Research and optimization of perception and trajectory planning algorithms for autonomous mobile robots and road vehicles

Autonóm mobil robotok, valamint közúti járművek környezetészlelését és trajektóriatervezését megvalósító algoritmusok kutatása és optimalizálása

Ernő Horváth - Doctoral dissertation

Modeling and Development of Infrastructural Systems Doctoral School of Multidisciplinary Engineering Sciences

Content

1. Introduction

It is a fascinating fact that we humans can solve such complex tasks every day without much attention, whereas we can hardly imitate these behaviours with the most up-to-date sensors, actuators and algorithms. We are showing some serious motoric skills in hectic traffic, during preparation of a simple meal or during sport activity. On the other hand, non-complex algorithms easily overperforms us in complicated logic games or even intelligence tests. Similar paradoxical phenomenon was observed by Hans Moravec [1] in the '80s. According to Moravec's paradox, the imitation of a low-level sensory motor skills of up to a few years old can require enormous computing resources, while a simpler chess program can even easily overcome an adult. Evaluating the difficulty of a given task is not always clear or sometimes conditions are involved. In my research I always treated this fact humbled, and my aim is not to change this in the future. Therefore, for example, a mobile robot navigation in a factory or a self-driving vehicle in traffic is facing some serious tasks. In designing and operating such systems, sensing and perception are one of the most important tools. Therefore, my doctoral thesis also targets the problem of robotic perception especially new design perspectives and methodologies. Consequently, aimed to familiarize myself with the methodological approaches of the field and make proposals for their improvement.

The whole problem set is further refined by prof. Roland Siegwart, a recognized scientist in the field of artificial intelligence at University of Zürich. He has revealed some considerations [2] in 2018, according to which if it is not counted with the structured gradation of a given task, some of us may go too far when we put the achievements of NN on a pedestal and create false expectations. Nowadays NN systems can be applied only to structurable problems, but of course the research will always focus on stepping over or expanding these limitations. I used NN technology intensively in my own research, but always considering an important conclusion of the article mentioned above. Robotics will always use NN technology, but they also must outspread it.

Currently there is an extensive research tendency regarding autonomous (also known as self-driving) vehicles were the perception and path planning part of my doctoral work is also relevant. At the time of writing besides widespread academic or industry-related research there are several start-ups (Voyage, Waymo, Uber, Aptiv, Navya) who provide

autonomous taxi services in some cases even without safety drivers, but always with limitations. Also, in driving assistance (level 2) technologies are remarkably advanced, the most well-known example is Tesla. Drago Anguelov [3] who is a Principal Scientist at Google-Waymo views the situation as taming the long tail, because for the most common cases there are existing solutions, but solving extremities is a similar magnitude problem. So, the long tail of event means that it is a hard problem to build algorithms, train your neural networks for the common cases of autonomy, but it is the same or even challenging level of difficulty to solve the rest. In autonomous vehicles NN technology can be used in different levels such as in perception and in decision making. NN perception is quite common because large amount of - visual or LIDAR - data can be processed this way. Autonomous decision-making tasks such as path planning can also be solved with or without NN. During the years of my PhD studies I was lucky enough to work together with lots of talented students together on autonomous vehicles so some of my theoretical results also applied in practice. I would highlight the Szenergy team at our university who are competing in a word-wide autonomous competition with the help of many lecturers including myself.In order to bear out my research, I chose the means of comparing the methods acknowledged by the scientific community and the results of my own. Besides that, I also used different evaluation methods and I shared publicly the related source codes. During my PhD studies I was able to publish my results even from small conferences to recognized international or impact factor journals. During my PhD work I had the possibility develop with a numerous programming language, such as MATLAB, Python, C++, C# or LabVIEW. Even more libraries, technologies, APIs, SDKs, tools and middleware helped my work; including but not limited to: TensorFlow, Neural Network Toolbox, ROS, rviz, rqt_plot, V-REP, Gazebo, OpenCV, NumPy, Numba, PyQt, NI Real-Time Module, NI FPGA Module, CUDA and cuDNN.

My doctoral thesis has the following structure. After this introductory section, in the second section the related work and recent results is discussed. This section gives the information required to have an insight to the basis of my work. Here the historical background and the current state of the art is detailed about neural networks, path planning and map representations. My three theses are based on this knowledge, which can be also considered as an extension of this knowledge, and it is detailed in the third section. The rest of the document contains the conclusions, the outlook and the related publication list.

2. Related work and recent results

This section overviews the historical background and the current results the related fields to my PhD work. I intend to adumbrate a complete guide to the most important scientific fields. These fields involve but not limited to robotic and self-driving associated tasks such map representations, trajectory planning and execution and neural networks. Also, I briefly describe the vehicles, robots, sensors and hardware used during the validation of the proposed algorithms and methods.

As a rule of thumb, I intend to illustrate every concept with *own measurements and graphics*. The section gives a general intuition about his field but focuses on the closely related architectures of the doctoral work. In addition, the limits and the drawback of the techniques is discussed alongside with some possible solutions.

2.1. Robots, vehicles and hardware used

During my PhD work I was lucky enough to collect experiences from variety of robots and vehicles. I always had possibility to try out my new algorithms on mobile robots such as Khepera III, Neobotix MP500 and TurtleBot 3. These robots were MATLAB and ROS compatible, if not by default, after modification. I also had access to the research centre's vehicles, the Szenergy vehicles, and the Nissan Leaf.

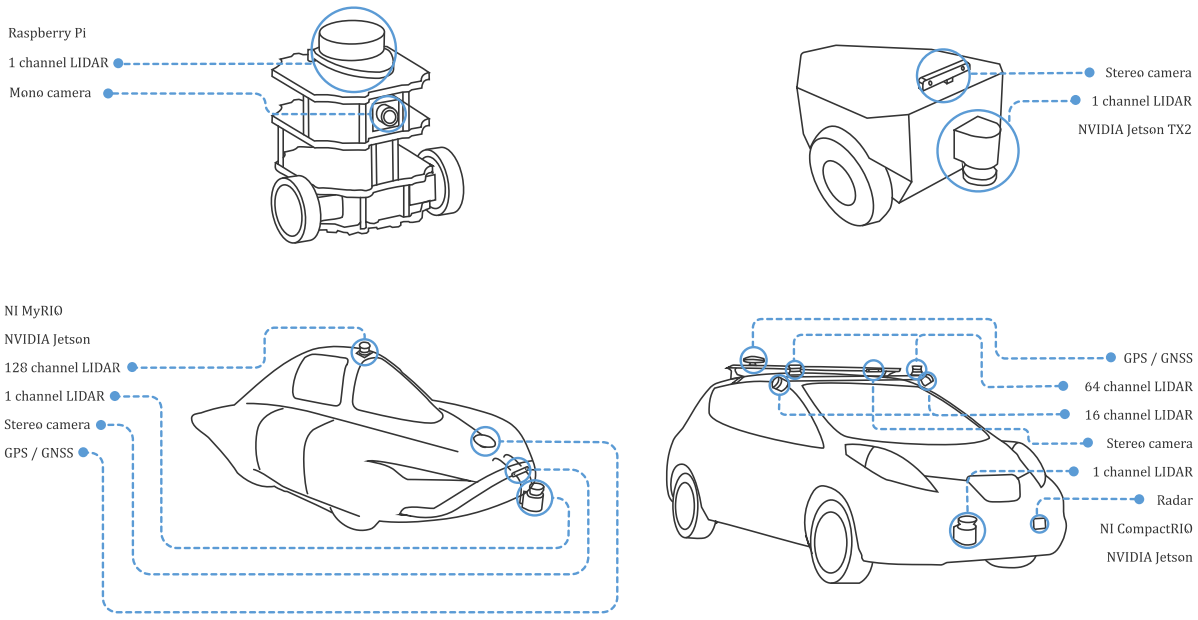
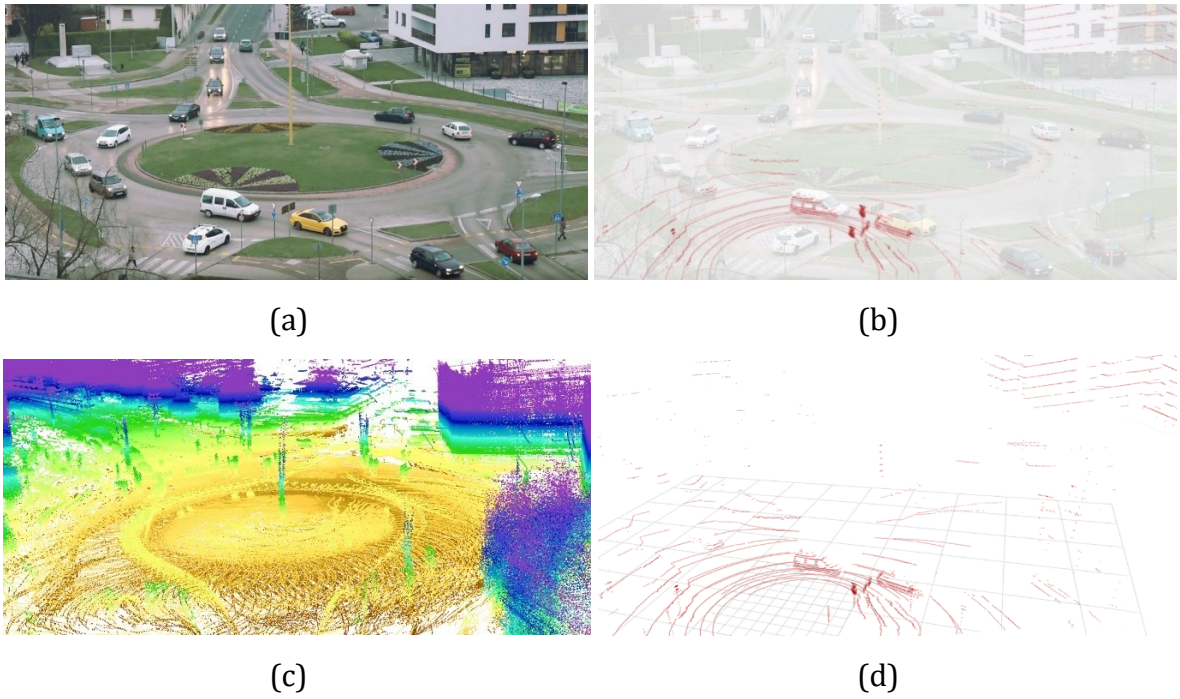


Figure 1 - Robots and vehicles: Turtlebot, Neobotix, Szenergy and Nissan Leaf

These robots and vehicles used different sensors, hardware and architectures. This was often a challenge, but as a result of working with these technologies I learned how to get along with a variety of tools. To arbitrary highlight the most distant technologies I used FPGA and real-time OS for the most time critic control and data acquisition tasks, on the other hand I used different script languages for visualization and data processing. From perception point of view, I used LIDAR, radar, ultrasonic and (depth) camera sensors.



(a) (b)

(c) (d)

Figure 2 - An example scenario: the Nissan equipped with 2x16 channel LIDARs (a) RGB image (b) point cloud and RGB (c) concatenated point cloud (d) single measurement – at the Széchenyi University Campus, Győr

In the discussed theses I mostly rely on LIDAR, camera and GPS data and from these sources my algorithms realize perception, trajectory planning and trajectory execution (following) for self-driving vehicles and autonomous robots.

From a system architecture point of view the components of an autonomous vehicle or robot can be described as follows. The main modules are sensing, perception, planning and following. The sensor data is collected in the by sensing module, this means a simple data acquisition task. The perception module contains more complex processes, i.e. the interpretation of the collected data. Planning deals with global planning, i.e. how to generally reach point B from point A; while local planning is restricted to re-plan the trajectory in the sensory range. If a given path or trajectory plan is current, the execution of this plan will be done by the following component.

These modules can be further divided into sub-modules and components.

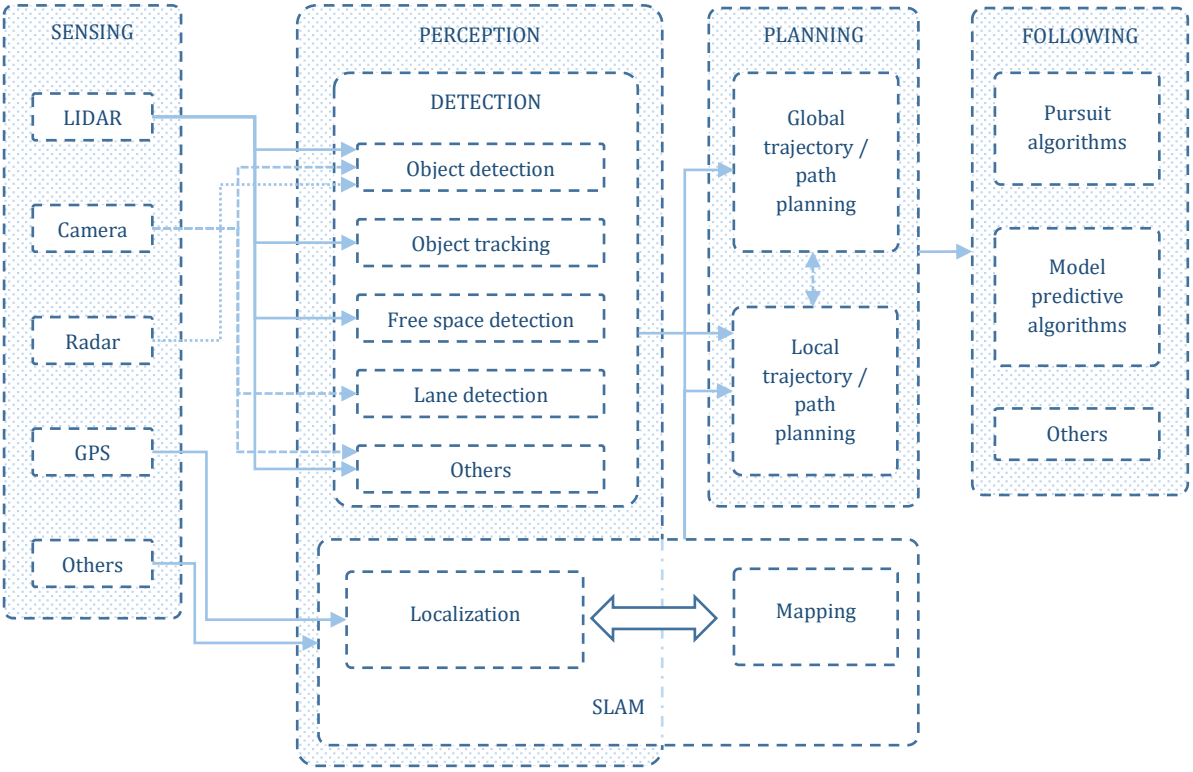


Figure 3 - High level description of autonomous vehicle (robot) components

The sensing module contains the mentioned sensors: LIDAR, camera, radar, GPS and other components. An example for other sensors is the odometry. This can be obtained from the calculation of various kinematic or dynamic models (e.g. bicycle model of a car, or differential model of a differential robot). The perception module is divided into detection and localization sub-modules. Detection deals with object detection and tracking, free-space detection, lane detection etc. Localization can be done by itself, but also commonly it is part of Simultaneous Localization and Mapping (SLAM). The trajectory / path execution can be done by various pursuit or by model predictive methods.

2.2. Map representations as environmental model

For the autonomous or semi-autonomous tasks of vehicles and mobile robots, the first step is to get to know the environment and to create a model of it e.g. to build your own map. Such algorithms first appeared in mobile robots ahead of vehicle applications, especially for indoor applications in the 1980s. Taking into account just the most important techniques [4] there are topological maps, line maps, landmark-based maps and occupancy grid map representations. These early applications have already taken

advantage in the certain attributes of the robot's environment that it can be modelled into a vertical 2D data structure, which uses the ground floor as a reference. This recognition led to one of the first such map representations, known as the occupancy grid map. Another distinguishing feature of the indoor environment is that it often contains straight boundary surfaces (walls, furniture, etc.). As a result of this recognition, line-based map representation has been developed. The next major step forward was the introduction of uncertainty and probabilities in the model. The results of the Hungarian Kálmán Rudolf Emil, especially the Kálmán filter is used then and still today [5] [6]. The next step, going beyond the problem of indoor mapping was outdoor mapping. In the meantime, the sensors have also evolved, making it possible to implement map building for less structured outdoor environments. It was necessary to supplement the 2D representation with elevation data, which made the management of probabilities and uncertainties more complicated in the model. Nowadays, the significant part of autonomous vehicle applications had exceeded the 2D representation with altitude data and use real 3D representation, such as point clouds, 3D grids, or meshes. This, of course, means a much larger amount of data and a greater amount of time required for processing. By eliminating this deficiency, the data collected are classified at multiple levels. Such a level can be the grouping required for navigation, the grouping of the landmarks of the environment, the grouping of objects according to their role, and the grouping of moving and static objects.

2.2.1 Line map

Line map is a popular [4] 2D environment representation technique, which have numerous advantages over these nonparametric representations. The basic idea behind the concept is to measure distance with a distance sensor (sonar or LIDAR), transform measurement into the 2D space (Cartesian coordinates) and finally align lines to the measurement. These sets of lines are often called as a multi-line-segment or less often polygon maps. With the mentioned methodology, which is related to regression, line maps may require less memory than grids representation and therefore scale better with the size of the environment. Moreover, they can be even more accurate if the alignment algorithm is well defended and the environment has advantageous characteristics. In that way they can even eliminate discretization problems.

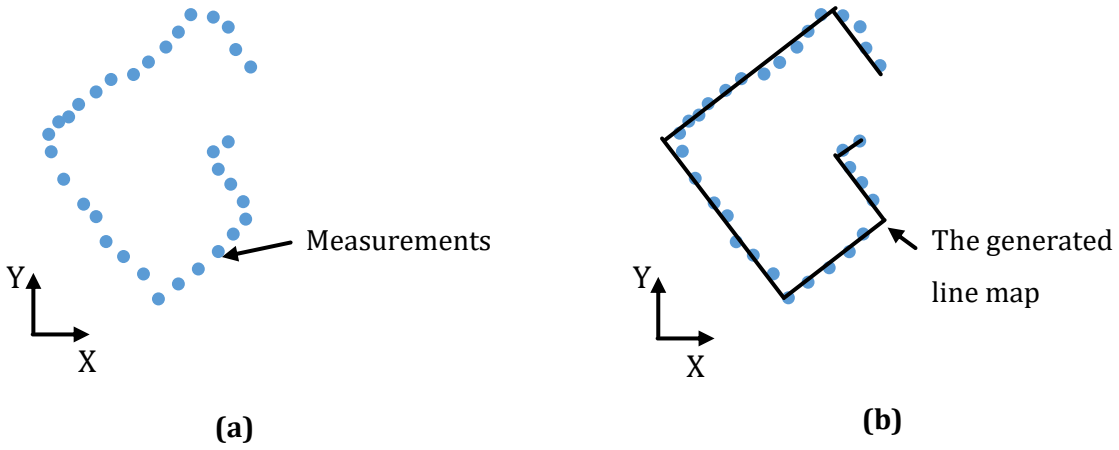


Figure 4 - Alignment of lines (b) to measurements (a) in Cartesian coordinates

There are several approach in order to align - with another terminology merge or fit - every single line for range scans with multiple linear structures. One of the most popular approach to solve this problem [4] is known as the split-and-merge algorithm. The main concept is to recursively subdivide the point set into more subsets; these subsets can be more precisely approximated by a single line. Whereas the split-and-merge algorithm is quite effective, it is not guaranteed that the result is optimal. Other methods use clustering techniques or weighted line fitting to determine multi-line-segment merging. One of these approaches is called k-means clustering. This algorithm aims group measurement points into clusters where each measurement point belongs to the cluster with the nearest mean, serving as a prototype of the cluster. In this case of course the number of lines approximately needs to be known in advance. There are approaches which are based on random sample consensus (RANSAC) which is also an iterative method to approximate multi-line-segment and ignore outliers.

2.2.2 Landmark-based map

To environments which has certain characteristics, for example locally distinguishable features [4], landmark-based maps have been extensively used. These kind of environments can be trees in forest, rocks in desert, etc. The main difference between landmark-based and other map types is that it is bounded more closely to the localization problem. The reason comes from the fact that is this kind of maps were developed to contain landmark-based features. These features needs to be extracted from sensory data. This phase of the algorithm is the so-called data association phase. After that, the

produced features or landmarks serve as a reference for localization. The map building and localization together is called SLAM (Simultaneous Localization And Map building) and will be discussed in section 2.3.9. It should be noted that landmark-based map only works well, when the environment has unique features.

2.2.3 Topological map

The basic idea behind topological map [4] is instead of focusing on the geometric structure of the map, it considers the connection between unique places in a form of a graph. One of the first realization to topological mapping is the work by Kuipers and Byun [7]. Topological maps are treating the environment in a graph-like structure, in which the *nodes* are locally distinguishable, unique places and the *edges* are the connections between the places.

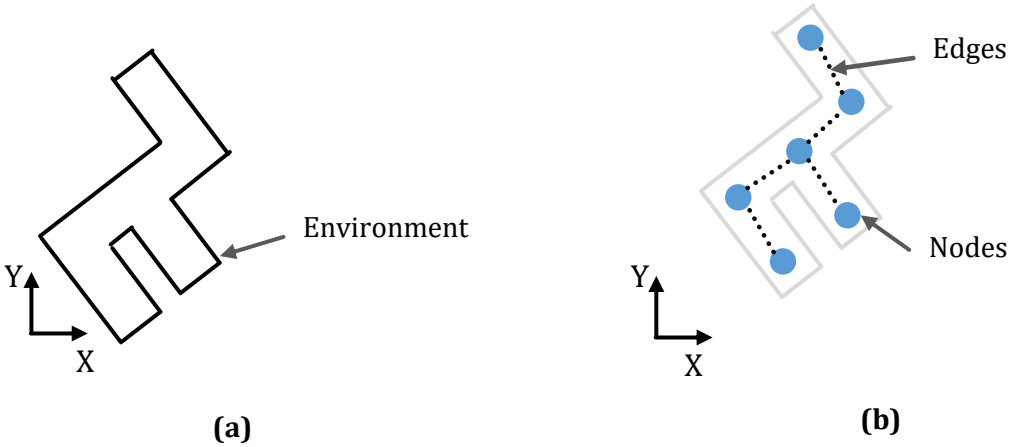


Figure 5 - The original environment (a) a topological map fitted to the environment (b), the circles are the nodes, dotted lines are the edges

Considering a simple case, a topological map is for example the metro map. You cannot directly fit the metro map to the map of the city, and the reason for that is because the metro map only considers connection between the stations, not the exact location of the stations. For a metro-user this simplified map is more understandable. The stations in this example are the nodes, and the connection between them are the metro tracks. Likewise, the topological map on considers connection between certain places, the nodes are places which a robot or vehicle can visit and the route between them are the edges. Because of this, topological maps are excellent for route planning even if the environment is large.

The only drawback of this approach is that usually another environment representation is needed in order to precisely navigate.

2.2.4 Cost map

Cost maps can be 2D or 3D maps, and the main purpose is to serve as a collision avoidance map [4]. This type of map is usually used along with a traditional map representation (e.g. line map or occupancy grid map) and also along with the kinematical constrains of the robot or vehicle. Like the landmark-based maps are closely related to the localization problem, the cost map is heavily related to the planning, especially the local planning problem. The costs are computed by comparing the local obstacles with the kinematic constraint of the robot or vehicle. These kinematic constraints involve the geometric restrictions of the robot, the footprint model (collision shape), and the non-holonomic characteristics of it. The cost map can be a really effective way to calculate the local trajectory if certain parameters such as closeness tolerance, resolution of the cost map etc. are set properly. This is important because otherwise a robot or vehicle can plan dangerous manoeuvres or even damaging something.

