of autonomous navigation, such as perceiving the environment, extracting additional information based on the perception, such as information about the position and dynamics of obstacles in the scene, planning the trajectory for the agent, so that to move towards its final destination, without colliding with static or dynamic obstacles in the environment, but also to control the agent as it follows the local trajectory generated by the planner.

The fundamental theoretical notions used in the conducted research were presented in the first part of the paper. Thus, a synthesis of the concepts of Artificial Vision was achieved, especially those used for operating with information from the sensors equipping mobile robots. The filtering processes of digital images, the procedures used for three-dimensional estimation, as well as certain high-level information extraction methods using Artificial Vision were presented. Then, notions of artificial intelligence applicable to the field of autonomous navigation were presented. Some basic concepts of artificial intelligence are reviewed, putting focus on artificial neural networks. The main types of artificial neural networks, their applicability, as well as the training mode were presented. Emphasis was placed on the types of networks used in autonomous navigation in general, respectively those used in the present work.

Later, the main algorithms used in specialized works for planning the trajectory of intelligent agents were described, as well as the main types of controllers used to control these agents. After that, the current state of research in the field of autonomous navigation was detailed, especially those that use artificial intelligence to achieve navigation functionality or certain related tasks, such as intelligent agent perception, localization or control.

The particularities of the algorithm proposed in this thesis for autonomous navigation were presented in detail, reviewing the components that perform all the specific tasks necessary for the navigation process. At the end, all the ways in which the algorithm was validated, respectively the obtained experimental results, were mentioned.

Starting from the aim of the work, as previously mentioned, to realize an autonomous navigation system for mobile robots, it was necessary to define the component modules for such a system. So the perception module for the intelligent agent, the local trajectory planning module, the global trajectory planning module, the localization module, as well as the agent control module were implemented.

Local trajectory planning for the intelligent agent was done using the DWA method modified to incorporate the predictions provided by the artificial intelligence module. Thus, the planner benefited from the availability of additional information in the form of those predictions, managing to generate better trajectories both in terms of collision avoidance and the movement of the agent itself.

The control of the intelligent agent was realized using the MPC method, more specifically the variant for non-linear systems - NMPC. This type of algorithm is suitable for the execution of order quantities due to its nature which involves applying the process model to predict the time evolution of the controlled system (the motion of the intelligent agent in this case).

All experimental results were obtained using the ROVIS robot operating system, implementing the necessary interfaces for both the simulated environments and the robotic platforms used.

Original contributions. Published work

The present work is part of the general scope of interest for the field of Artificial Intelligence (AI), more specifically for AI techniques applied in the field of autonomous navigation. The topic covered involved the study, respectively the use of complex concepts with an interdisciplinary character, such as mathematics, Computer Vision (CV), AI, robotics and information technology.
The present work proposes an autonomous navigation system for mobile robots, based on a perception-plan-control type architecture, where the perception and trajectory planning components were designed using artificial intelligence. On the other hand
part, the autonomous navigation system has been implemented for both simulated environment and real robotic platforms.
Thus, the main personal contribution resulting from the research carried out during the doctoral studies is the conceptualization and implementation of an artificial intelligence algorithm that has the role of creating a model of sensory observations for a mobile robot based on historical observations. The model of sensory observations is used to predict the evolution of static and especially dynamic obstacles in the environment where the robot navigates, thus contributing to better trajectory planning, respectively superior control compared to other techniques used in the field.
The model of observations differs from what has been studied in the field both by the structure of the artificial intelligence component and by the mode of operation. Unlike other architectures used for predicting the evolution of the surrounding environment, where the object detection step, whether 2D or 3D, is performed first, and then the respective objects are tracked, the proposed algorithm directly predicts the observations of the autonomous agent, without performing the step of recognition of objects in the scene.
The algorithm used to model observations of the intelligent agent is based on a recurrent neural network, which receives as inputs sequences of historical data in the form of past observations from the distance sensor located on the agent and based on them produces estimates for future observations, performing dynamic coding obstacles present in the environment in which it navigates.
Another contribution resulting from the studies carried out is the embedding of the model of these sensory observations into the local trajectory planning component for the mobile robot. Thus, the planning component receives, in addition to information about the current state of the autonomous vehicle, predictions of the artificial intelligence algorithm. Thus, the local trajectory that the intelligent agent must follow takes into account both the obstacles close to the robot and those that may pose a danger to the agent in the near future due to their dynamics.
Also, the pre-processing of the sensory data used by the artificial intelligence algorithm was carried out, so as to improve the performance of the system, both in terms of accuracy and speed of execution. So, the input data in the form of point clouds were transformed into other representational structures, suitable to be processed by the neural networks used.
Another contribution came in the form of the autonomous navigation algorithms implemented on the platforms used for validation, namely the implementation of the interfaces for different sensors necessary for the perception of the navigation system, the implementation of the local trajectory planner, as well as the controller.
The proposed algorithm was implemented to be used in two simulated environments as well as on two robotic platforms. Thus, adaptations were needed in terms of interfaces for the sensors used for perception, as well as changes to the controls for the actuators, as well as for the localization system, these being different for each platform used.

The comparison of the proposed algorithm with other implementations from the specialized literature was made, for each hardware or software platform used. Challenging scenarios for the autonomous agent were designed and implemented in order to highlight the adaptability of the algorithm for different situations that may arise in navigation tasks. Thus, through experiments, the performance of the complete autonomous navigation system was ensured. The results obtained for the ObserveNet Control algorithm were validated by publication in the following specialized journals, respectively conferences:

- **C. Ginerica**, M. Zaha, L. Floroian, D. Cojocaru, S. Grigorescu, „A Vision Dynamics Learning Approach to Robotic Navigation in Unstructured Environments” in Robotics. 2024; 13(1):15, https://doi.org/10.3390/robotics13010015, IF=3.741.
- **C. Ginerica**, M. V. Zaha, F. Gogianu, L. Busoniu, B. Trasnea and S. M. Grigorescu, "ObserveNet Control: A Vision-Dynamics Learning Approach to Predictive Control in Autonomous Vehicles," in IEEE Robotics and Automation Letters, doi: 10.1109/LRA.2021.3096157, WOS:000678343900031, IF=5.79.
- **C. Ginerica**, D. Cojocaru, S. Grigorescu, „A Vision-Dynamics Learning Approach to Prediction-Based Control in Autonomous Vehicles”, International Symposium on Signals, Circuits & Systems - ISSCS 2021, SCOPUS.
- **C. Ginerica**, V. Isofache, S. Grigorescu, „Vision Dynamics: Environment Modelling, Path Planning and Control Based on Semantic Segmentation”, International Joint Conference OPTIM-ACEMP 2021, IEEE Xplore.
- S.M. Grigorescu, **C. Ginerica**, M. Zaha, G. Macesanu, B. Trasnea, ”LVD-NMPC: A Learning-based Vision Dynamics Approach to Nonlinear Model Predictive Control for Autonomous Vehicles”, Advanced Robotic Systems, Sage Journals, 2021, WOS:000660671900001 IF=1.652.
- B. Trasnea; **C. Ginerica**, M. Zaha, G. Măceşanu, C. Pozna, S. Grigorescu, OctoPath: An OcTree-Based Self-Supervised Learning Approach to Local Trajectory Planning for Mobile Robots. Sensors 2021, 21, 3606, IF 3.847, https://doi.org/10.3390/s21113606, Web of Science.