2.2.5 Occupancy grid map

The essence of the occupancy grid map representation [4] is to divide the environment into discrete square-based grid points (in other words cells or pixels) and assign to them occupied or free values. Compared to landmark-based, an occupancy grid makes no hypothesis about features or landmarks. A modified version of this is probabilistic occupancy grid, where these values are not binary; they represent a probability value.

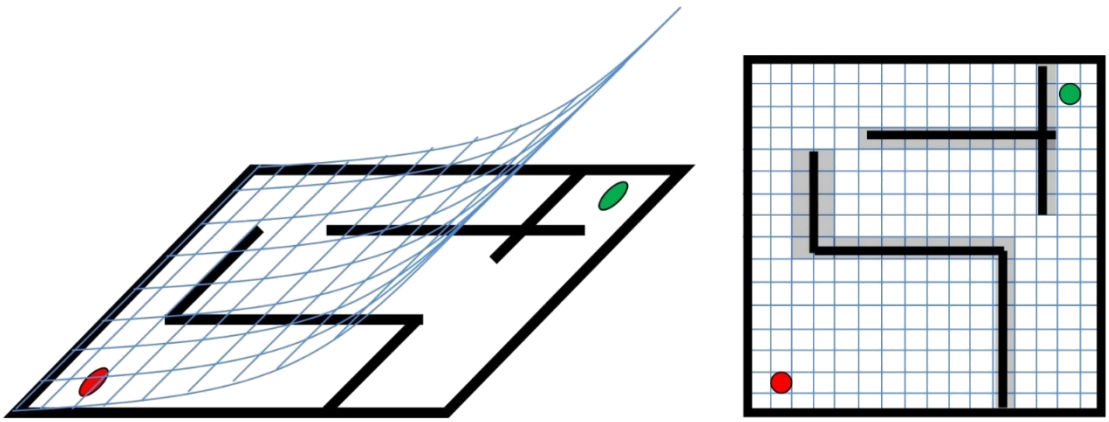


Figure 6 - Occupancy grid map

Occupancy grid maps, which were introduced in 1985 by Moravec and Elfes [8] are dominant and popular [9] [10] probabilistic approach to represent the environment. The resolution of the occupancy grid map is a trade-off. The larger the resolution, the more memory and computation needed. On the other hand, if the resolution is too low obstacles which may are important are not observable on the map. Figure 7 shows this phenomenon. Occupancy grid or as sometimes referred metric maps are representing each coordinate as a grid on certain resolution, thus they can be efficiently updated. In this representation the grid cells are always available for update or query. The other advantage is that it provides the ability to represent unobserved areas, so it can build a map even no a priori map existed.

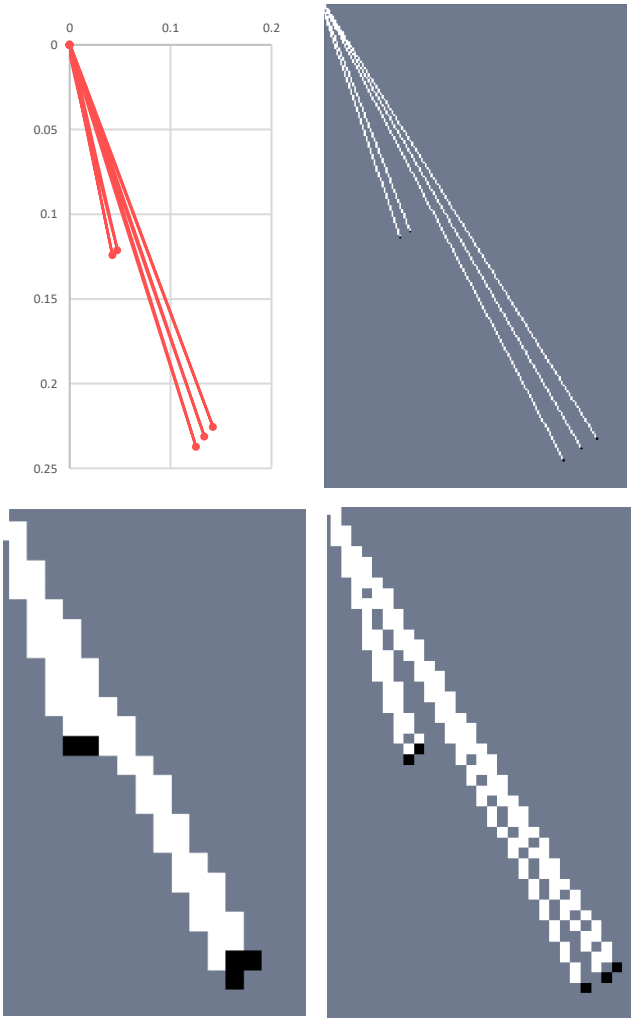


Figure 7 - Occupancy grid of 5 distance measurements (first image) on 3 different resolution (last 3 images)

There are of course disadvantages; the first one is quite obvious: the bigger the map the higher the memory allocation applies. A naïve attitude may suggest this is the weakness of every representation, but that is not true. For example topological and line maps can manage the memory in some cases in a more efficient way. The second disadvantage comes from its discrete characteristic, but the discretization errors can be typically avoided if the correct grid cell size is chosen for the given task.

2.2.6 Signatures

In this section the signatures and operators respect to them are described. Signatures are mathematical constructions which are able to describe symbolic data representations, they can be used even for complex data manipulation, such as expert systems or like in this case, robotics. Signatures are mathematical constructions which are able to describe symbolic data representations. The definition can be expressed as a recursive definition of the set $S^{(n)}$.

Let **R** be the set of real numbers. The set $S^{(n)}$ is defined recursively as $S^{(n)} = \prod_{i=1}^{n} S_i$. Here $S_i = \mathbf{R}$, $i = \overline{1,n}$ (i.e., $i \in \{1, 2 \dots, n\}$), or $S_i = S^{(n)}$ $m \geq 1$, and $\prod$ is the Cartesian product [11].

Let X be a nonempty set [11]. The collection of signatures is defined as the function $A: X \to S^{(n)}$, and the signature of the element $x \in X$ is $A(x)$ given as

$$A(x) = \begin{bmatrix} \dots \\ \begin{bmatrix} a_{i,1} \\ a_{i,2} \end{bmatrix} \\ \begin{bmatrix} a_{i+1,1} \\ \begin{bmatrix} a_{i+1,2,1} \\ a_{i+1,2,2} \\ a_{i+1,2,3} \end{bmatrix} \\ a_{i+1,3} \end{bmatrix} \\ \begin{bmatrix} a_{i+2,1} \\ a_{i+2,2} \end{bmatrix} \\ \dots \end{bmatrix} \quad (1)$$

The signature values are the elements, which appear in the signature defined in (1), for example $a_{i,1}, a_{i,2}, a_{i+1,1}$, and so on. The transposition of the signature $A(x)$ is $A^T(x)$.

$A^T(x) = \left[\dots [a_{i,1}, a_{i,2}\right]$

$\left[a_{i+1,1} \left[\left[a_{i+1,2,1}, a_{i+1,2,2}, a_{i+1,2,3}\right] a_{i+1,3}\right]\right.$

$[a_{i+2,1}, a_{i+2,2}] \dots]$ (2)

The following notations can be used to simplify the characterization of signatures [11].

A signature $A(x)$ with the values $a_1, a_2, \dots, a_n, a_{i,1}, a_{i,m}, a_{j,k,l}, \dots$ is expressed as $a^{\cdots}$.

The other simplified notation happens when $\exists x \in X$ and $A^T(x) = [a_1 \dots a_n]$. In this case the notation $A(x) = a^{1,\dots,n}$ can be used.

Another case happens when $\exists y \in Y$ and $A^T(y) = \left[a_1 \dots\ a_{i-1}\ \left[a_{i,1} \dots\ a_{i,m}\right] a_{i+1} \dots\ a_n\right]$, then it can be written as $A(y) = a^{1,\dots,[1,\dots,m]i,\dots,n}$. In this case the sets are defined as $S_1 = S_2 = \cdots = S_{i-1} = S_{i+1} = \cdots = \cdots = S_n = \mathbf{R}$, and $S_i = \prod_{i=1}^{m} \mathbf{R} = \mathbf{R}^m$.

A signature of type [... $[[a_1]]$ ...] is equivalent with the signature $[a_1]$, where $a_1 \in \mathbf{R}$. Two signatures $a^{\cdots}$ and $b^{\cdots}$ have the same structure if and only if [11] for each value $a_{i_1,\ i_2,\ \dots, i_S}$ $\mathrm{a_{i_1 i_2 \dots i_s}}$ of the signature $a^{\cdots}$ there exists the value $b_{i_1,\ i_2,\ \dots, i_S}$ of the signature $b^{\cdots}$.

A signature can be manipulated with basic operations, such as contraction, extension, pruning, and grafting, also with complex operations such as addition and multiplication. In robotics not every operation is used so in the current thesis only the most relevant operations are described. For example, the extension of a signature is defined as the function

$g_{\overline{@}_i^p}: S \to S,$

$$\begin{cases} g_{\overline{@}_i^p}(a^{1,2,..,i,..n}) = a^{1,2,..,[i,..p]..n}, & if\ i \le n, \\ g_{\overline{@}_i^p}(a^{\cdots}) = a^{\cdots} & otherwise \end{cases} \quad (3)$$

where $g: \mathbf{R} \to \mathbf{R}^p, g(a_i) = [a_{i1}, \dots, a_{iq}]$, or

$g_{\overline{@}_{i,j,k}^q}: S \to S,$

$$\begin{cases} g_{\overline{@}_{i,j,k}^q}\left(a^{1,..[1,..[1,..k,..r,..]_j,..,m]_i,..n}\right) = a^{1,..[1,..[1,..q]_k..,r]_j...m]_i,..n}, & if\ i \le n, j \le m, k \le r, \\ g_{\overline{@}_i^p}(a^{\cdots}) = a^{\cdots} & otherwise \end{cases} \quad (4)$$

where $g: \mathbf{R} \to \mathbf{R}^p, g\left(a_{i,j,k}\right) = [a_{i,j,k1}, \dots, a_{i,j,kq}]$, and we can continue to generalize it by adding extra indices after i, j and k. Extension allows to a scalar element of a signature to extend it into a vector. The opposite of extension is contraction (compression) of a nested

vector which belongs to the signature structure. With this operator the entire vector or a nested vector is transformed into a scalar. This is defined as either the function

$$^{f}@: S \to S,\ ^{f}@(a^{1,2\dots,n}) = a^{1} = [a]\,, \tag{5}$$

where $a = f(a_1, a_2, \dots, a_n)$, $f: \mathbf{R}^n \to \mathbf{R}$, or the function

$$^{f}@: S \to S, \tag{6}$$

$$\begin{cases} ^{f}@_i\,(a^{1,..[i..m]i,..n}) = a^{1,..,i,..n}, & if\ i \le n, \\ \quad ^{f}@_i\,(a^{\cdots}) = a^{\cdots} & otherwise \end{cases}$$

where $a = f(a_{i1}, \dots, a_{im})$, $f: \mathbf{R}^m \to \mathbf{R}$, or the function. These operations and the corresponding signatures can handle complex data.

2.3. Path planning

There are many algorithms, algorithm variants, methods which intend to solve the coverage path planning problem (CPP), such as Random Path Planning [12], Exact Cellular Decomposition [13], Boustrophedon Cellular Decomposition (BCDC) [13], Backtracking Spiral Algorithm (BSA) [12] Internal Spiral Search (ISS) [14] U-turn A* Path Planning (UAPP) [14] or the Neural Network Approach [15]. The problem can be briefly summarized as covering an area (map) with the least possible traversal. There are many forms of the initial problem. Coverage path planning (CPP) deals with the problem of crawl routing in a particular area, most often in the domain of a mobile robot (MR). The problem itself can be categorised many ways, e.g. it can be *complete* or *heuristical*. The complete coverage, as the name implies, will guarantee that the entire map area will be covered. On the other hand, the heuristic coverage the full coverage is conceivable, but not at all guaranteed. It can be grouped from another perspective: *offline* if the area is known in advance, otherwise it is *online* if the robot needs to discover its environment or a part of it at least. The path planning of the mobile robot is described with the movement of it which is related to kinematics of it. Kinematics defines the mechanical behaviour of the robot regarding its speed and without its mass. On the other hand, if other additional constraints due to mass and force considerations involved the dynamics can describe the motion. Regarding the thesis, the differential model of a mobile robot and the bicycle model of a car-like vehicle is considered.

2.3.1 Kinematics of a differential robot

In this section one of the most common type of kinematics, the kinematics of a Differential Robot (DR) is discussed. The DR means that a mobile robot has two wheels which cannot be turned, but can be separately controlled via two speed references; an additional support wheel (also known as caster wheel) is either present or not. When the control and navigation of the model is executed, practically the geometric model is simulated, which results the pose, consisting of orientation, and position. This also means that higher levels of abstraction can be used.

Figure 8 visualizes three different representation at different abstraction levels. The first abstraction is a schematic top view of the robot, basically a footprint model. The second abstraction replaces the previous schema with a simplified triangular representation, and the third one uses only one reference image to approximate the robot's position.

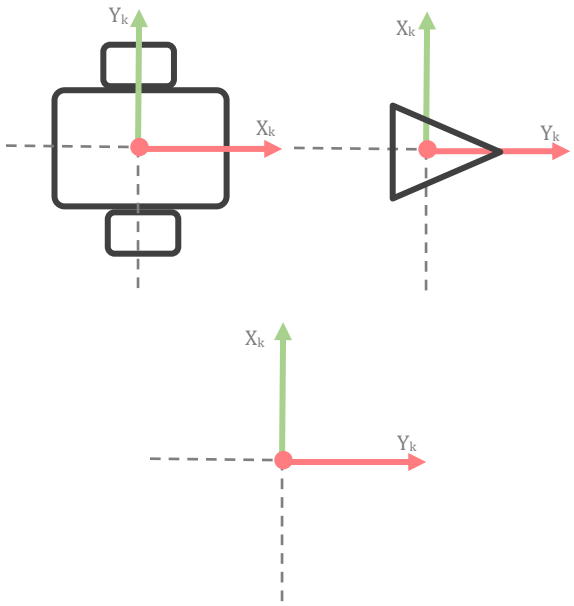


Figure 8 - DR geometric model (GM) representation illustrated on various abstraction levels

The Geometrical model (GM) methods refers to:

The pose computation into an absolute referential system. This means the possibility to represent the model for different position and orientation of the DR. So the inputs of the method can be:

- The DR referential position ${}^{k}_{0}\mathbf{t} = [t_{x,k} \quad t_{y,k} \quad t_{z,k}]^T$

- Orientation α_k, relative to the absolute referential frame, for an absolute pose computation (see Figure 10)
- The outputs are the new coordinate of the GM points. The method consists on a homogenous transformation

$$\mathbf{x}_0 = {}^{0}_{k}\mathbf{T}\mathbf{x}_k \tag{7}$$

where:

$${}^{k}_{0}\mathbf{T} = \begin{bmatrix} c\alpha_k & -s\alpha_k & 0 & t_{x,k} \\ s\alpha_k & c\alpha_k & 0 & t_{y,k} \\ 0 & 0 & 1 & t_{z,k} \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$\mathbf{x}_k = \begin{bmatrix} x_{0,k} & x_{1,k} & . & . & . & x_{5,k} \\ y_{0,k} & y_{1,k} & . & . & . & y_{5,k} \\ z_{0,k} & z_{1,k} & . & . & . & z_{5,k} \\ 1 & 1 & . & . & . & 1 \end{bmatrix};$$

$$\mathbf{x}_0 = \begin{bmatrix} x_0 & x_1 & . & . & . & x_5 \\ y_0 & y_1 & . & . & . & y_5 \\ z_0 & z_1 & . & . & . & z_5 \\ 1 & 1 & . & . & . & 1 \end{bmatrix};$$

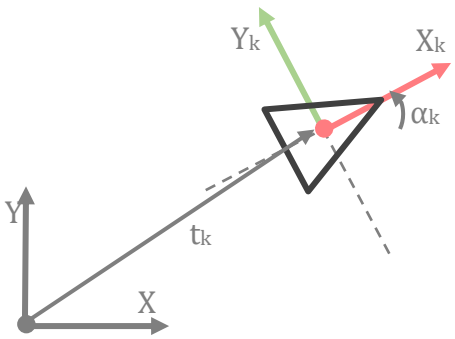


Figure 9 - GM pose in the absolute coordinate system, the referential frame

A The DR referential position ${}^{k+1}_{k}\mathbf{t} = [tx_{k,k+1} \quad ty_{k,k+1} \quad tz_{k,k+1}]^T$ and orientation $\alpha_{k,k+1}$ relative to the previous pose, for a relative pose computation, as visible on Figure 9.

$$\mathbf{x}_{k+1} = {}^{0}_{k+1}\mathbf{T}\mathbf{x}_k = {}^{0}_{k}\mathbf{T} \cdot {}^{k}_{k+1}\mathbf{T}\mathbf{x}_0 \tag{8}$$

where:

$$ {}^{k}_{0}\mathbf{T} = \begin{bmatrix} c\alpha_{k,k+1} & -s\alpha_{k,k+1} & 0 & tx_{k,k+1} \\ s\alpha_{k,k+1} & c\alpha_{k,k+1} & 0 & ty_{k,k+1} \\ 0 & 0 & 1 & tz_{k,k+1} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The following observations are necessary: the chosen GM is a planar model this means that the z coordinate of all points are constant (usually zero); the orientation of the GM refers to rotations around the {Z} axis.

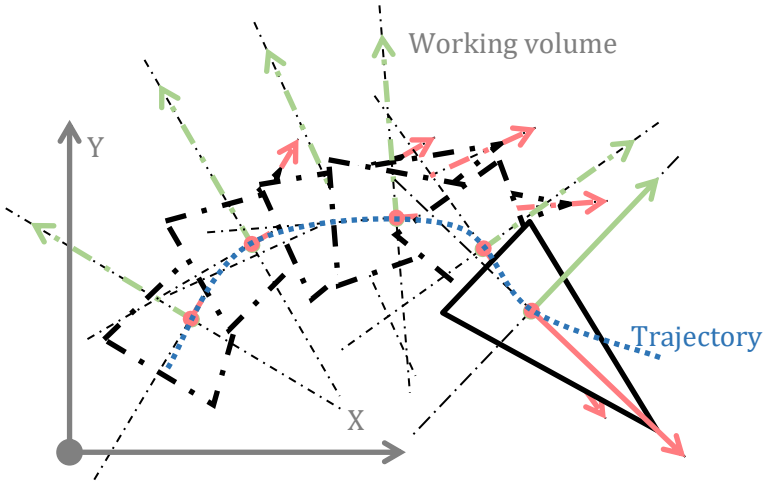


Figure 10 - Working volume and trajectory

Figure 10 illustrates the GM trajectory and working, or the sake of clarity, it is important to emphasize that in this context, trajectory means a path associated with a time dependency. The path is an ordered set of spatial position and orientation pairs defined in the operational space, describing the movement of the DR, in other words:

- Trajectory consists of sequential series of time-discrete vectors that are trained by the origin of the reference system to describe the movement of the robot or vehicle
- The working volume is the superposition of the values recorded by the DR, which the DR reaches during the movement.

When describing the kinematic model, the hypothesis, which supports the DR model, are determined by left and right wheel speeds. The inertial properties of DR do not affect the movement phenomenon. As a result, it can be assumed that the reasons for the change in position and orientation are only the wheel speeds responsible. Given the speed and direction of both wheels' rotation, it can be observed that in case the wheels do not rotate

at the same speed, the DR turns into the direction of the slower rotating wheel. The centre of this turn movement is always somewhere on the common axes designated by the wheels. This point is called the Instantaneous Centre of Curvature (ICC) [16]. E.g. in case the wheel speeds (v_r, v_l) are the same but the direction is the opposite, the ICC will be exactly between the two wheels, on the common axes, and the robot will rotate around itself. Another typical case happens when the wheel speeds and directions are the same. In this case the ICC will be infinitely far on the common axes, as a result the robot moves on a straight line. To imagine this phenomenon better, the robot still moves around circle which centre is the ICC; but this time the radius of this circle is infinitely big, so the movement becomes a straight line.

$$v_r; v_l \Rightarrow t, \alpha \tag{9}$$

where: $v_r; v_l$ are the right and left wheels speeds

In order to solve equation (9) another hypothesis that of DR in connection with a planar motion it will be introduced. Based on this assumption Figure 11 illustrate the named solution.

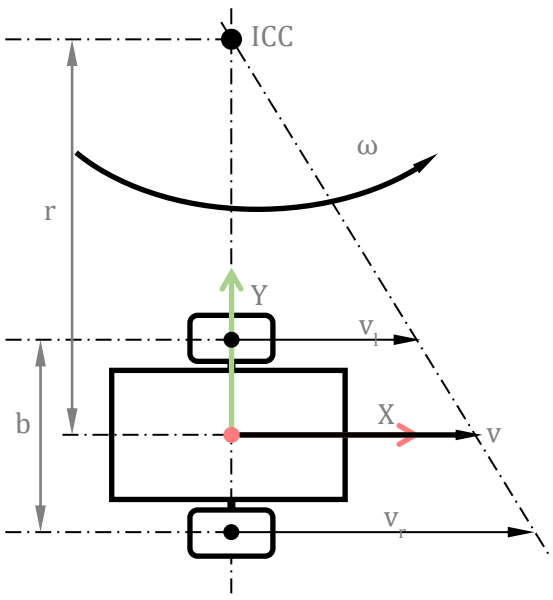


Figure 11 - DR velocity vectors

$$\begin{cases} v = \frac{v_r + v_l}{2} \\ \omega = \frac{v_r - v_l}{b} \end{cases} \tag{10}$$

where: ω is the angular speed of DR, b is the distance between the two wheels (see Figure 11).

Additionally, the radius of the gyration is:

$$r = \frac{v}{\omega} = \frac{b}{2}\frac{v_r + v_l}{v_r - v_l} \tag{11}$$

Before kinematical model is derived, an additional understanding is important: most time the input data for the model are not the linear speed but the angular speed of the wheels. Figure 12 gives information about the link between these speeds.

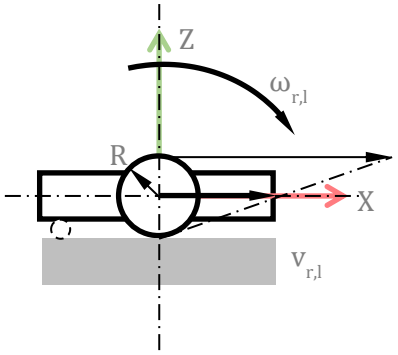


Figure 12 - Demonstration the side view of the DR and the correlations of velocities

In the case $v_r = v_l$, the robot moves in a straight line with uniform motion. Then r will be virtually infinite, and ω is zero. Another typical use case is when the robot needs to be rotated around its own axis. Then $v_r = -v_l$ becomes $r = 0$, and the robot can turn around in a $\frac{b}{2}$ radius circle. The model with the relationship between linear speed and wheel speed is:

$$v_{r,l} = R\omega_{r,l} \tag{12}$$

And now the final step in deriving the outputs (see Figure 12):

$$\begin{cases} \dot{t} = \begin{bmatrix} v \cdot \cos(\alpha) \\ v \cdot \sin(\alpha) \end{bmatrix} \\ \dot{\alpha} = \omega \end{cases} \quad (13)$$

This means:

$$\begin{cases} \dot{t}_x = \frac{v_r + v_l}{2} \cdot \cos(\alpha) \\ \dot{t}_y = \frac{v_r + v_l}{2} \cdot \sin(\alpha) \\ \dot{\alpha} = \frac{v_r - v_l}{b} \end{cases} \quad (14)$$

Two problems can be defined with (14). The first is the direct kinematic problem where the input (known) data are (v_r, v_l) and the output (unknown) data are (t_x, t_y, α). The second is the inverse kinematic problem where (t_x, t_y) are the input data and (v_r, v_l) are the output data.

The solution of the first problem is needed for the simulations of the DR behaviour. The second problem imply the open loop hypothesis i.e. if the trajectory of the robot is impose the causes (v_r, v_l) which produce this trajectory can be computed. Solving this problem is a nice exercise but the mentioned hypothesis is naive (because of the perturbation which can be rejected only by the close loop).

The numerical solution of direct kinematic problem can be formulated (15) with the Euler approximation.

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \begin{bmatrix} \delta_d \cdot c(\alpha + \delta_\alpha) \\ \delta_d \cdot s(\alpha + \delta_\alpha) \\ \delta_\alpha \end{bmatrix} \quad (15)$$

where: $\mathbf{x}(k) = \begin{bmatrix} t_x(k) \\ t_y(k) \\ \alpha(k) \end{bmatrix}$, $\begin{cases} \delta_d = v_k \cdot \Delta t \\ \delta_\alpha = \omega_k \cdot \Delta t \end{cases}$, Δt sampling time.

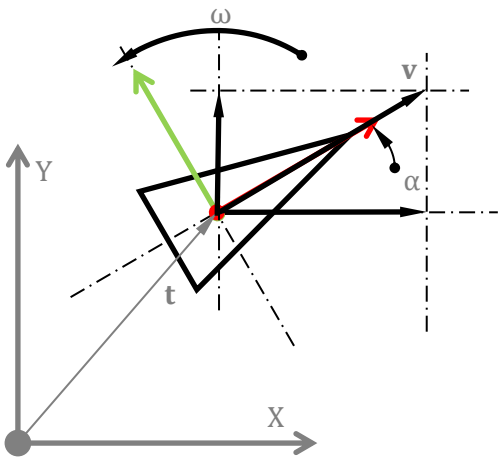


Figure 13 - Kinematics of the DR

The numerical solutions for inverse kinematic starts from the input data (t_x, t_x) and compute the left and right wheel speeds:

$$\begin{cases} \dot{t}_x(k) = \frac{t_x(k+1)-t_x(k)}{\Delta t} \\ \dot{t}_y(k) = \frac{t_y(k+1)-t_y(k)}{\Delta t} \end{cases} \tag{16}$$

$$\alpha(k) = arctg\,\frac{t_y(k+1)-t_y(k)}{t_x(k+1)-t_x(k)} \tag{17}$$

$$\dot{\alpha}(k) = \frac{\alpha(k+1)-\alpha(k)}{\Delta t} \tag{18}$$

$$\begin{cases} v(k) = \frac{\dot{t}_x(k)}{\cos(\alpha(k))} \\ v(k) = \frac{\dot{t}_y(k)}{\sin(\alpha(k))} \end{cases} \tag{19}$$

$$\omega(k) = \dot{\alpha}(k) \tag{20}$$

$$\begin{cases} v_l(k) = \frac{v(k)-b\omega(k)}{2} \\ v_r(k) = \frac{v(k)+b\omega(k)}{2} \end{cases} \tag{21}$$

The acceleration of the DR has two components: the tangential acceleration and the normal acceleration. The importance of these concepts consists on illustrating the directions and the measure of the kinematic model perturbation. If the model hypothesis

is accepted, we will confront with perturbations like slippages in n and τ directions. Larger accelerations imply larger perturbations (slippages).

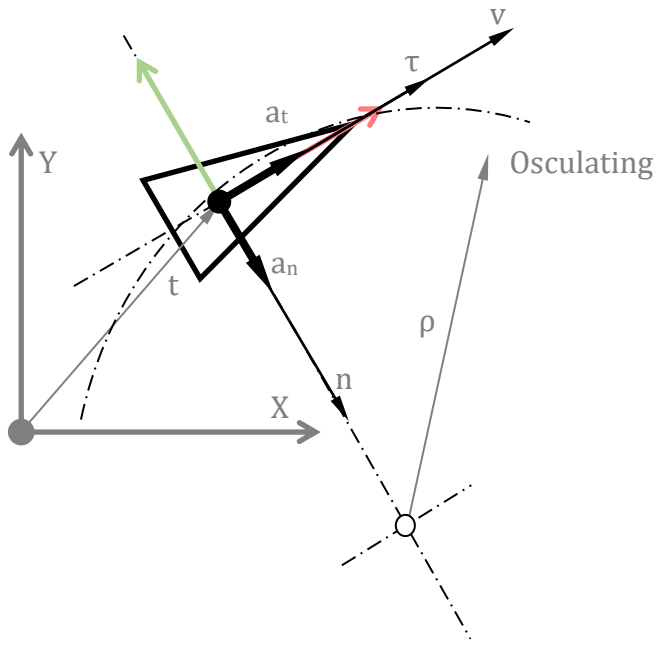


Figure 14 - Acceleration of the DR

$$\mathbf{a} = \mathbf{a_t} + \mathbf{a_n} = \frac{dv}{dt}\tau + \frac{v^2}{\rho}\mathbf{n} \tag{22}$$

$$\rho = \left| \frac{(\dot{x}^2(t) + \dot{y}^2(t))^{3/2}}{(\dot{x}(t)\ddot{y}(t) + \ddot{x}(t)\dot{y}(t))} \right|$$

where: the trajectory equation $\mathbf{t}(t) = [x(t) \quad y(t)]^T$; and $\boldsymbol{t}$ is the trajectory parameter. As a remark, acceleration and perturbation are sometimes not involved into the path planning problems some e.g. in certain coverage path planning problems.

2.3.2 Kinematics of a car-like vehicle, the bicycle model

The other common kinematics model which describes the car-like robots is called the Bicycle Model (BM) [17]. The vehicle or robot in that case has four wheels where the change of direction is achieved by rotating the front wheels. To prevent slipping of the front wheels, usually Ackermann steering is applied. In accordance with the working philosophy the mentioned type of model is initiated by a geometrical model.

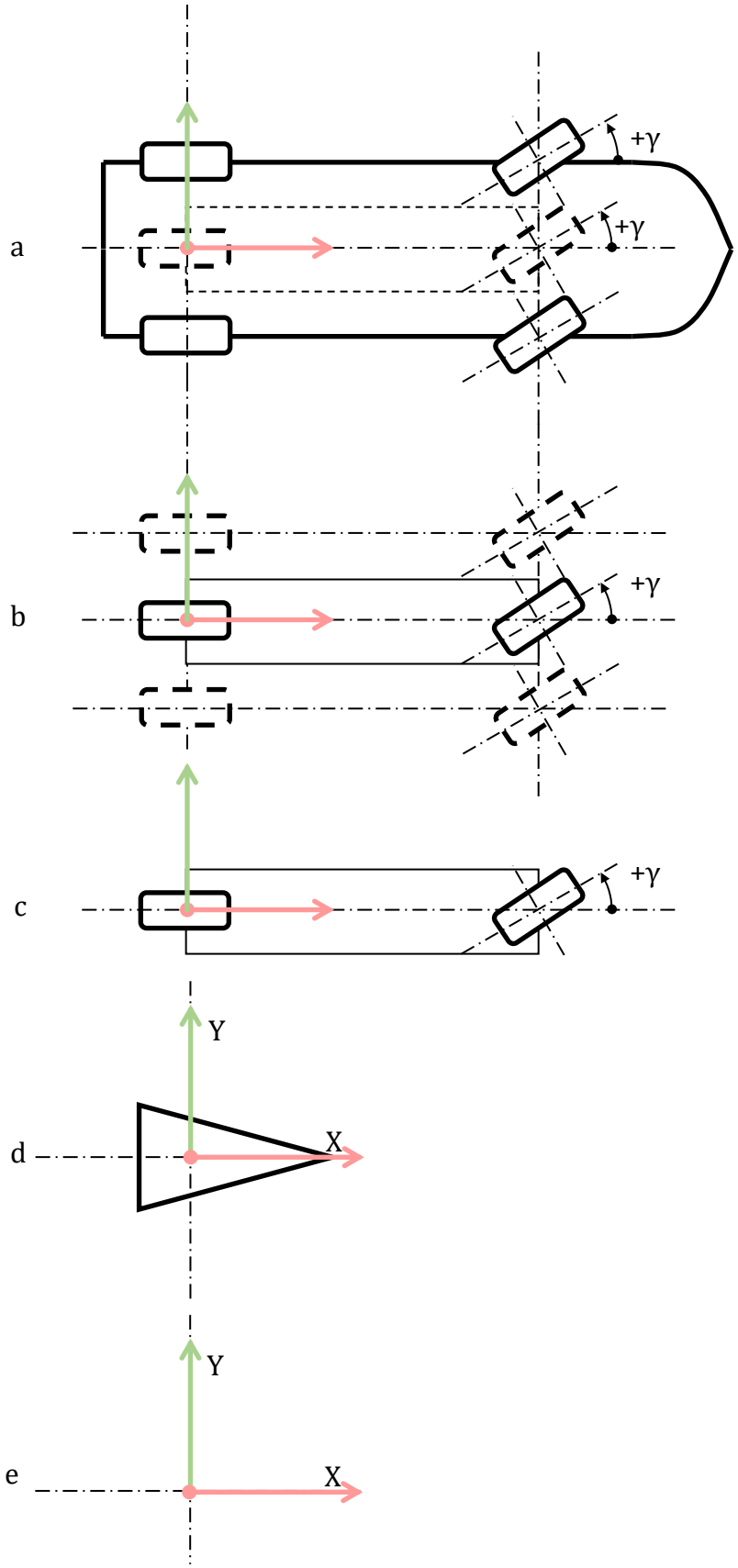


Figure 15 - Geometrical models used for the BM representation

The representations (geometrical models) are presented in Figure 15. To describe the car-like robot, five type of representation is selected; each one can be used, but this is a trade-of, because all of them have its own advantages and drawbacks. The first two representations are used for the kinematical model construction. They illustrate the reasons for transforming a four-wheeled vehicle in to a two-wheeled vehicle (the bicycle). The third representation is the canonical representation of the bicycle model. The last two representations can also be used; they have the advantage of simplicity.

Starting from the hypothesis that the BM performs a planar motion and we add an extra hypothesis of a unique rotation centre ICC i.e. the Instantaneous Centre of Curvature. From Figure 16 it can be seen that this hypothesis are the reasons of transforming a four-wheel vehicle in to a two wheels model.

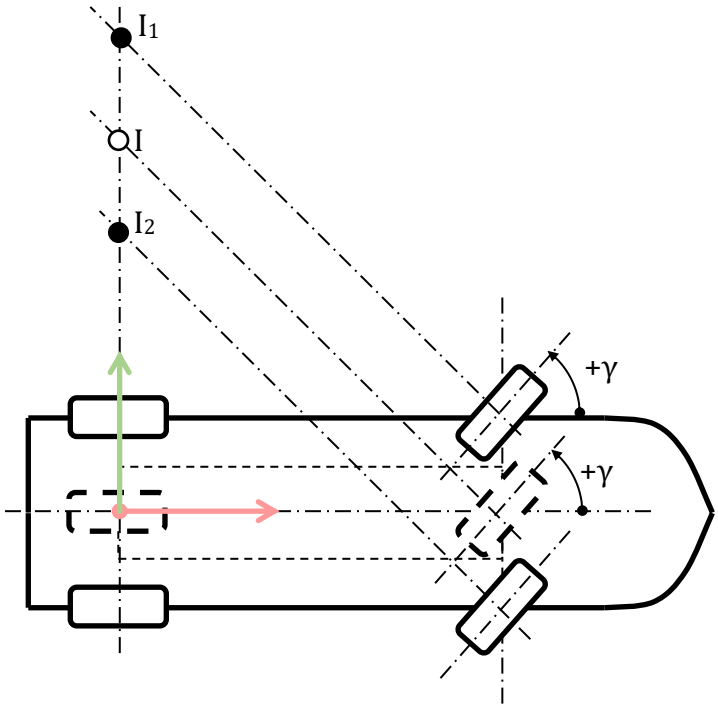


Figure 16 - The multiple rotation centres

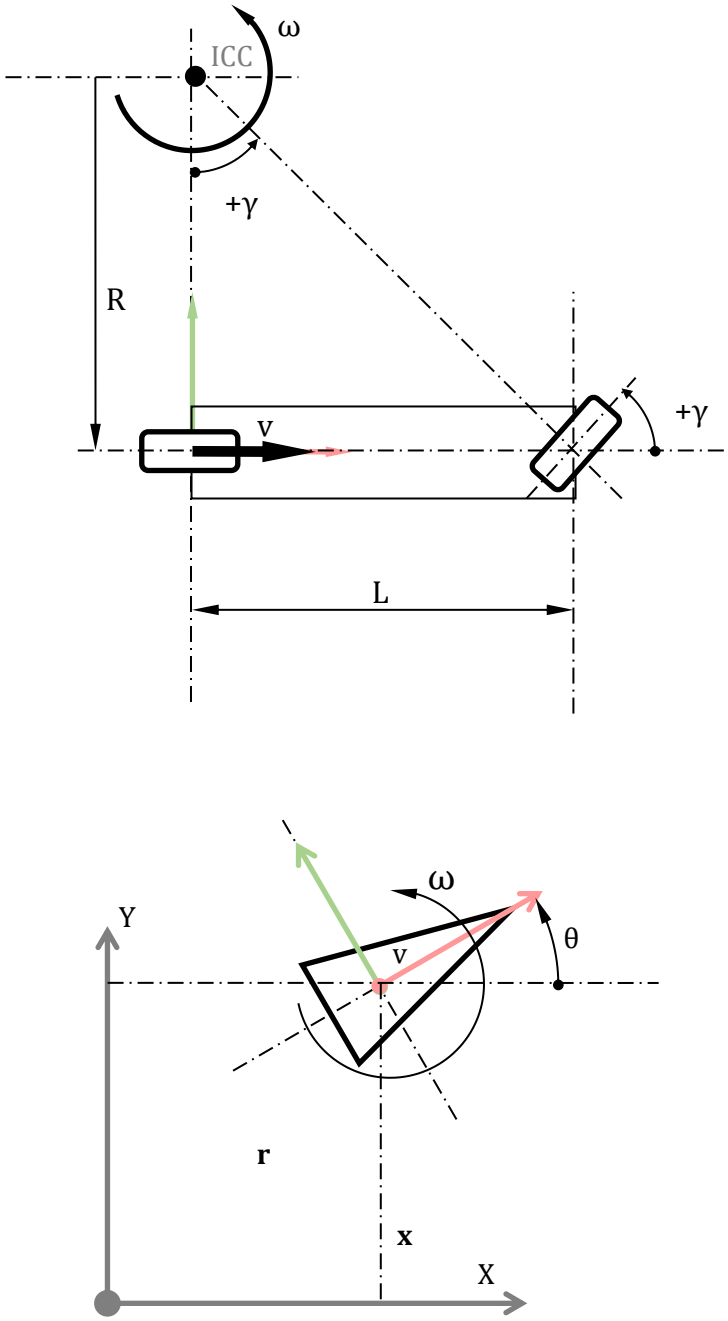


Figure 17 - The unique rotation centre

The kinematic model of the bicycle is a relation between the BM steering angle, the BM velocity and the BM position in the global referential frame.

$$\dot{q} \rightarrow \begin{cases} \dot{x} = v_1 \cdot \cos(\theta) \\ \dot{y} = v_1 \cdot \sin(\theta) \\ \dot{\theta} = \omega = v_1 \cdot \tan(\gamma)/L \end{cases} \quad (23)$$

Where

- x is the longitudinal component of the position vector;
- y is the lateral component of the position vector;
- v_1 is the (norm) of the velocity;
- γ is the robot orientation;
- ω is the robot angular speed;
- γ is the steering angle;
- L is the BM wheelbase.

The numerical solution - using Euler approximation - of direct kinematic problem can be formulated in equation (24) as follows.

$$\boldsymbol{x}(k+1) = \boldsymbol{x}(\mathrm{k}) + \begin{bmatrix} \delta_d \cdot c(\theta + \delta_\alpha) \\ \delta_d \cdot s(\theta + \delta_\alpha) \\ \delta_\alpha \end{bmatrix} \tag{24}$$

where:

$\boldsymbol{x}(\mathrm{k}) + \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix}$, $\begin{cases} \delta_d = v_k \cdot \Delta t \\ \delta_\alpha = \omega_k \cdot \Delta t = \frac{v(k)}{L} \tan(\gamma(k)) \cdot \Delta t \end{cases}$, where Δt is the sampling time

Now the lateral dynamical model of bicycle model is discussed. After the accelerations are computed which act on the robot (vehicle) these accelerations draw attention about the direction and the source of perturbations. If the vehicle trajectory has a curvature a normal acceleration in the direction (sense) of the curvature centre need to be created (by an appropriate force). In fact, this force does not exist so the vehicle will tend to slip in the opposite direction (the direction of the inertial force). The lateral dynamical model is a complex model which explains this phenomenon. More precisely, the vehicle target is a point which moves on a trajectory and lateral slippages are happening. The construction of this model involves three steps. First, compute the inertial force (d'Alembert) which act on the robot and obtain the lateral dynamic model (the BM dynamic model). Second, compute the differential model of a mobile point targeting. And finally, combine the previous models.

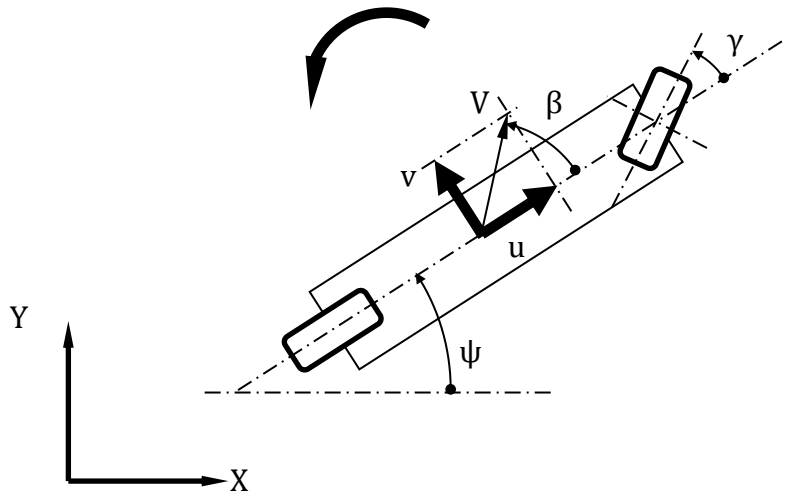


Figure 18 - Computing the accelerations

The BM dynamical model, in the local (B) referential frame we have:

$$\boldsymbol{V} = \begin{bmatrix} u \\ v \\ 0 \end{bmatrix} \tag{25}$$

$$\dot{\boldsymbol{V}} = \frac{\partial \boldsymbol{V}}{\partial t} + \dot{\Psi} \times \begin{bmatrix} u \\ v \\ 0 \end{bmatrix} = \begin{bmatrix} \dot{u} - \dot{\Psi} \\ \dot{v} + \dot{\Psi} \\ 0 \end{bmatrix} \tag{26}$$

where: u, v are the velocity components , visible on Figure 18 and $\dot{\Psi} = \begin{bmatrix} 0 \\ 0 \\ \dot{\Psi} \end{bmatrix}$ is the angular speed. This leads Newton-Euler equation formulation:

$$\begin{cases} m(\dot{u} - \dot{\Psi} v) = \sum F_u \\ m(\dot{v} - \dot{\Psi} u) = \sum F_v \\ J_z \ddot{\Psi} = \sum M_z \end{cases} \tag{27}$$

where: m, J_z are the vehicle mass and the vehicle inertia momentum on z direction; in the right side of the equations are the sum of external forces and torques which acts on the vehicle, which is visible on Figure 19.

In Figure 19 we have illustrated the lateral force which acts on the vehicle, we have neglected the longitudinal forces. The speed is considered in the mass centre of the BM.

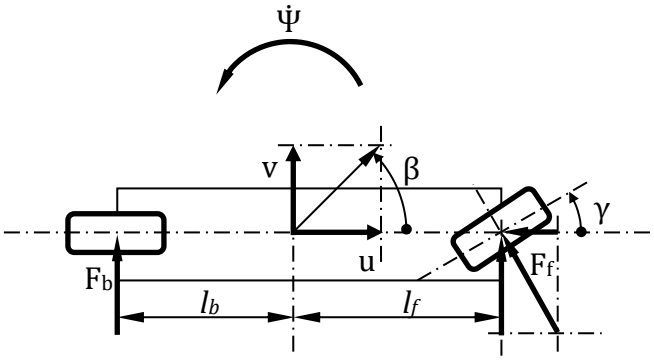


Figure 19 - The lateral external forces

$$\begin{cases} m(\dot{u} - \dot{\Psi}v) = -F_f \sin(\gamma) \\ m(\dot{v} - \dot{\Psi}u) = F_b + F_f \cos(\gamma) \\ J_z\ddot{\Psi} = l_f \cdot F_f \cos(\gamma) - l_b \cdot F_b \end{cases} \quad (28)$$

Several approximations have been made, for small angle γ we can obtain:

$$\begin{cases} m(\dot{u} - \dot{\Psi}v) = -F_f \gamma \\ m(\dot{v} - \dot{\Psi}u) = F_b + F_f \\ J_z\ddot{\Psi} = l_f \cdot F_f - l_b \cdot F_b \end{cases} \quad (29)$$

We will consider the hypothesis that V is constant (the dynamic of the slippages is quicker than the vehicle acceleration) and because β can be also considered small we obtain:

$$u = V\cos(\beta) \approx V \Rightarrow \dot{u} = 0$$

$$v = V\sin(\beta) \approx V\beta \Rightarrow \dot{v} = V\dot{\beta} \quad (30)$$

$$\begin{cases} m\dot{\Psi}V\beta = F_f \gamma \\ m(V\dot{\beta} - \dot{\Psi}V) = F_b + F_f \\ J_z\ddot{\Psi} = l_f \cdot F_f - l_b \cdot F_b \end{cases} \quad (31)$$

The lateral forces are elastic forces generated due the rotation of the wells relative to the ground.

$$F_f = -\alpha_f c$$

$$F_b = -\alpha_b c \quad (32)$$

where: $\alpha_{f,b}$ are the twisting angles of the front and back wheel; c is elastic coefficient, we have considered $c_f = c_b = c$.

Figure 20 illustrates the angles computation. The twisting angles are generated by the vehicle rotation. We have computed them in the hypothesis of small angle where the tangent value can be approximated by its inputs (variable).

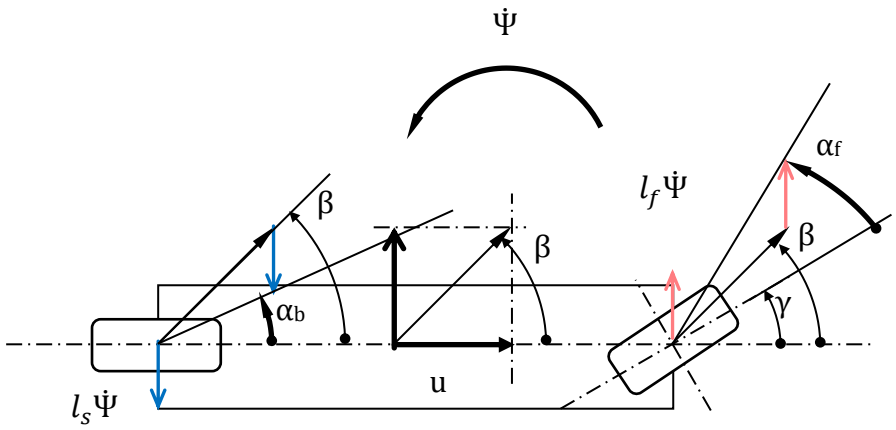


Figure 20 - Twisting angles computation

$$\alpha_b = \beta - \frac{l_s \dot{\Psi}}{u} = \beta - \frac{l_s \dot{\Psi}}{V}$$
$$\alpha_f = \beta + \frac{l_f \dot{\Psi}}{u} - \gamma = \beta + \frac{l_f \dot{\Psi}}{V} - \gamma \quad (33)$$

If we subtract these equations, we obtain

$$\alpha_b - \alpha_f = \gamma - \frac{l\,\dot{\Psi}}{V}$$
$$\gamma = \alpha_b - \alpha_f + \frac{l}{R} \quad (34)$$

where: $l = l_b + l_f$

For steady state conditions, equation (31) simplifies to:

$$\begin{cases} m\left(\frac{V^2}{R}\right) = F_b + F_f \\ 0 = l_f \cdot F_f - l_b \cdot F_b \end{cases} \quad (35)$$

The following can be obtained:

$$\gamma = \alpha_b - \alpha_f + \frac{l}{R}$$

$$\gamma = \frac{F_f}{c} - \frac{F_b}{c} + \frac{l}{R} = m\frac{l_b - l_f}{lc} \cdot \frac{V^2}{R} + \frac{l}{R}$$

From kinematic model:

$\gamma_{kin} = \frac{l}{R}$, so:

$$\gamma = m\frac{l_b - l_f}{lc} \cdot \frac{V^2}{R} + \gamma_{kin} \tag{36}$$

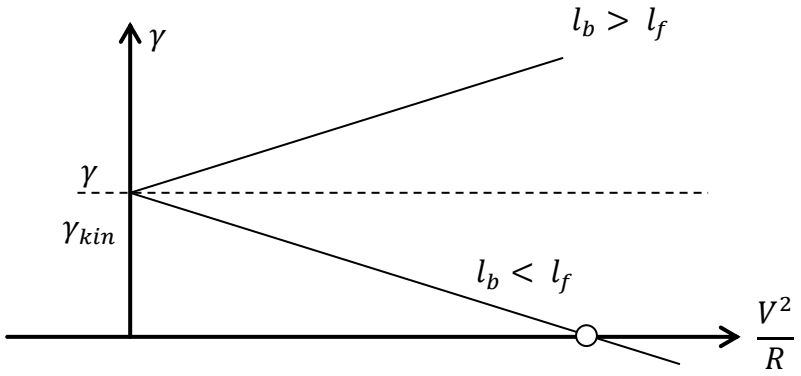


Figure 21 - Understeer and oversteer phenomena

Equation (36) is illustrated on Figure 21 where two possible behaviour is observable: if $l_b > l_f$ the vehicle becomes under steer and if $l_b < l_f$ the vehicle becomes over steer. The understeer phenomenon can be described in the following way: suppose that the vehicle turns with a constant radius R. Using kinematic hypothesis, the vehicle's steering angle γ is less than the amount intended by the driver. By contrary in the lateral dynamic hypothesis for an oversteer vehicle the steering angle must be increased if the speed increases (for the same steering radius). If the steering angle will not increase the radius will increase. The oversteer phenomenon (invers), the state space form of the dynamic model can be obtained from equation (31) to equation (34):

$$\begin{bmatrix} \dot{\beta} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} -\frac{2c}{mV} & \frac{c(l_b - l_f)}{mV^2} - 1 \\ \frac{c(l_b - l_f)}{J_z} & \frac{c(l_b^2 + l_f^2)}{J_z V} \end{bmatrix} \begin{bmatrix} \beta \\ \dot{\Psi} \end{bmatrix} + \begin{bmatrix} \frac{c}{mV} \\ \frac{cl_f}{J_z} \end{bmatrix} \gamma \tag{37}$$

where:

- β is the angle between the velocity V and the vehicle direction;
- δ is the steering angle;
- ψ is the vehicle direction angle;
- m, J_Z are the mass and the momentum of the vehicle;
- V is the vehicle velocity which is considered constant;
- c is the rotational stiffens of the wheels;
- l_f and l_b are the length from the mass centre of the front and back wheels.

For simplicity we will rewrite this equation:

$$\begin{bmatrix}\dot{\beta}(t)\\ \dot{\Psi}(t)\end{bmatrix} = \begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix}\begin{bmatrix}\beta\\ \dot{\Psi}\end{bmatrix} + \begin{bmatrix}b_1\\ b_2\end{bmatrix}\gamma \tag{38}$$

For the numerical solution of equation (38) the following equation can be concluded:

$$\begin{bmatrix}\dot{\beta}(k+1)\\ \dot{\Psi}(k+1)\end{bmatrix} = \begin{bmatrix}1-\Delta t\frac{2c}{mV} & \Delta t\frac{c(l_s-l_f)}{mV^2}-1\\ \Delta t\frac{c(l_s-l_f)}{J_z} & 1-\Delta t\frac{c(l_b^2+l_f^2)}{J_zV}\end{bmatrix}\begin{bmatrix}\beta(k)\\ \dot{\Psi}(k)\end{bmatrix} + \Delta t\begin{bmatrix}\frac{c}{mV}\\ \frac{cl_f}{J_z}\end{bmatrix}\gamma(k) \tag{39}$$

After that it can be calculated how to target a mobile point, with a differential model. The phenomenon of targeting the P point is illustrated in Figure 22. The BM (represented by its simplest geometrical model) targets a point P (its desired position). The distance from the actual pose of the BM and the desired pose is $[x, \theta\Delta]$. This distance depends on s and r which are the natural coordinates of the two curves i.e. the actual trajectory and the desired trajectory.

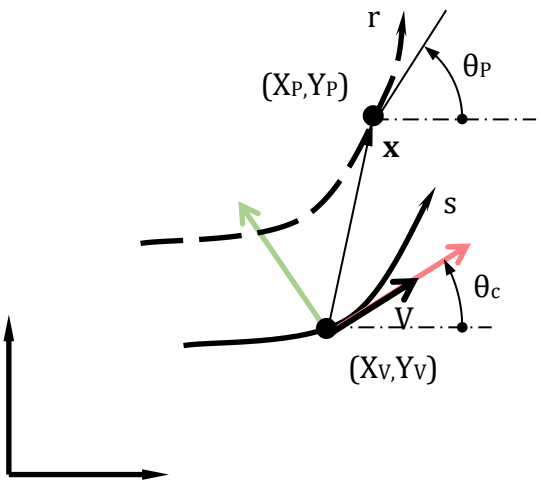


Figure 22 - Targeting a point *P*

The first step is to compute the variations of the pose (40) in the vehicle referential system:

$$\begin{bmatrix} \mathbf{x} \\ \boldsymbol{\theta}_{\Delta} \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta_{\Delta} \end{bmatrix} \tag{40}$$

Considering the Frenet relations and the coordinate transformations it can be obtained:

$$\begin{cases} dx(r,s) = \cos\theta_{\Delta}(r,s)dr + (y(r,s)\kappa_v(s) - 1)\, ds \\ dy(r,s) = \sin\theta_{\Delta}(r,s)dr - \big(y(r,s)\kappa_v(s)\big)\, ds \\ d\theta_{\Delta}(r,s) = \kappa_p(r,s)dr - \kappa_v(r,s)ds \end{cases} \tag{41}$$

where:

- $\theta\Delta = \theta P(r) - \theta V(s)$;
- κ_p is the curvature in point P
- κ_v is the curvature in point V

With the visual help of Figure 18 and Figure 22 the following equations can be written:

$$a_c = \Psi + \beta \tag{42}$$

$$\begin{cases} dx(r,s) = \cos\theta_\Delta(r,s)dr + \left(x(r,s)\frac{\partial\Psi(s)}{\partial s} - cos\beta(s)\right) ds \\ dy(r,s) = \sin\theta_\Delta(r,s)dr - \left(y(r,s)\frac{\partial\Psi(s)}{\partial s} - sin\beta(s)\right) ds \\ d\theta_\Delta(r,s) = \kappa_p(r,s)dr - \frac{\partial\Psi(s)}{\partial s} ds \end{cases} \quad (43)$$

The following hypothesis is accepted: the slippage is perpendicular to the speed direction, see Figure 23:

$$x(r,s) = 0; dx(r,s) = 0 \quad (44)$$

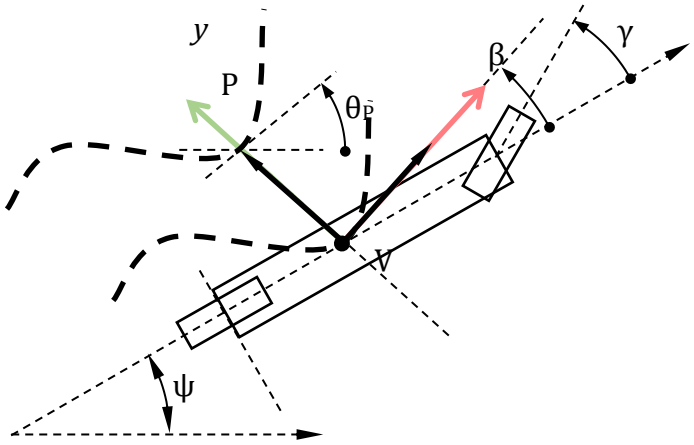


Figure 23 - The accepted slippages model

Using the hypothesis and after linearization (43) becomes:

$$\dot{y} = V(\theta_\Delta + \beta)$$
$$\dot{\theta_\Delta} = V\kappa_p - \dot{\Psi} \quad (45)$$

where: $\theta\Delta = \theta P(r) - \psi(s)$;

If we add the BM model and the trajectory model, it can be obtained:

$$\begin{bmatrix} \dot{\beta} \\ \ddot{\Psi} \\ \dot{\theta_\Delta} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ 0 & -1 & 0 & 0 \\ V & 0 & V & 0 \end{bmatrix} \cdot \begin{bmatrix} \beta \\ \dot{\Psi} \\ \theta_\Delta \\ y \end{bmatrix} + \begin{bmatrix} b_1 & 0 \\ b_2 & 0 \\ 0 & V \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \gamma \\ \kappa_p \end{bmatrix} \quad (46)$$

After a small rewriting of the previous equation:

$$\begin{bmatrix} \dot{\beta} \\ \ddot{\Psi} \\ \dot{\theta_\Delta} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ 0 & -1 & 0 & 0 \\ V & 0 & V & 0 \end{bmatrix} \cdot \begin{bmatrix} \beta \\ \dot{\Psi} \\ \theta_\Delta \\ y \end{bmatrix} \cdot \gamma + \begin{bmatrix} b_1 & 0 \\ b_2 & 0 \\ 0 & V \\ 0 & 0 \end{bmatrix} \cdot \kappa_p \quad (47)$$

Equation (47) considers γ the control signal and κ_p the perturbation and y the output, so then the BM lateral dynamic model becomes:

$$\begin{bmatrix} \dot{\beta} \\ \dot{\Psi} \\ \dot{\theta_\Delta} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ 0 & -1 & 0 & 0 \\ V & 0 & V & 0 \end{bmatrix} \cdot \begin{bmatrix} \beta \\ \Psi \\ \theta_\Delta \\ y \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ 0 \\ 0 \end{bmatrix} \cdot \gamma$$

$$y = [0 \quad 0 \quad 0 \quad 1] \cdot [\beta \quad \Psi \quad \theta_\Delta \quad y]^T \tag{48}$$

For the sake of simplicity, the equation can be reformulated as follows.

$$\begin{bmatrix} \dot{\beta} \\ \dot{\Psi} \\ \dot{\theta_\Delta} \\ \dot{y} \end{bmatrix} = \tilde{A} \cdot \begin{bmatrix} \beta \\ \Psi \\ \theta_\Delta \\ y \end{bmatrix} + \tilde{B} \cdot \delta$$

$$y = \tilde{C} \cdot [\beta \quad \Psi \quad \theta_\Delta \quad y]^T + \tilde{D} \cdot \delta \tag{49}$$

As a result of this the robot (vehicle) model and the trajectory model is formulated properly. To summarize it, whether, the case is with car-like or differential robot, it is important not only about the robot's ability to reach the required final pose, but also about how to get there. In this section the necessary kinematics models were introduced which are essential to understand the thesis.

2.3.3 Boustrophedon cellular decomposition coverage (BCDC)

The Boustrophedon Cellular Decomposition Coverage (BCDC) was published by Howie Choset and Philippe Pignon in 1997, and later became a popular [18] coverage process. The word itself (boustrophedon) comes from a Greek word, which literally means "the turning of the ox" [19]. The algorithm is based on the observation that the oxen walk through the clearing while grazing. In practice, this means moving up and down in a square area. To illustrate this with a real-life example, this is how lot of the people cut the grass in the garden.

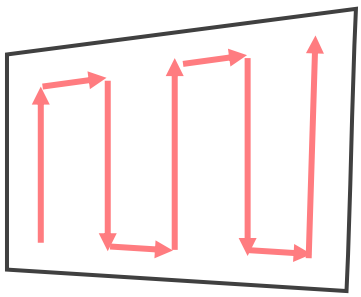


Figure 24 - A simple Boustrophedon movement

The original algorithm assumes a line map (polygon maps), but of course the principle works in the same way, for example, in the case of occupancy grid-based map [20]. The basic idea of the BCDC is to expand (decompose) the area into areas that can be easily covered with up and down motions, and then connect the resulting parts (cells). Thus, it can be seen that the whole area is finally covered. Cell splitting, or decomposition can be done using the less efficient trapezoidal division [21] or exact cellular decomposition [13] method. As described in the introduction of this section, algorithms for the overlay problem may have algorithmic variations. In the following exact cells decomposition will be detailed and compared against the trapezoidal decomposition.

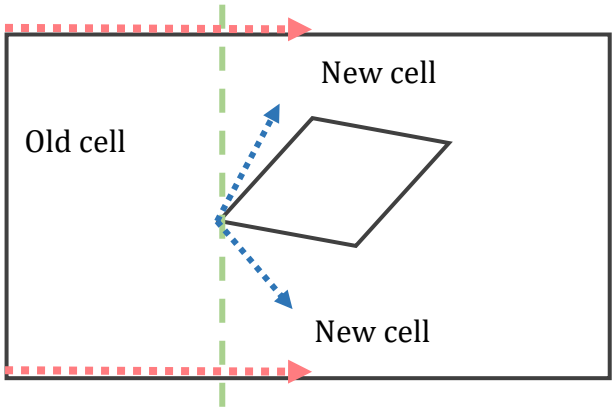


Figure 25 - Cell opening, a demonstration of the IN event

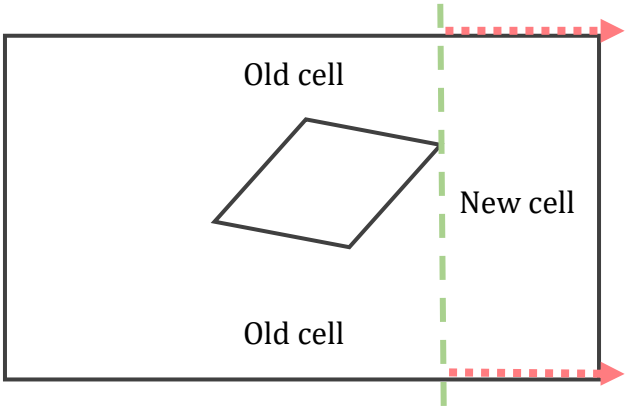


Figure 26 - Cell closing, a demonstration of the OUT event

The exact cells are formed through a series of opening and closing operations that occur when an event from the left-to-right "slice" takes place. An event here is crossing the tip of a polygon as the slice moves. There are three types of events [19], IN, OUT and MIDDLE. In a loose sense, at an IN event, when moving from left to right, it meets a vertex of one (or more) new internal polygons passing vertically across the map. Then new cells need

to be created. The OUT event is just the opposite; it happens, when two (or possibly more) cells reach the end of the inner polygon, and these cells need to be closed, and a unified need to be opened. The IN event can be considered as a cell that decomposes into multiple cells while the OUT event occurs when multiple cells merge. During the MIDDLE event, the numbers of the cells do not change [19].

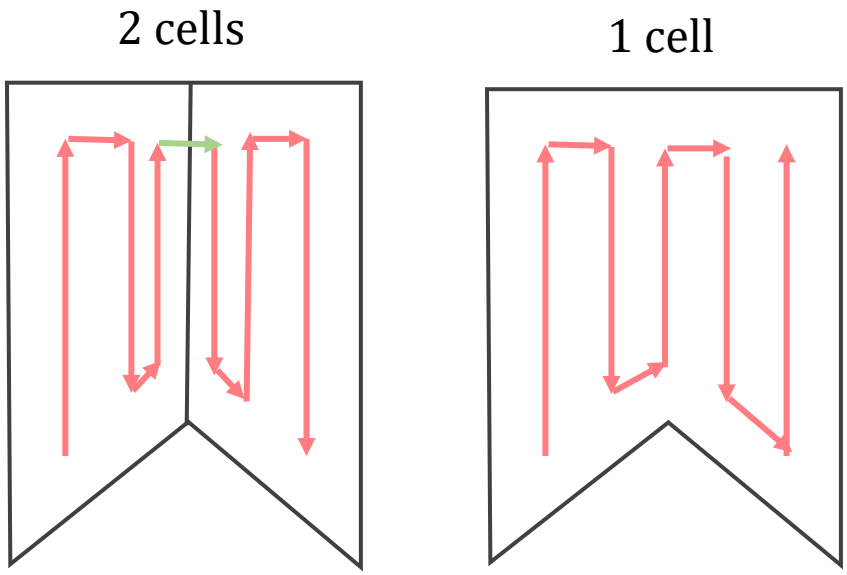


Figure 27 - Boustrophedon movement on 2 cells and 1 cell; the less cell, the shorter the path

On the other hand, trapezoidal decomposition generates trapezoids from the original polygon map (line map). This is quite a straightforward algorithm, the map is sliced with vertical lines at the tip of every segment, and the trapezoid cells are present. As the example on Figure 27 shows the trapezoidal decomposition generates in the best case the amount of cells as the exact decomposition, which is of course not desirable. It is no surprise that the simple trapezoidal decomposition performs worse, that a slightly more complex exact decomposition.

Figure 28 shows a trapezoidal decomposition example, whereas 5 cells were generated. In contrast Figure 29 shows the same map, but the decomposition method here is exact cellular decomposition. This method generates less cells, 4 to be precise. The cells of course can be covered with a simple boustrophedon motion, which is illustrated with the red arrows. After that, the connection between the covered cells needs to be determined, thus create a coherent path. This can be seen as a graph routing problem where the nodes of the graph are the cells. The smaller the graph the less computation is needed to create

the cell-connecting path, which is why the exact decomposition is preferred over trapezoid.

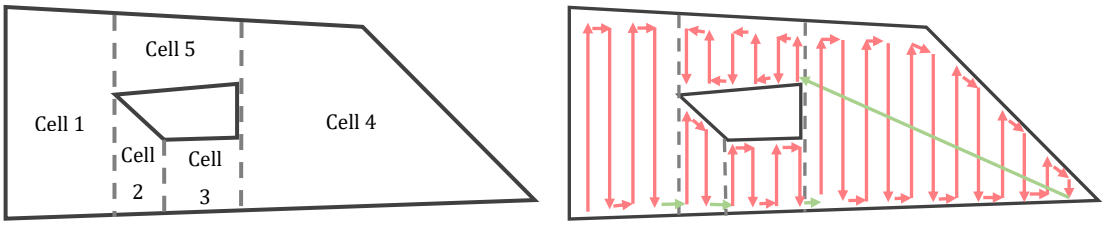


Figure 28 - Trapezoid decomposition example

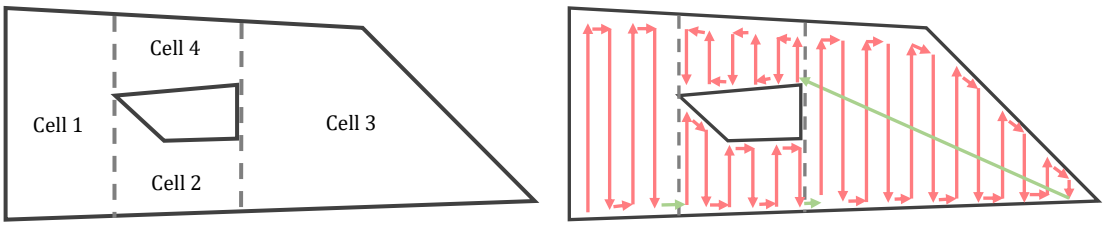


Figure 29 - Exact decomposition example

Until now the cellular decomposition was only examined on line maps or in other words polygon maps. For my thesis the occupancy grid map is the most significant. According to the best of my knowledge, prior to my implementation, there was no realization of the boustrophedon exact cellular decomposition to occupancy grid; at least not publicly available, although the realization is quite straightforward. I implemented it to be able compare with my own results.

2.3.4 Random Path Planning

Random path planning is the simplest possible coverage path planning algorithm. The basic idea is to start with a simple motion until it reaches an obstacle, and then change direction randomly [12].

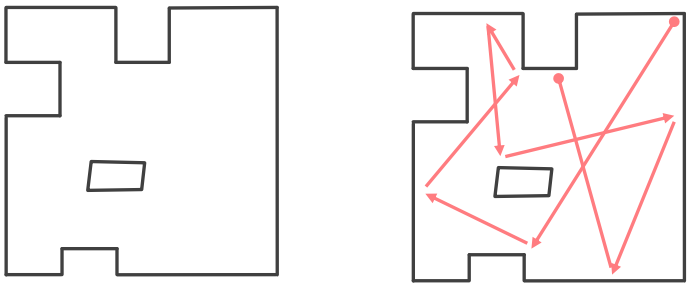


Figure 30 - Random path planning example

This approach is sometimes combined with pre-programmed movements: such spirals, serpentines or wall following mechanism, to increase efficiency. Despite the simple approach there are some advantages. For example, it does require a lot of computation power nor expensive sensor. Additionally, the robot does not need to use a map of its environment. These qualities made this kind of algorithms successful in the early robotic vacuum cleaners.

2.3.5 Backtracking Spiral Algorithm (BSA)

Backtracking Spiral Algorithm (BSA) covers the area starting from the outside and going towards to the centre [12]. In real life there are hardly any environments where a single spiral can cover the entire area. Thus, multiple spirals are needed, which necessitates a procedure like backtracking. This means going back to places where the robot has been before, but where it found a way to go to a still uncovered area. This process finally can lead to complete coverage. A drawback of the backtracking spiral algorithm is that it needs to store all the locations it has visited.

2.3.6 Neural Network Approach

The neural network approach of coverage [15] trains a neural network (NN), whose dynamic neural activity landscape represents the dynamically varying environment. This approach operates by properly defining the external inputs from the varying environment and the internal neural connections. The neural activities of the unclean areas and obstacles are guaranteed to stay at the peak and the valley of the activity landscape of the neural network, respectively. The unclean areas globally attract the robot in the entire state space through neural activity propagation, whereas the obstacles have only local effect to avoid collisions.

2.3.7 Internal Spiral Search (ISS)

The basic idea behind the Internal Spiral Search (ISS) algorithm is kind of similar to the backtracking spiral algorithm, the robot traverses the area in a certain direction, until free area is present, otherwise it turns right [14]. This results a spiral path, similarly to BSA, but difference is in case the robot reaches a state where it is surrounded with covered

area, to a so called dead state [14], ISS uses a well-known A* algorithm in order to find the optimal path to a state where it still can cover area.

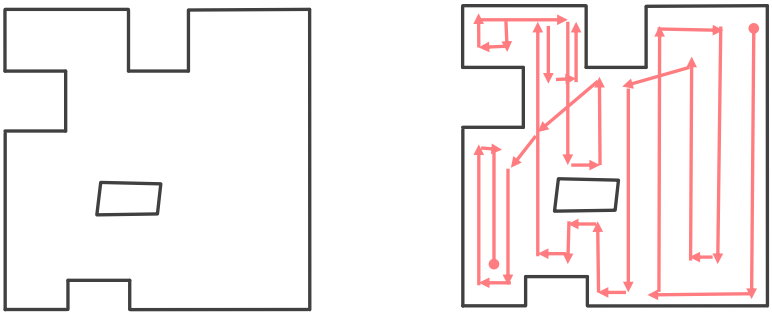


Figure 31 - Internal spiral search example

2.3.8 U-turn A* Path Planning (UAPP)

The concept of [14] relies on U-turn algorithm in order to search the complete coverage path. In a situation where it finds obstacles it takes the concept of 180^0 rotation. In other situation, where it cannot cover further areas with this 180^0 rotation, it uses A* to go to another region. Thus this kind of coverage can be viewed as an extension of the boustrophedon coverage. It is obvious that the UAPP algorithm is superior [14] to ISS algorithm in the environment with irregular obstacles.

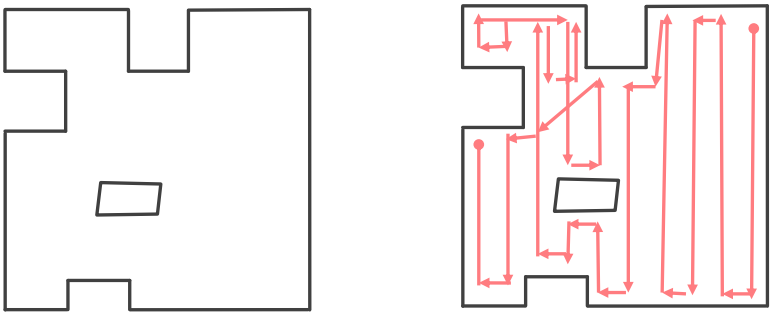


Figure 32 - U-turn A* path planning example

2.3.9 Online methods and SLAM

As mentioned in the introduction, the presented path planning methods are mostly offline, where the area is a priori known. This section deals with the online methods, where the robot or vehicle needs to discover the environment, and with SLAM where the robot simultaneously localizes itself creates the map. SLAM is the abbreviation Simultaneous Localization and Mapping and it belongs to the domain of navigation problems. The domain of the navigation contains many subdomains such as pathfinding, exploration, localization or map building, this is shown in a Venn diagram on Figure 33.

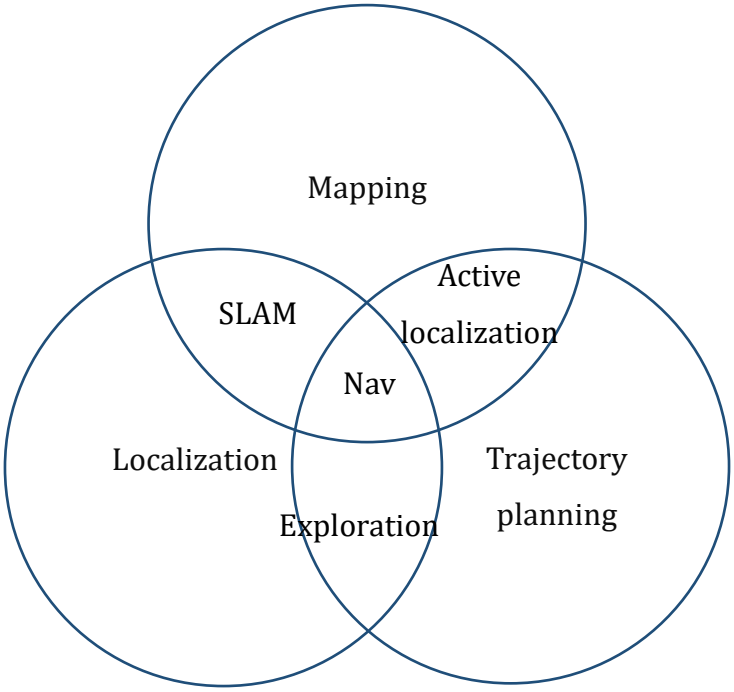


Figure 33 - The domain of the navigation (Nav) problem

Localization means establishing the robot position and orientation regarding the reference system, which can be one of the maps discussed further. *Mapping* means to create the model of the environment. *Trajectory planning* means planning the motion from a known start to a known goal pose while avoiding collisions over time. I would like to emphasize the time constraint; without the time parametrization it would be simply *path planning*. Regarding the thesis, the most significant aspect of path planning is coverage path planning, where the robot needs to cover the available area. Trajectory planning usually divides into *global* planning, where initial trajectory generated, and *local* planning which recalculates the path to with respect to the kinematic constraints and obstacles. These obstacles can be static or dynamic too, but a global planner only calculates with static objects. The local planner is sometimes referred as a controller. During *exploration*, the task of the robot is to use the sensory data to get around the environmental barriers and to move around the unknown environment as optimally as possible. Straightforwardly simultaneous localization and mapping, also known as SLAM means updating environmental model while simultaneously keeping track of the robot position and orientation respect to this model. Currently, GraphSLAM still used extensively in many robotics applications, e.g. in ROS as gmapping. It was first described in [22] as a Dynamic Bayes Problem and most implementations are fundamentally based on these assumptions. A very recent implementation is Google's Cartographer [23]. This

approach distributes the SLAM problem into local and global mapping with continuous pose optimization. During its lifecycle it stores submaps (defined by their poses) in the form of probability grids based on pose estimation and scan matching. SLAM algorithms also differ in sensor usage; there are SLAM methods based on visual monocular, stereo, RGB-D (visual + depth), and LIDAR (depth). Monocular-vision-based SLAM algorithms are usually divided into two groups: feature-based methods and direct methods. E.g. LSD-SLAM is a direct SLAM algorithm for monocular and stereo cameras. The sensor is tracked using direct image alignment and the geometry is approximated in the form of semi-dense depth maps. E.g. ORB-SLAM2 is a feature-based SLAM where the algorithm searches for certain features, key-points and only uses these features to estimate the location. RBG-D-based methods such as RTAB-Map usually do not work outdoors since they often use infrared-based sensors for depth-sensing, which interferes with direct sunlight.

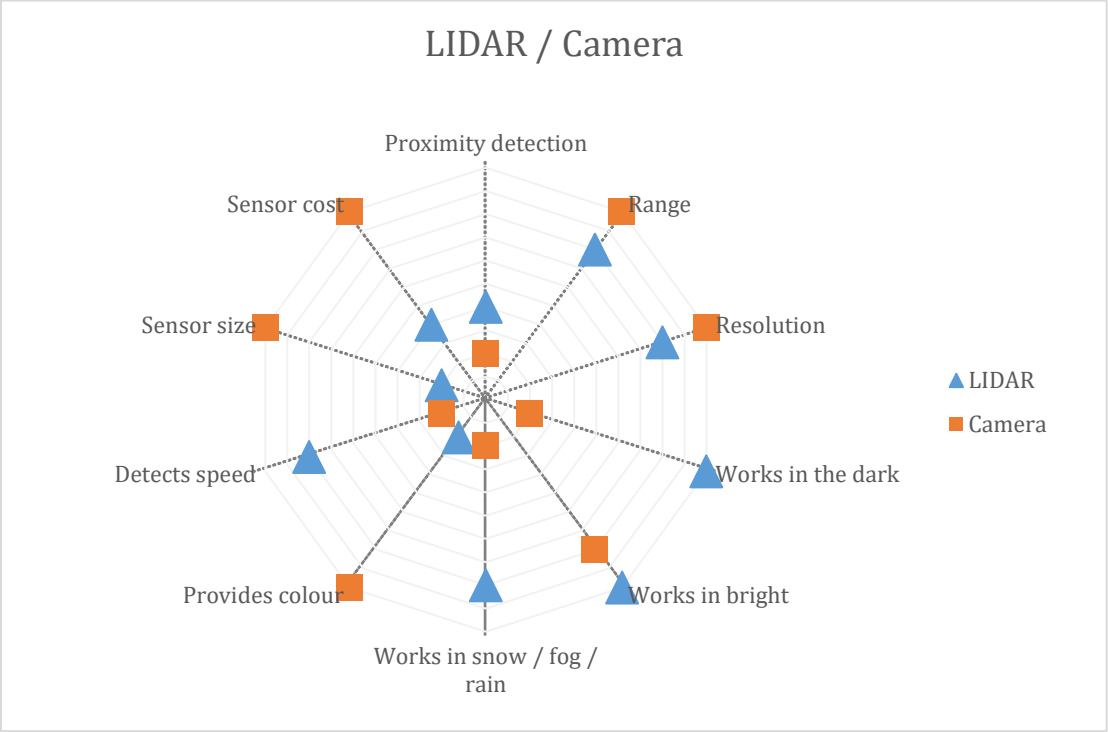


Figure 34 - Comparison of LIDAR and camera [24] [25]

At the time of writing there are two main approaches on which sensor to use as a main sensor for self-driving vehicles. Cameras are cheap and with neural networks may provide good object detection. On the other hand, LIDARs also work in bad lightning conditions and provide distance data in a for a much more expensive price.

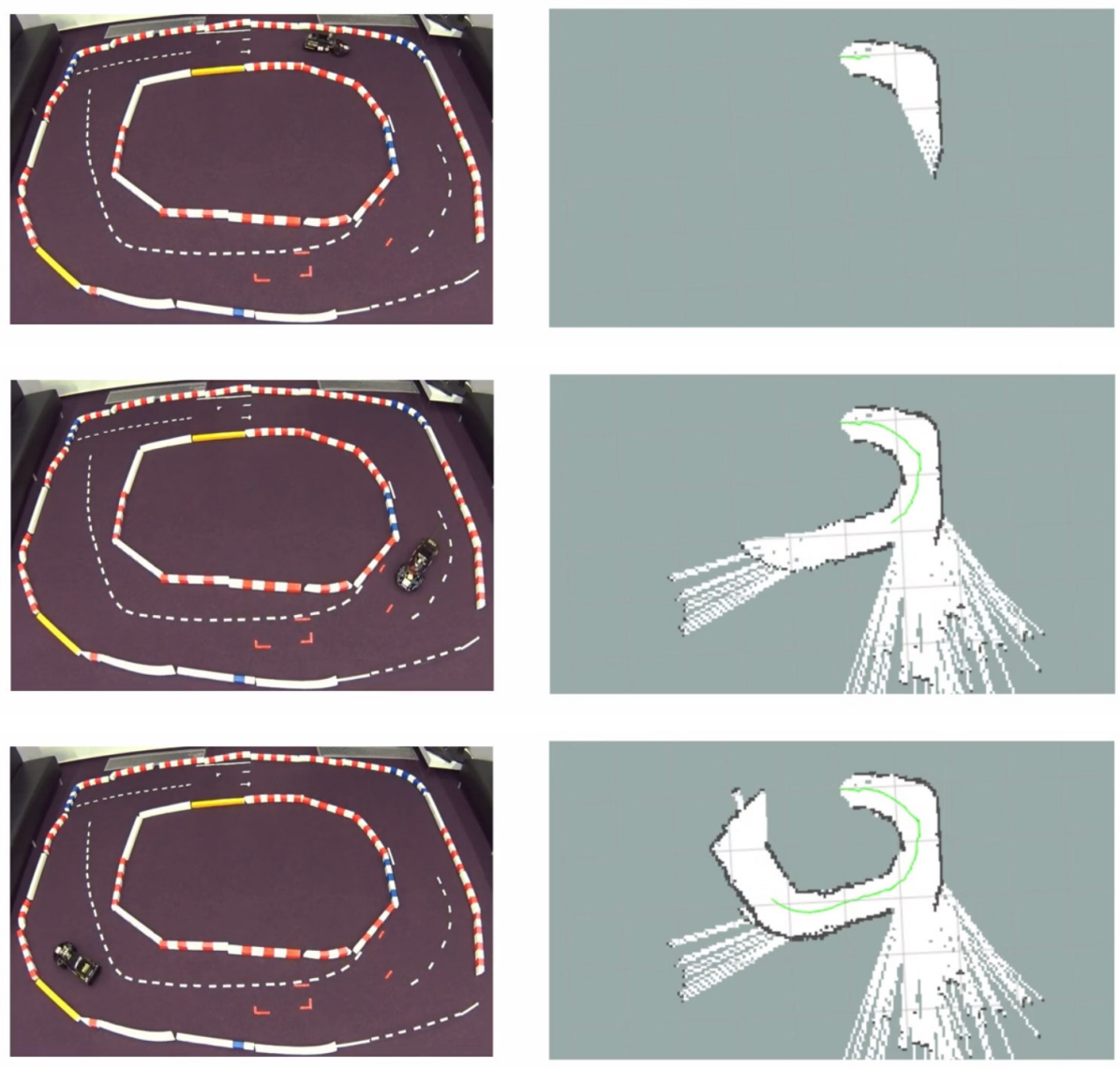


Figure 35 – How a real SLAM algorithm builds map and localizes itself (the measurement was done at the Research Centre for Vehicle Industry). On the model vehicle only a single LIDAR was involved with no additional sensor

Loop-closure is a fundamental problem regarding the SLAM algorithms. The authors of Google Cartographer formulate the loop-closure problem as an optimization problem based on submap and scan poses.

2.3.10 Trajectory and path following approaches

Trajectory following is the task of minimizing the distance to a given reference trajectory. One of the simplest imaginable trajectory-following approaches is called the follow-the-carrot algorithm. The follow-the-carrot algorithm is based on a rather modest idea. It obtains a goal-point, then aims the robot or vehicle towards that point. The name comes

from the old tales where is an imaginary rider controlled a donkey with a carrot. The carrot is held by the human rider so that it extends above and in front of the donkey's head, and the carrot hangs on a string from the far end of the stick. The donkey aims the carrot, but it is linked to the rider, and although it cannot reach it, at least it moves in that direction. The working principle is similar as the imagined case. The robot moves into the direction of the always moving goal point, so it only reaches it at the end of the trajectory. This is of course a simple approach and it suffers from a lot of weaknesses, like the extremely large deviations, but it can be at least a good starting point to understand these approaches.

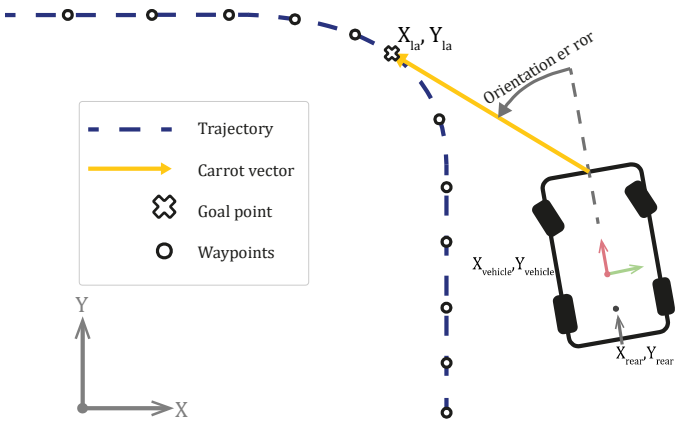


Figure 36 - High-level representation of follow-the-carrot algorithm

By comparison, the pure-pursuit approach calculates the *curve* that will take the vehicle from its current position to the selected goal position. The goal point is calculated similarly as for the follow-the-carrot algorithm. A circle (circular arc) is defined by two points: the current vehicle position and the currently selected goal point. After that, a vehicle receives a control reference according to the circular arc and a new goal-point is determined. Pure-pursuit algorithm has been widely used in robotics and automotive applications as a simple path / trajectory following solution.

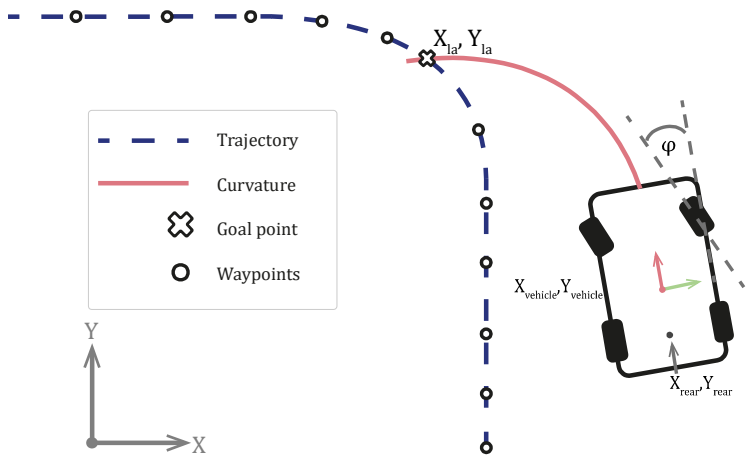


Figure 37 - High-level representation of pure-pursuit algorithm

The idea of the pure-pursuit can be realized in many and slightly different ways. For example, the classic pure-pursuit algorithm uses fixed look-ahead distance. This means that in the given track, a look-ahead point can be determined as the subset of waypoints, which is from a fixed distance from the vehicle. Choosing a look-ahead distance is a tradeoff between control overshoots and precision. If the look-ahead distance is small overshoot will appear because the chosen point is so close that the vehicle cannot react instantaneously, thus exceeds its target continuously. In this case the absolute distance (deviation or error) is smaller, but the vehicle will oscillate. In contrast, in case the chosen point is too far, the vehicle follows the point after the next corner, so it cuts off the curves. This behaviour can be eliminated if the look-ahead distance is adjustable or changeable. One of the most obvious ways to modify the look-ahead distance is to scale according vehicle speed [26]. The other method to look-ahead distance modification means involving a fuzzy controller that automatically tunes the look-ahead distance based on path characteristics, velocity, and tracking errors [27] [28] Correspondingly to the fuzzy-approach [29] it was shown that an autonomous vehicle can drive above 80 Km/h along explicit paths using differential GPS data. To improve the classic pure-pursuit algorithm and eliminate steering latency CF-pursuit [28] replaced the circles employed in pure-pursuit with a clothoid curve to reduce fitting errors.

Although many papers are available in this domain, the available source code is rather rare. A few of the most notable are the Autoware version [30], the Python version [31], and the Raptor UGV version [32]. My research also includes a pure-pursuit method based

on multiple look-ahead points [33]. Pure-pursuit suffers from weaknesses highly attributed to improper selection of look-ahead distance, resulting in poor tracking performance. Recent developments (including mine) focused on overcoming this drawback via enhancements of the original algorithm. The original pure-pursuit algorithm has been realized in many ways; a particular example is present in TierIV Autoware [30]. Widely used implementations are slightly different, but most of the understandings works [34] [30] [35] [36] as follows (given a vehicle with wheelbase length l):

1. Determine the current location of the vehicle in the global coordinate system ($X_{vehicle}, Y_{vehicle}$) together with the rear wheel (X_{rear}, Y_{rear});
2. Find a goal point (X_{la}, Y_{la}) with a (constant) look-ahead distance, search only in the next waypoint array;
3. Transform the goal point ($X_{\{la\}}, Y_{\{la\}}$) to rear wheel coordinates (X_{rear}, Y_{rear});
4. Calculate the steering angle and derive the curve from it:
    - $\phi = \arctan\left(2l \sin\left(\arctan \frac{Y_{la}-Y_{rear}}{X_{la}-X_{rear}}\right)\right)$
    - where φ is the steering angle and l is the length of the vehicle
5. Derive a control reference - for vehicles it is ϕ_{t+1} and v_{t+1}- from the calculated curve.
6. Update estimated state of the vehicle; update the next closest waypoint array.

The original pure-pursuit algorithm calculates curve with fixed look-ahead distance. In this case in the given track, a goal point can be determined as the subset of waypoints which is from a fixed distance from the vehicle. This attribute of the original algorithm introduces some weaknesses in terms of performance.

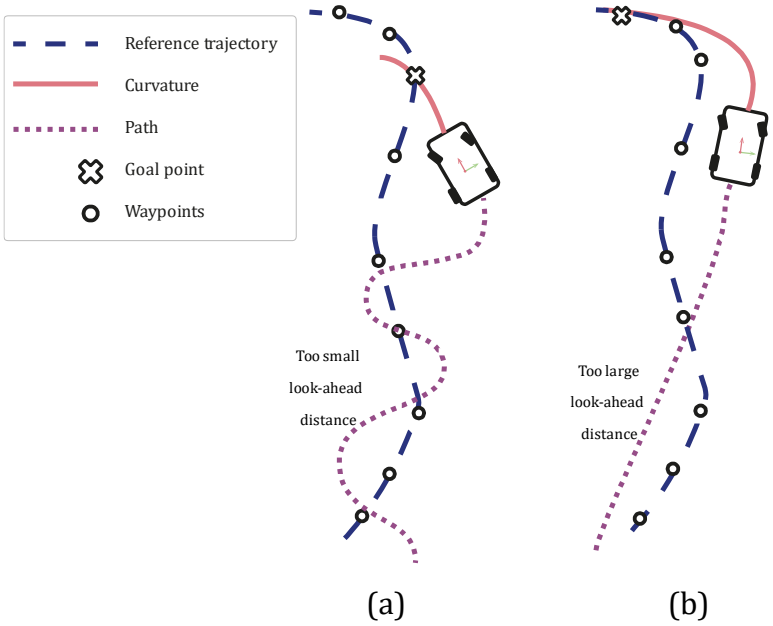


Figure 38 - How different values of look-ahead ratio effects the trajectory:
a) depicts small value, leading to overshoot
b) depicts large value, leading to undershoot

For example, if the look-ahead distance is chosen relatively small, the control overshoots the desired trajectory and the vehicle introduces an oscillating behaviour. In this case, the vehicle cannot move fast enough to properly follow the trajectory, so some fast overshoot will appear which results in the mentioned oscillating behaviour. On the other hand, when the look-ahead distance is chosen too large, the vehicle won't follow the trajectory precisely. The reason for this phenomenon comes from the fact that the vehicle goal is often after the curved trajectory which means that the manoeuvre considers a far goal point thus the original trajectory is followed with truncated curves. To get rid of the drawbacks of the fixed look-ahead distance phenomenon, some improvements to the pure-pursuit algorithm were proposed. One of them relies on a fuzzy controller that automatically tunes the look-ahead distance based on path characteristics, velocity, and tracking errors [36] [27]. Similarly to this approach [29] modified pure-pursuit algorithm was able driving autonomous vehicles at high speed above 80 km/h along explicit paths using differential GPS data. One of the other solutions to alter the look-ahead distance is to scale it with the vehicle speed [34]. The other problem with the pure-pursuit algorithm is that the curve does not completely match the vehicle's path. This comes from the phenomenon that steering has a latency. For example, CF-pursuit [36] replaced the circles employed in pure-pursuit with a clothoid curve to reduce fitting errors. The next logical

step would be to substitute the kinematic model to a dynamic one, but in the examined urban speed ranges, this does not worth the extra effort. As an additional argument to the kinematic model is that studies show [37] that they can perform similarly well compared to the dynamic model at low speeds.

2.4. Neural networks

In engineering (artificial) neural networks (ANN or simply NN) means an ordered collection of connected computing units, which are inspired by biological processes. These computing units are called neurons which are the nodes in the network and they roughly model the biological neurons from functional point of view. The basis of the methodology is inspired by the biological processes of the brain, where clusters of neurons are linked by biological axons. Neural networks are not algorithms or pre-programmed solutions in the classical interpretation, rather frameworks which are designed to perform task according to the (training) data. The advantage of this approach counter to the pre-programmed solutions is spotting patterns and learning from them. The methodology can be successfully applied in many fields from classification (e.g. voice and image recognition) to regression (e.g. forecasts). One of the drawbacks is that the reliability of the produced neural networks is difficult to measure considering the current state of the art.

Neural networks are part of soft computing techniques. The methods that are used here tend to be computationally intensive tasks with non-exact (approximate) solutions. Such computational tasks may include complex or NP-hard (non-deterministic polynomial) tasks for which there is no known algorithm that can calculate an exact solution in polynomial time. Soft computing topics include evolutionary algorithms, machine learning algorithms - including the neural networks -, metaheuristic algorithms, swarm intelligence algorithms and so on.

2.4.1 Biological outlook

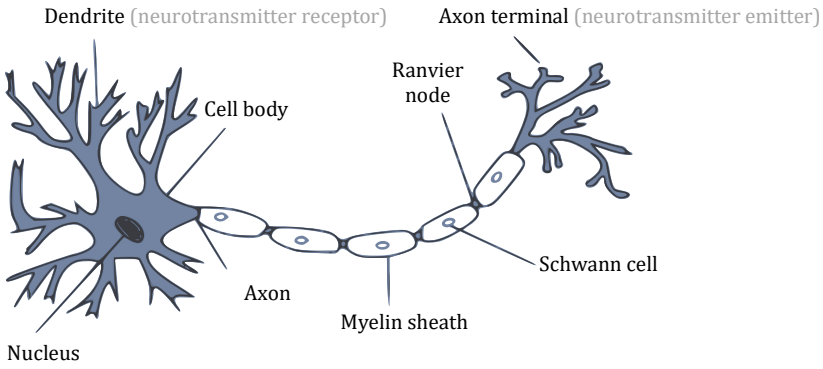


Figure 39 - Illustration of the biological neuron [38]

As mentioned in the previous section, the concept of an artificial neural network was inspired by the nervous system of the brain. It needs to be emphasized that this is only an inspiration, not an exact model from the nervous system; rather a simplified approximate, simplified conceptual and operational outline. On the other hand, the relationship between biology and the concept, are undeniable. The dendrite of the biological neuron is equivalent the input of the artificial neuron. Likewise, the cell body performs a function similar to the summary. Additionally, the axon corresponds to the activation function [38] [39] [40]. Another significant dissimilarity compared to today's cutting-edge computers is that biological brain overperforms approximately 10,000 times the computers in respect of computational performance [41]. For example about 86 billion neurons are present in the human nervous system, which are connected with approximately between 10^{14} - 10^{15} synapses [41].

2.4.2 Historical outlook

The idea and the application of neural networks [38] [40] dates back to the 1940s. In 1943, Warren McCulloch and Walter Pitts developed the model to demonstrate that these networks are widely applicable as logical and arithmetic functions as a convergent or substitute function. In 1959, Bernard Widrow and Stanford Marcian Hoff developed the models called "ADALINE" and "MADALINE" [38]. The first name comes from ADAptive LINear Elements. ADALINE has been developed to recognize binary patterns, and it had a weight, a bias and a summary function. These components later became de-facto

components of the modern neural networks. MADALINE was the first neural network used in a real world problem. Its purpose was to eliminate echoes on telephone lines by using an adaptive filter. From 1966, János Neumann has also been involved with the field of neural networks, proposing to imitate neural functions using telegraphy relays or vacuum tubes [38]. In the '60s and '70s a number of smaller or larger achievements were born, but the upsurge of neural networks was actually in the end of the '80s [38] [40]. At that time John Hopfield was an active researcher, who personally convinced many researchers about the importance of neural networks, along with Rumelhart, Hinton and Williams who established the concept of backpropagation which greatly accelerated the learning of networks. Soon after that LeCun combined the convolutional neural networks with backpropagation and achieved very good results in recognizing handwritten numbers [38] [42] [43] [44]. After the '80s some downturn occurred which continued till the second decade of the 21st century. This era which ended quite close to our recent days is sometimes referred as the second winter of neural networks. After the downturn in the 2010s many unquestionably successful solutions were presented, such as the Apple Voice Recognition System (Siri) [45], the VGG-VD (Visual Geometry Group, University of Oxford), which overperformed every traditional image recognition algorithm [46], AlexNet [47], or GoogLeNet [48].

Looking back on the broad history of NN, especially in terms of falling times (first and second winter), a justifiable question could be asked: will the current rising age similarly vanish as it has happened in the history multiple times? To answer this simple question, multiple factors need to be examined.

Before the current emerging NN era there was no highly ordered and easily reachable datasets as nowadays. This is important to highlight, because unlike human learning which requires low number of samples to learn, NN needs massive datasets. The other factor which can jumpstart the rising of the technology is the increased computational performance. Here I would underline the role of the nowadays available relatively cheap graphical processing units (GPUs). This is important because NN has a lot of parallelizable task which are suitable for the many cores of a GPU. Another factor which escalates the process is the theoretical improvement which relies on the further development of formerly successful methodologies e.g. backpropagation. Besides this there are easily accessible source code management applications (e.g. GitHub and others) and the developer friendly frameworks / software libraries / APIs / SDKs (e.g. TensorFlow, Caffe,

Theano, Keras, etc.) without which we would not be able to observe the increasing infiltration of the methodology today. Until the recent years working with neural network required significant C++ and CUDA know-how, which only a few people possessed. It has changed and now elementary Python scripting skills are enough for these kinds of researches. This progress has been driven most notably by the development of Theano, Caffe, TensorFlow and later Keras. After its release in early 2015, Keras quickly became the go-to deep-learning solution for large numbers of new startups, graduate students, and researchers pivoting into the field [49]. Although the mathematical concepts behind deep learning have existed for decades, libraries for creating and training these deep models have only been available recently. Therefore, increasing popularity of NN nowadays is due to the lucky combination of several factors. What can be surely said is that there will be certainly further practical applications that could not be done without neural networks.

In case of neural networks and other soft computing systems, usually three types of learning can be distinguished. The first one is supervised learning, when the network learns from input-output patterns, using labelled datasets (positive and negative patterns). The second is unsupervised leaning, where learning is done without the specified target values, here the purpose is to find patterns and relationships. The third one is reinforcement learning, where learning is done through interaction with the system and feedback to the original system and to the network is an important component. Form the point of the work supervised learning is the most important.

2.4.3 Components, connection structures of a network

2.4.3.1. Neurons

Neuron is a fundamental component of the neural network, and similarly to the artificial neural network is inspired by the biological one, the artificial neuron has comparable function to the biological neuron. As mentioned in the previous section, the in biological neuron the dendrite receives electrical signals from the axons of other neurons, the signals (input) of the artificial neuron are numerical values, see Figure 40. Neuron is a generalization of the idea of the perceptron, which is a simple function that maps its multiple inputs to a single output value. A neuron does one thing: calculates the weighted sum ($w_0 \,..\, w_{n-1}$) of n inputs ($x_0 \,..\, x_{n-1}$), adds a constant called the bias (b) and then feeds the result through some non-linear activation function (f). Some of the well-known

activation functions are: Hard-limit, Sigmoid, Softmax, ReLU, etc. [49]. Similarly to the weight W, the bias b is also constant (parameter) but the two kind of parameters have different roles. The weights are multiplied directly by the input from a previous neuron, while the bias is added to the combined sum of the other weights. The reason why bias is needed is to give an additional degree of freedom to outspreads how operations work on tensors with incompatible dimensions. The general way of the bias addition is called the broadcasting add, which differs from a classical addition. This means basically: if there exist two tensors which cannot be added because their dimensions are not compatible, then repeat the small one as much as needed to make the addition work.

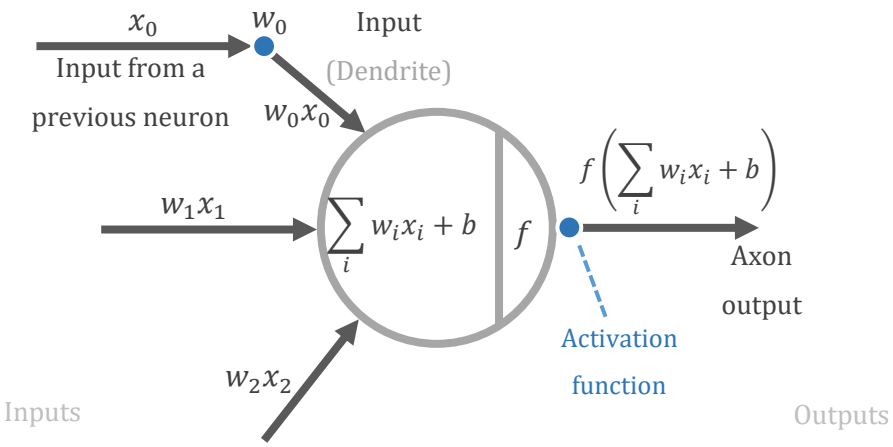


Figure 40 - Model of an artificial neuron [38]

In other terminology the input signal is called excitation, and the output signal is the activation. An important term for neurons is that the neuron "fires" or gets excited, which means that activation is above a certain limit, in other words the inputs generate a strong stimulus, so the network found something important regarding the input data. The output of the biological neuron is binary; it fires merely if the input is higher than a certain threshold. The dendrites or input vectors allow the biological cell to receive signals from a large (more than thousand) number of connected neurons. According the weight value, each dendrite is able to perform multiplication. That multiplication is accomplished by growing or declining the ratio of synaptic neurotransmitters to signal chemicals introduced into the dendrite in response to the synaptic neurotransmitter. Otherwise

artificial neurons are more flexible; their output signal can be arbitrary. Please note that from now on if not particularly noted, the subject will be the artificial network, not the biological one.

The simple activation function can be described in the following formula, equation (50) which is a simple one neuron, on layered neural network:

$$\boldsymbol{Y} = f(x_i, W, b) = f\left(\sum_i w_i \cdot x_i + b\right) \tag{50}$$

- $\boldsymbol{W}$ - weight, one of the input parameters, $\boldsymbol{W} = \{w_0 \,..\, w_{n-1} \mid n \in \mathbb{Z}\}$
- b - bias, the other input parameter
- x_i - actual inputs
- f - activation function (e.g. sigmoid, ReLu, softmax)
- $\boldsymbol{Y}$ - predicted output $\boldsymbol{Y} = \{y_0, y_1 \,..\, y_{n-1} \mid n \in \mathbb{Z}\}$
- $\boldsymbol{Y}'$ - the actual (desired) output, the so called ground truth, not in the equation, discussed later $\boldsymbol{Y}' = \{y'_0, y'_1 \,..\, y'_{n-1} \mid n \in \mathbb{Z}\}$

2.4.3.2. Activation functions

In various computing networks, the activation function (f) is one of the key components which determines the output of the given node, input or input set. From neural networks's point of view, one of the main roles of the activation function is to bring nonlinearity to the network. Over the years numerous activation functions became noticeable, in the following sections they will be introduced shortly.

2.4.3.2.1 Hard-limit

The mathematical formulation of the non-linear hard-limit activation function is:

$$f_1(x) = \begin{cases} 0, & if \;\; x < 0 \\ 1, & if \; x \geq 0 \end{cases} \tag{51}$$

As it can be seen from the definition (51), this is fundamentally a unit step function or in another terminology a Heaviside step function.

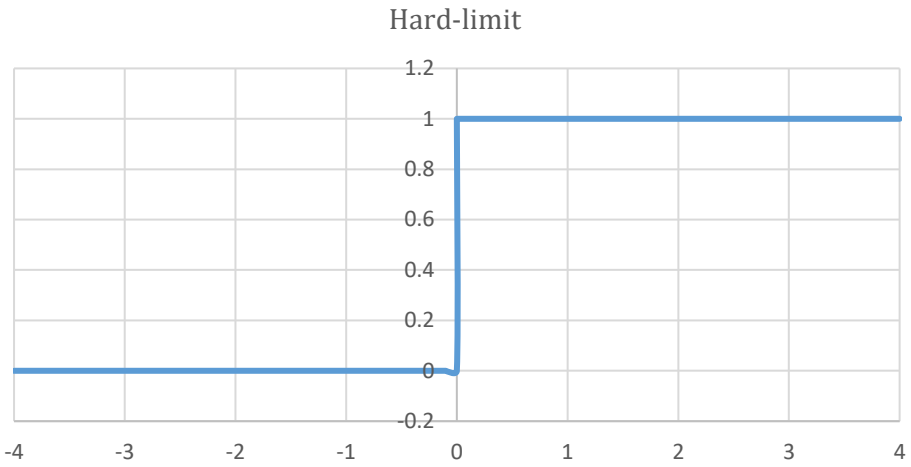


Figure 41 – Hard-limit activation function

In practice, its significance is low, yet perhaps this is one of the simplest non-linear functions we could theoretically apply.

2.4.3.2.2 Sigmoid and hyperbolic tangent

Sigmoid activation function is denoted usually by lower-case Greek letter sigma (σ). The mathematical formulation of the non-linear sigmoid activation function is:

$$f_2(x) = \sigma(x) = \frac{1}{1+e^{-x}} \tag{52}$$

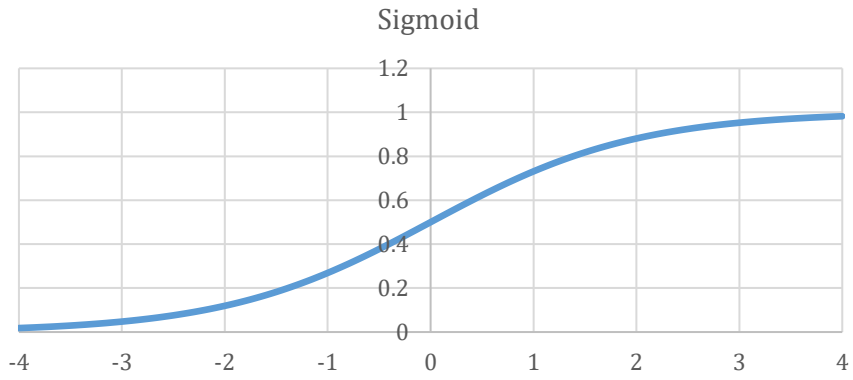


Figure 42 - Sigmoid activation function

The output takes a real number and transforms it between "`0`" and "`1`", so its output interval is [`0,1`]. High negative numbers lead to `0` and big positive numbers to `1`. The

sigmoid function have been used in the past, because it can interpret the "firing" of a neuron attractively. At low values, it does not fire at all (0), while at high values it is at maximum (1). In practice, the sigmoid is rarely used as it used to be, and there are two main reason for that.

The first reason is that the sigmoid saturates and vanishes, in other words diminishes the gradients. The problem is neuron outputs and their gradients can vanish entirely, this phenomenon is called the *vanishing gradient* problem. This is a highly undesirable property of sigmoid where the activation of the neuron at the two endpoints is saturated with 0 or 1, and in these regions the gradient is nearly zero. In the backpropagation algorithm, the local gradient will be multiplied by the gradient of the gate output that exits through the entire network. Therefore, if the local gradient is very small, it effectively "obscures" the gradient, and almost no signal flows through the neuron and recursively to its data.

Another aspect of this phenomenon is that particular attention should be paid to the initialization of sigmoid neurons to prevent saturation. For example, if the initial weights are too large, most neurons will be saturated and the network barely learns.

The second reason why the sigmoid function is being used less and less in practice is that the sigmoid outputs are not zero centred. In other words, the interval [0,1] is 0.5 centred. This is undesirable because later layers of neuronal networks also receive non-zero centre inputs. This affects the descent dynamics of the gradient, because if the data entering the neuron is always positive, then the gradient of weights in the backpropagation will either be positive or negative. Desirable would be logically to obtain weights of variable sign. A solution to this might be the tangent hyperbolic function $f_3(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$.

As the formula shows, the tangent hyperbolic can be understood as a sigmoid, but on the interval at [-1,1]. This is providentially zero centred, but the gradient saturation and damping characteristics remains.

2.4.3.2.3 ReLU

Rectified Linear Unit function (ReLU) is a popular choice for its expedient characteristics and simple definition. The ReLU is simply an activation function which has an assigned value of 0 under 0, otherwise it is itself. This can be mathematically formulated as follows:

$$f_4(x) = \begin{cases} 0, & if \ x < 0 \\ x, & if \ x \geq 0 \end{cases} = \max(0, x) \tag{53}$$

A while ago, in 2012 Alex Krizhevsky et al. [47] showed in his famous AlexNet paper that a four-layer convolutional neural network with ReLU activation on a CIFAR-10 [42] database to 25% training error can be done six times faster compared to the above-mentioned tangent hyperbolic activation. The CIFAR-10 is a famous dataset consists of 60000 colour images (32x32 pixel) which is divided into 10 classes. There are 50000 training images and 10000 test images. The disadvantage of ReLU is that the high gradient through the neuron can update the weights so that the neuron will never activate at any data point. Then the gradient flowing through the unit can be zero forever from the given point. This can also be understood as an unintended dropout, but it is obvious that this is not a favourable effect at all. However, their advantage over tangent hyperbolic or sigmoid activations is that the costly operations there can be replaced by a simple, computationally cheap threshold operation.

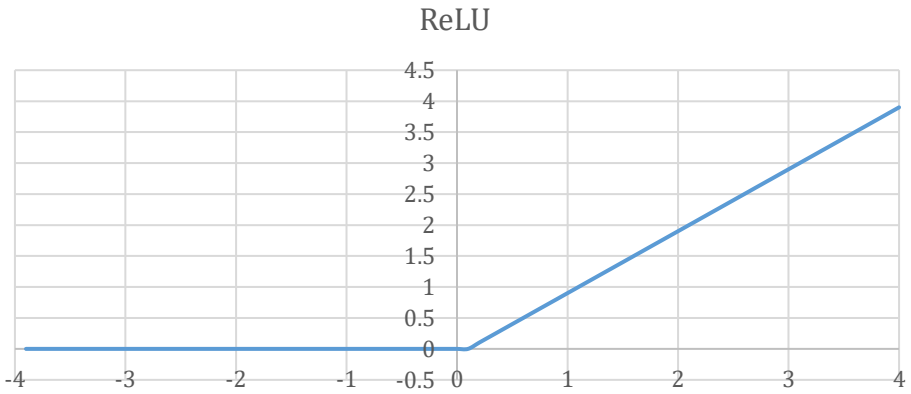


Figure 43 - ReLU activation function

Leaky ReLU, also called parameterized ReLU (PReLU) $f_5(x) = \begin{cases} \alpha x, if \ x < 0 \\ x, if \ x \geq 0 \end{cases} | \alpha <$ 1 differs from the classic version by taking a value multiplied by an $\alpha < 1$ parameter in the range of less than 0. Behind the creation of leaky ReLU is the attempt to induce the unintentional dropout property of ReLU. This intention is successful in some cases, but unfortunately it is not consistent.

2.4.3.2.4 Softmax

Another name for softmax is the normalized exponential function. It can be formulated as:

$$f_6(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \; for \; i = 1, \dots K \tag{54}$$

Fundamentally, how it differs from ReLU and sigmoid in that it calculates probability distribution, so it is more suitable for classification. Softmax, in contrast to the above, is a multinomial regression activation function. Instead of the x parameter vector, this function receives values marked with x_i. The softmax function is characterized by the fact that produces steeply increasing values, practically increasing the differences between the elements of the input vector. In this way, it also rapidly adds high values as an intermediate step, even before the output is valued. Then, normalization occurs and during that the normal dominant element returns to a value closer to 1, while all other elements normalize to about 0, and this appears on the output. Knowing this, the name even makes sense: it keeps the original order of the values so it is "soft", but it clearly empathizes the largest element so it is "max".

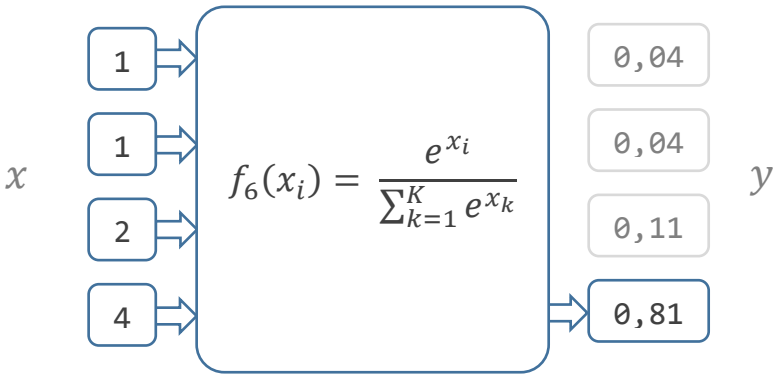


Figure 44 - Softmax activation function

As described above, the function was not plotted on the graph, instead as an output response to the given inputs.

2.4.3.3. Learning process

The learning process of a neural network usually requires a training, a validation and a test dataset. In supervised learning all of these datasets contains input (***X***) data and the desired output data (***Y***' - which is sometimes referred as labelled or annotated ground truth). The model is fitted on a training dataset, this specifies the values the weights and biases. As the name suggests the validation dataset is used for validation purpose during

the training, but it is not involved directly in the training process. This means that with the learned weights and biases, from time to time an estimation is evaluated, this generates the predicted output data ($\boldsymbol{Y}$). The predicted and the actual output is compared and this can help to determine for example if the model overfitting or not. Overfitting means whether the neural networks only works on the training set only or on other data too. Unintentionally overfitting can also happen via some parameter changes without even noticing it. So it is advised to keep a test dataset too which can be used to provide an unbiased evaluation of a final model.

The loss function at first can be viewed as a measure how erroneous the model is. The entropy loss, or simply loss is a function of weights, biases, input data and the desired output data. When dealing with a neural network which performs classification or prediction, it is generally better to use cross entropy error than mean squared error to evaluate the quality of the neural network. The short explanation for that is that mean squared error gives too much emphasis to the unfitting outputs. Mean squared error (or sometimes referred as L2 norm) is computed as the average of squared differences between the predicted output (y) and ground truths (y'):

$$MSE(y, y') = \frac{1}{N}\sum_{i=1}^{N}(y_i' - y_i)^2 \tag{55}$$

Then cross-entropy loss is defined as negative sum of the logarithmic ground truths (y') times the predicted output (y):

$$CEL(y, y') = -\sum_{i=1}^{N} y_i' \log(y_i) \tag{56}$$

The loss can be calculated both on training and validation set it defines how well the model is doing regarding these two sets. Not any function can be chosen though, there are some criteria which needed to be met first. The first one is that the loss function C can be expressed as an average over loss functions C_x for individual training examples and secondly, the loss function C can be written as a function of the outputs f^L.

The inputs x and the desired outputs y are not learned by the network but play a crucial part in how the weights and biases will be updated by making use of loss function C. Once again, this process is called the training of the network. The goal is to find how changing

the weights and biases of the network affects the loss function, this is done by computing the partial derivatives of weights $\partial C/\partial w_{jk}^{l}$ and biases $\partial C/\partial b_{j}^{l}$, along with introducing an error term δ_j^l of j^{th} neuron in l^{th} layer with weighted inputs x_j^i as [50]:

$$\delta_j^l = \frac{\partial C}{\partial w_{jk}^{l}} \tag{57}$$

using the chain rule, the formula is given as [50]:

$$\delta_j^L = \frac{\partial C}{\partial w_j^L} = \sum_k \frac{\partial C}{\partial w_k^L} \frac{\partial f_k^L}{\partial x_j^L} \tag{58}$$

The l^{th} layer error is denoted with δ_j^l. The number of neurons k in the output layer, denoted as L, is determined by a partial derivatives according to the output activations. The output activation function f_k^L of the k^{th} neuron depends only on the input x_j^L when $k = j$. Consequently the error term for the output layer's j^{th} neuron is simplified. In order to compute the output layer's errors (59) all previous layer's error (60) need to be calculated. As a result, the output layer's error needs to *propagate* its error to the previous layer, similarly the second one to its previous layer and so on. That is where the name backpropagation comes from.

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial x_j^L} = \frac{\partial C}{\partial f_j^L} f'(x_j^L) \tag{59}$$

$$\delta_j^l = \sum_k \delta_j^{l+1} \, w_{kj}^{l+1} \, f'(x_j^l) \tag{60}$$

In the following the partial derivatives of the weight $\partial C/\partial w_{jk}^{l}$ (61) and bias $\partial C/\partial b_{j}^{l}$ (62) is computed.

$$\frac{\partial C}{\partial w_{jk}^{l}} = \delta_j^l \, f_k^{l-1} \tag{61}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{62}$$

As seen in the previous description backpropagation is powerful algorithm which is used to train deep neural networks. It works by propagating a feedback signal from the output loss from previous layers. The earlier mentioned vanishing gradient problem is more understandable in this context. If the feedback signal has to be propagated through more layers (e.g. in deep neural networks), sometimes the input becomes weak or even be completely vanished. For more efficient learning, during (or after) the neural network

training it is advised to monitor some quantities. The parameters, hyperparameters principally influence how successful the training process is. One of the most important quantity to observe during or after the training is the training is the cross entropy loss. Contrast to the model accuracy, cross entropy loss is not a percentage. Generally speaking, accuracy is a percentage of the dataset correctness. For a single prediction and ground truth pair correctness is either be `0` or `1`, the accuracy over a batch or partition of the dataset as a percentage of the correct predictions. With N predictions $Y = \{y_0, y_1 \; .. \; y_{N-1} \mid N \in \mathbb{Z}\}$ and N ground truths $Y' = \{y'_0, y'_1 \; .. \; y'_{N-1} \mid N \in \mathbb{Z}\}$, accuracy can be formulated as:

$$\text{correct}(y, y') = \begin{cases} 1 & if \ \text{argmax}(y) = \text{argmax}(y') \\ 0 & otherwise \end{cases} \tag{63}$$

$$\text{accuracy}(Y, Y') = \frac{1}{N} \sum_{i=0}^{N-1} \text{correct}(y_i, y'_i) \tag{64}$$

If the partial derivatives of the cross entropy loss is computed relatively to all the weights and biases, we obtain a gradient. In Figure 45 typical learning rates are visualized. The x axis displays epochs; the y axis displays loss. An epoch means that the whole training data has been involved in the training process exactly once.

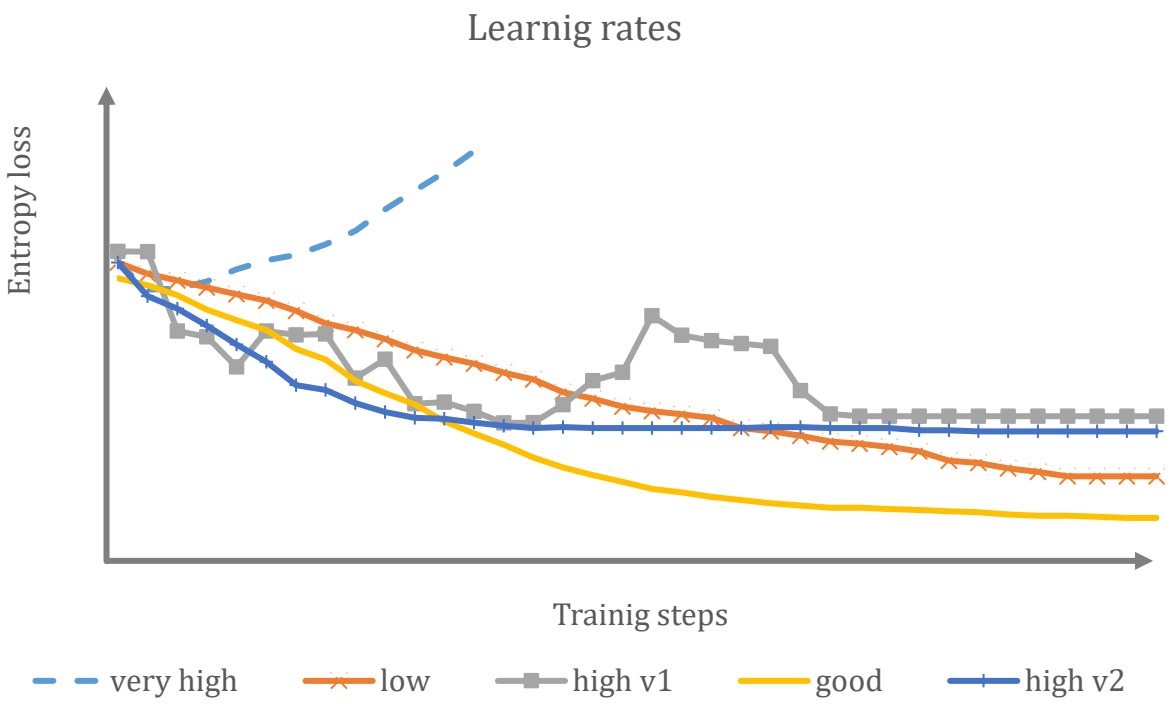


Figure 45 - Visualization of learning rate characteristics on training data

As for the learning rates in Figure 45 if the training rate is chosen very high it can happen that the model does not even learn. This is represented as the only increscent function. All other functions are decreasing, such as the two other version (high v1 and v2) of high training rate. One of them is not converging fast enough the other is fluctuating too much. If the learning rate is chosen too low the model shows converging characteristics, but in a very slow way. Finally, if the good learning rate is chosen it a faster converging function can be observed.

2.4.4 Topologies, architectures

There are different possible ways of linking the neurons together, thus creating multiple types of neural network structures. The architecture of an artificial neural network determines how the neurons structure is arranged, positioned and how they interact with each other. These solutions are often inspired by the synaptic connections of the nerve cells, and thus it is possible to control the connections between neural networks. The topology [51] of a given neural network can be defined within a given architecture along different structural compositions. In other words, more topologies can be built on the same architecture where the topology is composed of neurons of different enumeration.

For example, an architecture can be a fully connected feed forward network or a convolutional neural network, in contrast a topology can be AlexNet or VGG-VD which happens to be both convolutional neural networks. On the other hand, the training of a particular architecture includes how to set the weight value of the interfaces between neurons and the ability to filter the transmission signal. The denominations of neural networks are not uniform. As usually, there are historical reasons in the background of this phenomenon. In the early stages of neural networks, newly created topologies often spread very quickly to public knowledge and began to be labelled under a separate name, even if they did not necessarily report elementary novelty. For example, variational autoencoders (VAEs) are practically different from autoencoders (AEs) only in training, not in topology, but are considered as separate topologies, although they are not. In terms of abbreviations, not everything is consistent: for example, RNN meant to be recursive neural network, but more recently it used for the recurrent neural networks. In addition, there is more disagreement in this field of science, but I will try to use the most common names below.

2.4.4.1. Feed forward neural networks

Feed forward neural networks (FF or FFNN) is one of the most simple and ancient neural networks [52]. Neural networks are often described as layers, where each layer consists of parallel or input, or hidden, or output neurons. The neural networks containing the hidden layers will be detailed in the Deep neural networks section, these are extensions of the simple feed forward neural networks. The fundamental remark here is that a layer never has connections to itself but adjacent layers are *fully* connected.

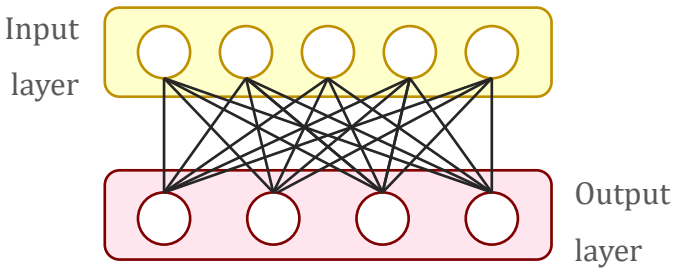


Figure 46 - Feed forward neural network

A simple feed forward neural networks can contain only one input and one output layer, where the two layers are fully connected. In Figure 46 a neural network containing two layers, 5 input neurons and 4 output neurons is illustrated. It is important that the signal processing flow is consequent from input to output; in the current thesis it will be illustrated from top to the bottom. Usually, this type of network is trained by supervised learning, where the neural network algorithm compares the output values with the desired value. The difference that is calculated, namely the error, allows the training algorithm to change weights and biases, progressively aligning with the output of the desired value.

2.4.4.2. Deep neural networks

Deep neural network, sometimes referenced as deep feedforward networks, or multilayer perceptron is the most typical deep learning architecture.

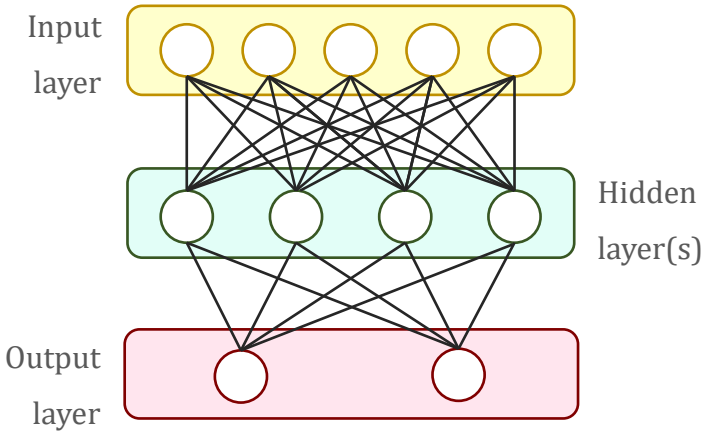


Figure 47 - Deep neural network

The deep neural network is a type of feed forward neural network where the neurons are located in more than two layers, so that at least one "hidden" layer is located between the input and output layers. There are 3 types of layer.

- Input layer: This layer is responsible for receiving input data, signals, and features from the outside world. These inputs are normalized by the activation function within a specified numeric value, so they are prepared for processing within the network.
- Hidden layer(s): These are in-between layers, which are responsible for interpreting the processes, or patterns you want to analyse from the input. The first hidden layer is connected to the input layer likewise the last layer is connected to the output layer.
- Output layer: This layer shows the valuation after the training process based on input data - which is evaluated by hidden layers of neurons - which is the actual output of the network.

The quantity of neurons in the first hidden layer usually differs from the number of signals in the input layer of the network. In practice, the quantity of hidden layers and their particular number of neurons depend on the nature and complexity of the problem being solved by the neural network, and the quantity and quality of the dataset for the given problem too.

2.4.4.3. Recurrent neural networks

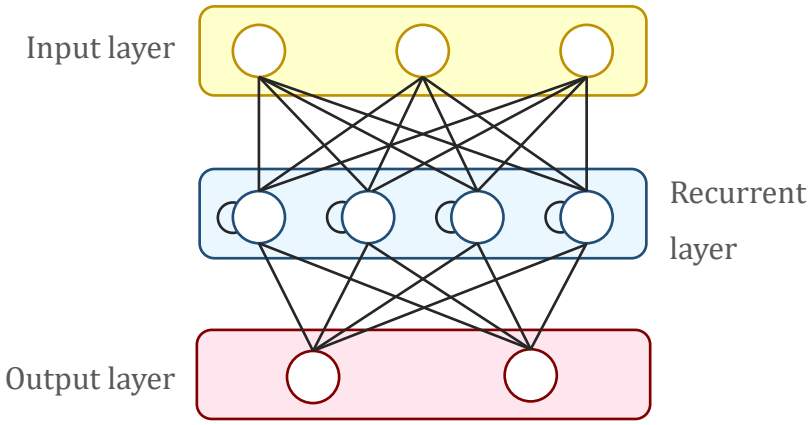


Figure 48 - Recurrent neural network

The recurrent neural networks (RNN) are feed forward, deep neural networks, that represent the state and time. [52] Neurons are not only related to the previous layer but also to themselves, and more specifically to their previous state. This is indicated in Figure 48 by the connections that have been returned to themselves. One of the major problems with RNN is the vanishing gradient problem, where the information quickly disappears over time, depending on the activation functions. These networks are designed to integrate some kind of memory, with the neurons connected to itself, so they can be applied in time-variant systems, such as time series prediction, optimization, speech analysis, process control or video stream classification, etc.

2.4.4.4. Hopfield

The Hopfield Network (HN) is a neural network where all neurons are connected to each other. Virtually every neuron in the network is connected with weighted connection the others. All nodes functions as input before the training process and after that it hides this functionality during the application, so basically it becomes output. Networks are designed to set the values of the neurons to the desired pattern, after which the weights can be calculated. After that the weights do not change. Having been trained for one or more samples, the network always converges to one of the learned patterns. These networks are often referred to as associative memory because conversion converts the input to the most similar state. The network is single-layer, the input layer provides the output. In the network, neurons can take two values, -1 or 1, but they are usually

converted to 0, 1 for clarity. The network was first published by John Hopfield [53], an American biologist, and it was named after him.

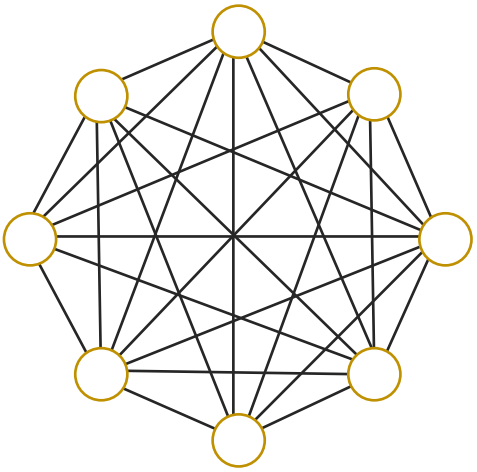


Figure 49 - Hopfield neural network

2.4.4.5. Boltzmann machines

Boltzmann machines (BMs) are very similar to HNs, but some neurons are designated as hidden neurons and others as inputs. The input neurons become output neurons at the end of a full network upgrade. Compared to HN, neurons usually have a binary activation pattern.

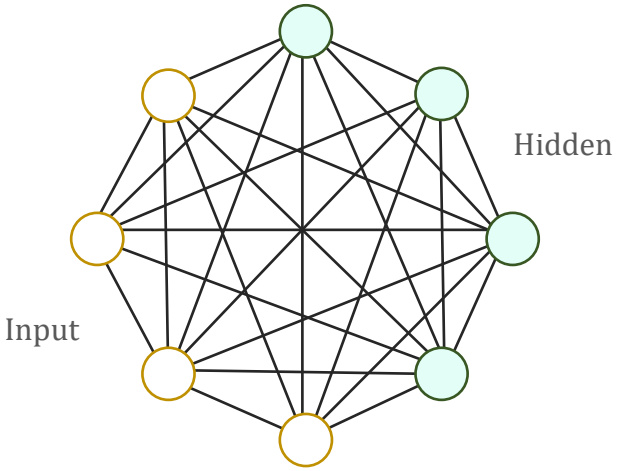


Figure 50 - Boltzmann machines

2.4.4.6. Convolutional neural networks

Convolutional Neural Networks (CNN) was the idea of Yann LeCun. The name of LeCun also can sound familiar because he is currently the Chief AI Scientist at Facebook AI research. LeCun's inspiration was the work of two biologists, David Hubel and Torsten Wiesel, who studied the visual cortex of mammals [42] [43] [44]. Perhaps this is the most

important type of neural network regarding my research. From the neural networks described so far, convolutional networks differ in their use (most often for processing complex information such as image, sound, etc.) and in that most layers are not fully connected to the layer before it. From my doctoral thesis point of view, image recognition or its narrow domain, environmental perception is the most important field. I would like to note here that I will distinct perception from sensing. Under (environmental) sensing, I mean data acquisition and transmission by the sensor (usually camera), while (environmental) perception is a more complex processes, because the interpretation of the collected data is done here.

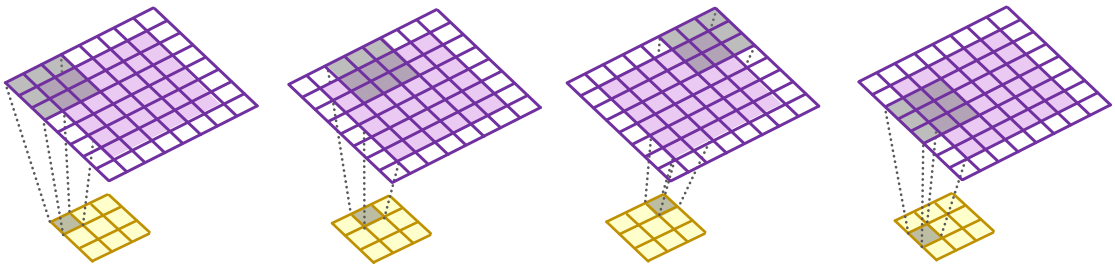


Figure 51 – Illustration of convoluting with a 3×3 kernel over a 6×6 input with 2×2 strides and 1×1 padding

In a fully connected layer every neuron in the upper layer is connected to every neuron in the layer below and each connection has its own weight. This is considered as a general purpose connection pattern and makes no assumptions about the features in the data. This solution is also very expensive regarding memory (weights and biases) and computation (connections). On the other hand, in a convolution layer the connection is defined by the convolutional kernel. This kernel iterates through the complex input information (e.g. image) and the kernel learns from local information from adjacent elements (e.g. set of pixels). Figure 51 and Figure 52 illustrates the main components of the convolution operation, the kernel, the stride and the padding. A kernel can be theoretically any smaller matrix than the input, but usually a squared small matrix is chosen, e.g. in 2D 3x3, 5x5, 7x7, in 3D e.g. 4x4x10, 5x5x3 are common. Stride determines how many pixels the convolutional kernel skips when it iterates trough a layer, thereby reduces the size of the next layer. Thus it permits to resolve how much overlap should kept between two output layer. When the stride is 1 then the kernel moves one pixel at a

time, accordingly if it is 2 it moves 2 pixels, and so on. Padding also determines the next size of the layer, but it has a different mechanism. Padding prevents the next layer size to get reduced, in other words adequate padding will keep the layer size intact. Roughly speaking padding is an extension of the image or layer at the borders.

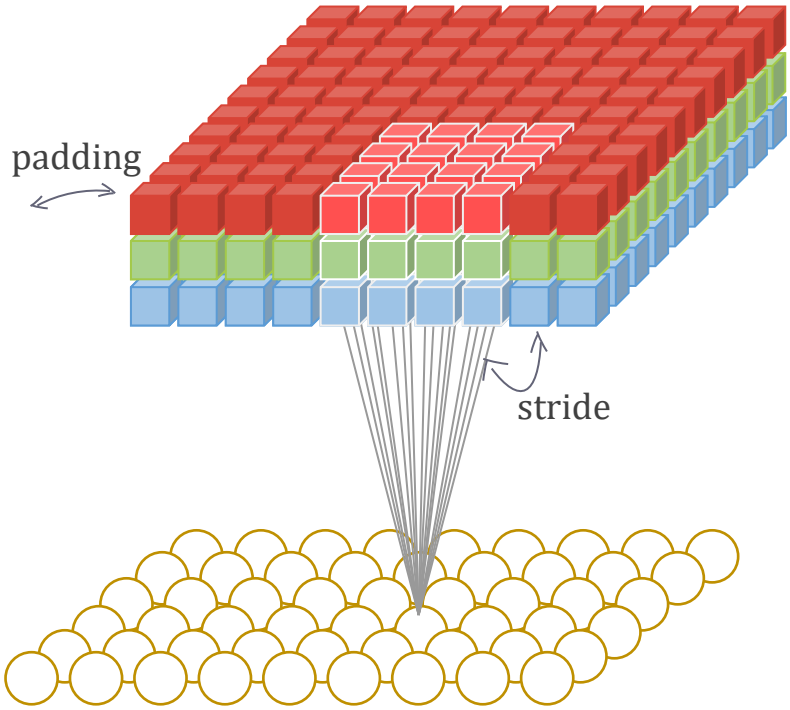


Figure 52 – Illustration of convolution on an RGB image

As mentioned earlier, convolution operation is the process of adding each element of the image to its local neighbours, weighted by the kernel. Basically here only the kernel weights need to be determined not a layer-to-layer fully connected weights. This is of course a serious reduction in means of memory. According to the calculated kernel operations such as edge detection, colour detection, sharpening, blurring and combining these: contour and object detection can be realized.

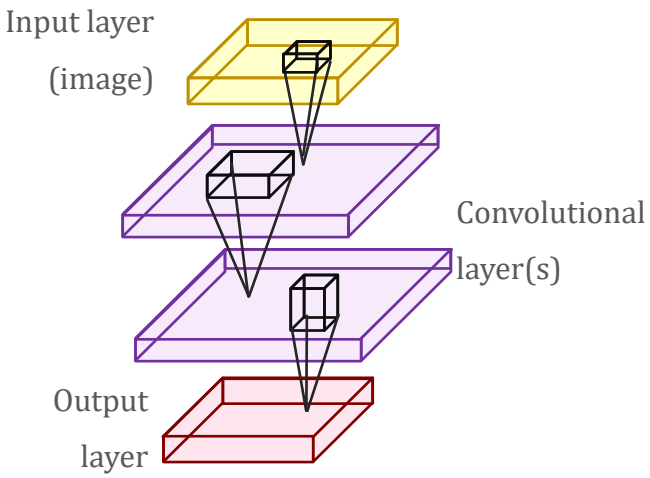


Figure 53 – Illustration of convolutional layers

Atrous convolution is the same as the general convolution, which are complemented to increase the "field of view" of the kernel. In the following the example of a 3x3 convolution filter will be examined. If the factor of dilation (expansion of the field of view of the kernel) is 1, it behaves like a standard convolution. But if the dilatation factor is set to 2, this will affect the expansion of the convolutional kernel.

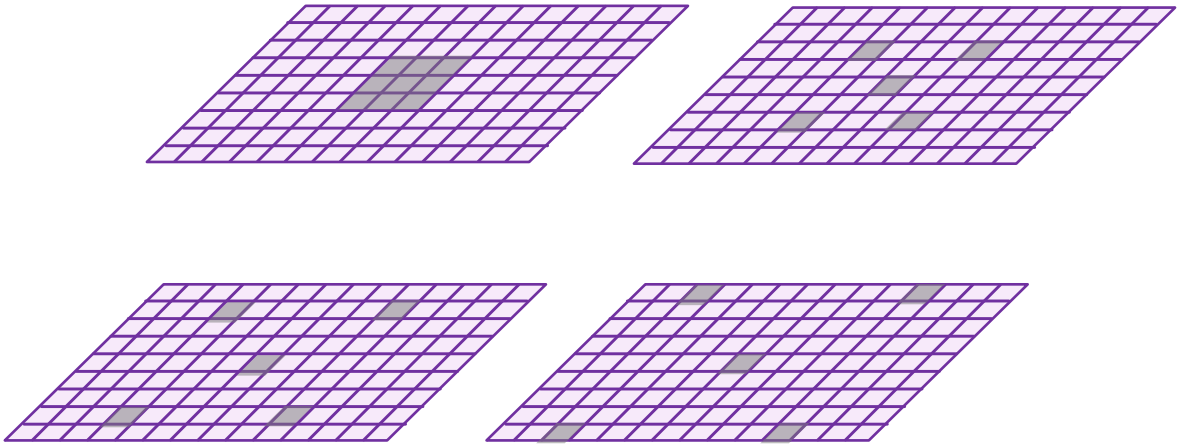


Figure 54 - The dilatation factor for the top left image is 1, the right 2, the bottom left 3, the bottom right 4; dilated value of 2 with a 3x3 kernel

The Atrous convolution results in a greater correlation between pixels without increasing the number of parameters or the amount of computation. The efficiency of the Atrous convolution largely depends on the well-chosen dilatation rate. As a rule of thumb, the value of the output stride is worth to choose to 8 or 16.

I highlight some of the important networks and also these networks will be presented in detail below: VGG-VD (Visual Geometry Group, University of Oxford) [46], AlexNet [47], GoogLeNet [48]. As it is known, image processing is a challenging task, caused by the change of viewpoint, the variation of object sizes, and the intra-class variation of objects [54]. One of the main challenges of image processing is the classification problem, which means that, using an algorithm for a given image determines which of the predefined classes is represented. AlexNet used for the classification problem, and it is one of the first CNNs, which has achieved good results in the most extensive image processing competition, ILSVRC (ImageNet Large Scale Visual Recognition Competition [55]) against neural networks and classical image processing approaches. AlexNet is a very simple, and small neural network. A VGG-VD (Visual Geometry Group- Very Deep Convolutional Network) is another famous network, it is much deeper than AlexNet, but its structure is still simple. In contrast GoogLeNet is a deep network and it is not a simple architecture at all, and nor it is sequential. Next to classification, segmentation is also a major problem where pixels need to be accurately labelled in the image. The segmentation problem is an intensely researched area, and nowadays neural network architectures solve it with great success [40] [56]. Many of them rely on the example of convolutional neural networks (CNN) or fully convolutional neural networks (FCN). In this case, FCN means that unlike simple CNN, the FCN only contains convolutional layers. For example, a widely known networks are BVLC FCN [57] and its varieties (FCN32-s, FCN16-s, FCN8-s), or even SegNet [58].

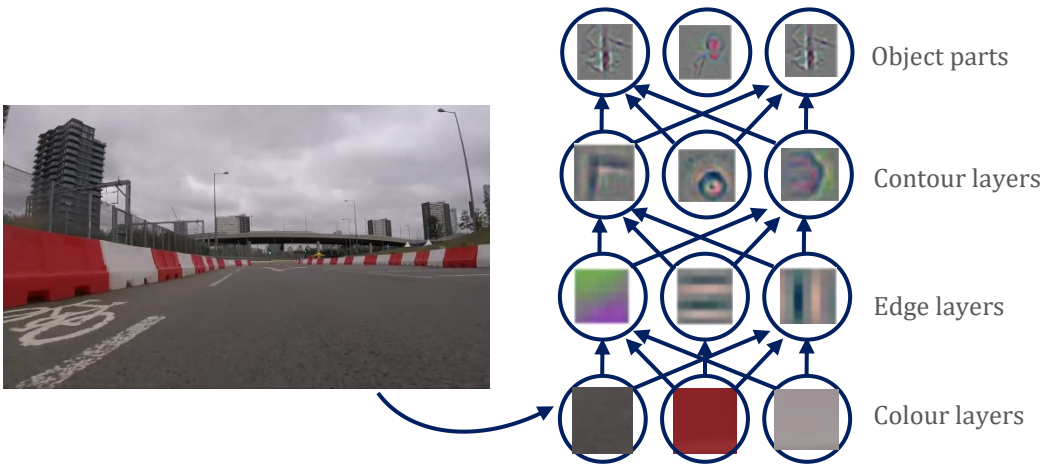


Figure 55 – Illustration of CNN logical layers

It is also possible to use CNNs that have been created for classification, but it can be extended and used for segmentation, by transfer learning. These networks then pass on their learned representations, and the often multi-week teaching process doesn't have to be repeated again. There are many examples of this approach such as AlexNet-FCN [59], a DeepLab-FCN-CRF [60], a VGG-FCN [46] [59].

2.4.5 Limits and usability of neural networks

According to Brandon Rohrer, Data Scientist from Facebook [61] intelligence can be defined as a product of *performance* and *generalization*. According to his thoughts formulated in 2018, human performance, marks a level corresponding to the competence of an expert. We can measure this in a several ways, such as the rate of error in performing a task, the time it takes to complete the task, the number of repetitions needed to complete the task, and so on. Generalization, on the other hand – regarding the previous example - is a set of tasks that a man can do. For example, to write a book, to bake a cookie, to build a house, or to discover the greatest mysteries in the universe. The degree of generalization is the greater as more tasks are involved, and also it is proportional to the degree of structuralism inside the task. To illustrate the theory of performance and generalization Figure 56 shows an evaluation of various tasks.

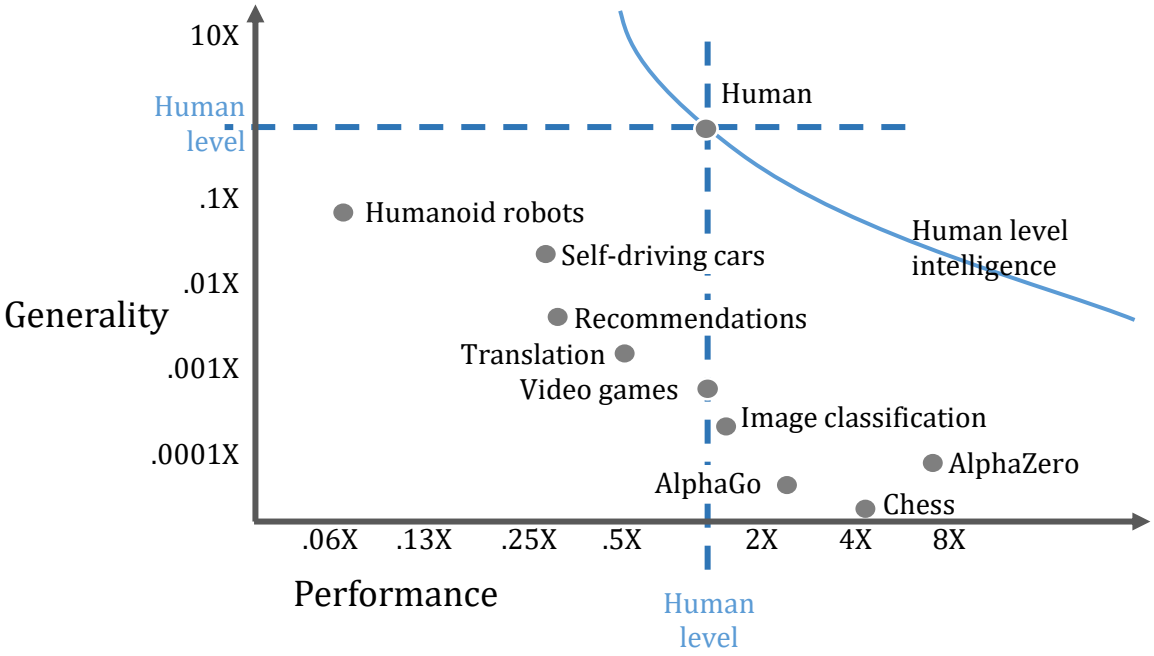


Figure 56 - Performance and generality evaluation of various task compared to the human level [61]

This theory harmonizes with the thoughts of Joi Ito, who is a professor at the MIT. Ito formalized his thoughts in 2018 [62] and according to them an NN can be more accurate in recognizing "dog" and "cat" images than an average person, however, the statistical models produced lack the ability to understand the meaning of these animals, and objects in general. Staying with the example, dogs and cats are animals that mostly behave in a friendly manner with people, but this information is still not obtainable by an NN. Due to of this phenomenon, such systems need huge amounts of data, since they do not intend to understand the complex processes of our world but are designed to pattern matching. There is indeed a significant difference between simple pattern matching and understanding.

Still NN have and will probably gain more interest regarding computational intelligence applications. It is a commonly known fact, that classical approaches of computer vision cannot perform as good and as general as the neural network-based approaches. Also they have the advantageous characteristics of being able to scaled. This means if e.g. more complex or more precise result is needed, the larger, computationally heavier network may can help. And this works vice-versa, if there are performance limits, a less accurate

network may still do the task. The drawback of this approach is that it is hard to validate and as the human recognition, it can be erroneous, but not by all means in the same way.

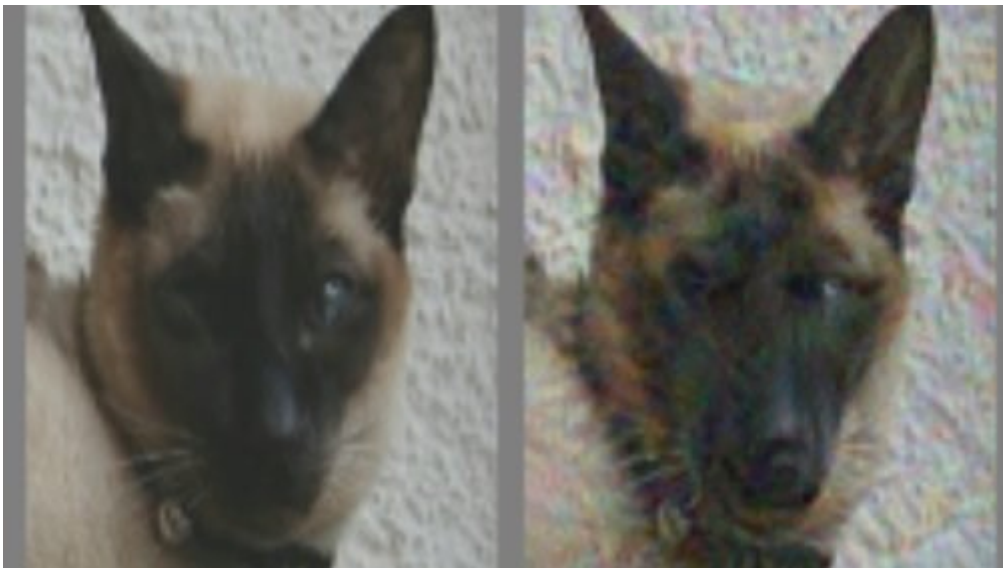


Figure 57 – Deception illustrated with a cat and a dog, small pixel changes can make us and a NN think that on the right picture a dog is present [63]

There are plenty of publications about few pixel changes, which can cause NN to make mistakes classifying images - called adversarial examples [63]. E.g. a dog can be mistaken to a school bus, and other hilarious examples; once again these examples are intentionally designed to cause a mistake. On the other hand, the modern-day neural networks can be so close to the human-level performance, that they adversarial examples can fool both human and neural network.

As mentioned, my thesis focuses on image recognition, especially road detection. The naïve way of road detection would be only to use traditional computer vision (CV) algorithms e.g. colour segmentation, region-growing methods or edge detection. These algorithms do not perform well in our case [64]. The reasons for that are the lightning conditions, the object's perspectives and sizes vary significantly, see Figure 58. Karpathy et al. gathered all the challenges of recognizing a visual concept, as follows: [54]

- Scale variation: Visual classes often show variation in their size: This means size in the real world, not only in terms of their amount in the image. [54]
- Viewpoint variation: A single instance of an object can be oriented in many ways with respect to the camera. [54]

- Deformation: Many objects of interest are not rigid bodies and can be deformed in extreme ways. [54]
- Occlusion: The objects of interest can be occluded. Sometimes only a small portion of an object (as little as few pixels) could be visible.
- Illumination conditions. The effects of illumination are drastic on the pixel level. [54]
- Background clutter. The objects may blend into their environment, making them hard to identify. [54]
- Intra-class variation. The classes of interest can often be relatively broad, such as vehicles in general. There are many different types of these vehicles or generally speaking objects, each with their own appearance. [54]

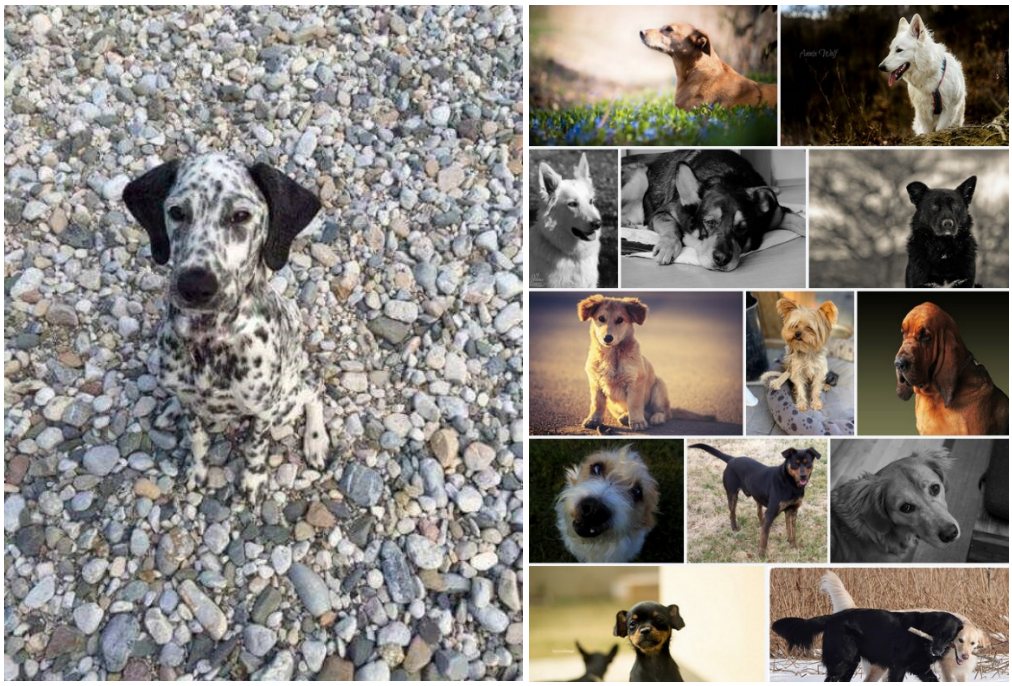


Figure 58 – Illustration of background clutter and intra-class variation

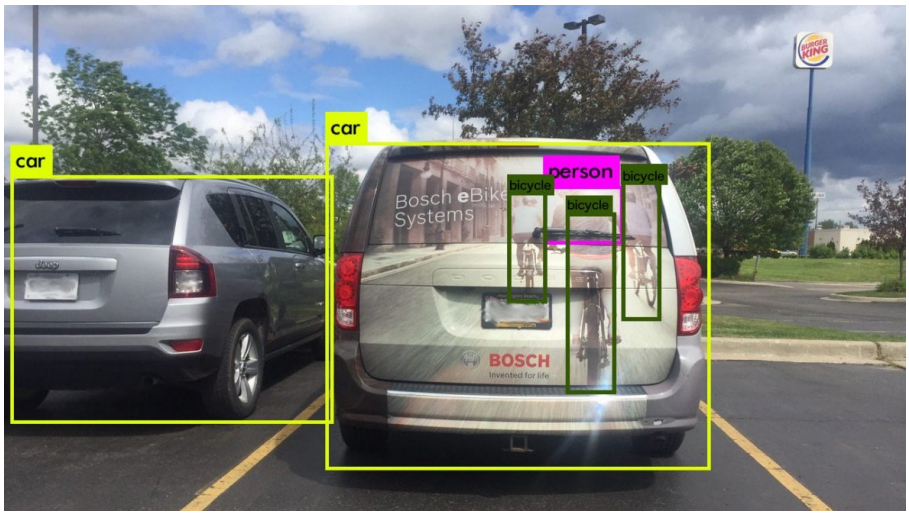


Figure 59 – False bicycle identification [65]

As mentioned, Drago Anguelov [3] solving extremities is one of the key problems of self-driving. In the field of neural networks these extremities appear when a trained network recognizes the objects but cannot relate it to the real-world's context. An example for this phenomenon is visible on Figure 59 when it has been fooled into thinking that images of bicycles on the back of a vehicle are genuine human cyclists. Luckily the problem is known and highly researched. There are also some early results which suggests that some neural networks may learn to organize the world into concepts, just like humans do. For example, Generative Adversarial Networks (GANs) [66] can be used to visualize what a network knows from trying to re-create the visual world. This can be also used for validation of a network and diminish its black-box properties.

To summarize the above mentioned, neural networks - as nowadays we know them - may not ideal for environmental perception, but still are the best known approach, and still have development opportunities and known limitations. I also would like to emphasise that there is a huge difference between object detection in a traffic situation and understanding the scenario of the same situation.

2.4.6 Current neural network framework

As a rule of thumb, it does not make sense to develop neural networks from scratch it would be a tedious task. On the other hand, choosing the right framework instead is generally a good idea but takes lot of consideration. As constantly newer neural network approaches rise, so do frameworks too. At the time of writing, there are several options

to choose from and I highlighted the most relevant ones regarding this work in Table 1. The basis of the comparison is the metrics used on the largest code-sharing service.

Table 1 - Neural network framework comparison

	GitHub	Stars	Contributors	Commits
TensorFlow	github.com/TensorFlow /TensorFlow	120 000	1 770	46 500
Keras	github.com/keras-team/keras	37 000	760	5 000
Caffe	github.com/BVLC/caffe	27 000	280	4 200
CNTK	github.com/Microsoft/CNTK	16 000	200	16 100
Caffe2	github.com/facebookarchive/caffe2	85 000	200	3 700
PyTorch	github.com/pytorch/pytorch/	24 000	880	16 000
MXNet	github.com/apache/incubator-mxnet	17 000	680	10 000

Well known tech companies are developing or supporting various frameworks for example Facebook develops and supports PyTorch and Caffe2, Google TensorFlow and Keras, Amazon and Apache MXNet, Microsoft CNTK.

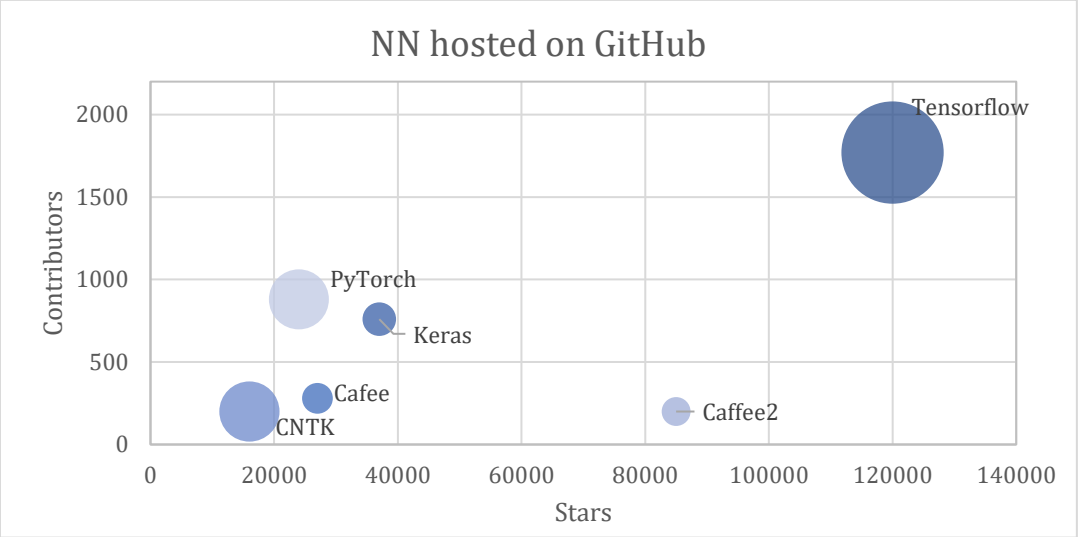


Figure 60 – Neural network frameworks hosted on GitHub (the size of the circle represents the commits)

In a thesis it does not make sense to write detailed description about specific technical realizations, these technical details change fast, rather I emphasize the general working principles of the current approaches. Of course there are numerous papers regarding the performance, usability, etc. comparison of these frameworks. For example, before TensorFlow 2.0, the framework was often criticized because of its session-based approach. According to this the tensor objects used to be accessible by running a

declarative code and then evaluating it using the sessions. So basically a session object encapsulated the environment in which operations are executed, and tensors are evaluated. In practice, it used to make harder to debug your code, and other rival frameworks did not follow this approach. But TensorFlow 2.0 changed it views about this approach and the eager execution made easier and more intuitive to debug. This is only one example, but it clearly shows that even when a framework has weaknesses the next release may fixes it. In the following, I will briefly write about the frameworks which are current at the time the thesis is written. *TensorFlow* is the definitely one of the leading neural network and generally speaking machine learning framework. A seen before it has the most GitHub activity, Google searches, online job offers, books and papers. *Keras* emerges from the other frameworks because of its usability for developers. Keras is rather an API standard for model definition, so it is not tied to a specific implementation. That is why for example TensorFlow Theano, or CNTK implemented it. *PyTorch* is also a popular overall framework, and as mentioned it is supported by Facebook. PyTorch has a deep integration into Python which allows popular and packages to be used for writing neural network layers. *Caffe* is one of the oldest framework in this list, but it has recidivist recent usage. Nowadays it is slowly replaced by *Caffe2*, which builds on Caffe but since new computation patterns have emerged, e.g. in distributed computation some design principles of it were abandoned. Caffe2 is now a very popular framework. *Theano* is the only framework in the list who is in the leaders for a really long time, it is the oldest significant Python-based machine learning / neural network framework. *MXNET* also a significant framework which is incubated by Amazon and Apache.

To conclude it, there are features according to which one or another is better. E.g. TensorFlow is faster when running large-scale models on the CPU-based platform, while Caffe2 is faster when running a small-scale model [67]. There is no apparent single framework which suits best for all purposes, so the choose is largely influenced by the field of usage, the requirements and also the personal preferences.

3. Contribution

This section is devoted to detail my scientific contribution. In the following three theses are proposed. The first one is a method for path construction. The second is a trajectory following method. The third one is a signature-based sensory model. The last one is a neural network phenomenon and a recommendation based on that.

3.1. Thesis 1.

I proposed a method to solve the Coverage Path Planning problem (CPP) called Iterative Structured Orientation Coverage (ISOC) and showed that it can lead to a shorter trajectory compared to the commonly used Boustrophedon Cellular Decomposition Coverage (BCDC). In the context of this work, I elaborated three different solutions where the first two relies on the rotation of parallel lines, whereas the third solution uses the concept of inertia. The suggested approaches are validated by simulation and experimental results.
References: [H19], [H14]

Coverage path planning (CPP) aims to cover a certain area with the shortest movements possible. This problem appears in the various domains such as agricultural applications, painting robots or cleaning robots. CPP makes use of two classes [18]: it is complete if it guarantees complete coverage or heuristic in other cases. There are also two main CPP strategies: offline if there is an a priori known map, otherwise online if the robot needs to discover the environment.
The most extensively used approaches to CPP are Random Path Planning (RPP) [12], Exact Cellular Decomposition (ECD) [13], Boustrophedon Cellular DeComposition (BCDC) [13], Backtracking Spiral Algorithm (BSA) [12], Internal Spiral Search (ISS) [14], U-turn A* Path Planning (UAPP) [14] and Neural Network (NN)-based CPP [15]. A critical analysis of these approaches is presented as follows.
Random Path Planning (RPP) imposes the mobile robot (MR) to move on several random trajectories. Each time when the current trajectories are obstructed a new one is chosen and next repeated. RPP is a simple but not effective approach. Combining RPP with pre-programmed trajectory patterns such as spirals and serpentines and/or a wall following mechanism the algorithm may increase efficiency [12]. Due to its simplicity, the path

planning algorithms specific to RPP are used in nowadays popular autonomous robotic vacuum cleaners.

Exact Cellular Decomposition (ECD) uses cells to fill the whole map. Usually the MR covers each cell using simple back-and-forth motions. After the current cell is covered, the robot moves to another cell. Finally, the whole map will be covered. The Trapezoidal Decomposition is a particular case of ECD given in [13], and characterized by the decomposition of the map into trapezoids, which are covered with simple back-and-forth motions.

Boustrophedon Cellular DeComposition (BCDC) has been introduced in [18] and the path planning algorithms based on BCDC became popular as highlighted in [14]. The word boustrophedon literally means "the way of the ox" in Greek [12]. The original supposition is that the map is composed from polygons, so it is a line map [15]. BCDC exploits this hypothesis and generates cells (easy to be covered) and finally generates the connection between these cells. BCDC performs an exact cellular decomposition, and each cell in the boustrophedon is covered with back and forth motions [12].

The drawback of the polygonal decomposition is the big number of cells. This phenomenon can be adjusted by merging cells [7]. If the cells are generated using a beam of parallel lines because of intersection with obstacles, these cells can be viewed as convex polygons that do not have any holes. Consequently, the cells are covered with back and forth motions. This approach leads to connected cell and ensures a complete coverage.

In the following I have proposed a new off-line approach, called Iterative Structured Orientation Coverage (ISOC), which will be introduced and compared with the previously existing algorithms. The proposed Iterative Structured Orientation Coverage method offers possible solutions for the coverage path planning problem and the improvement compared to the widely used [13], [18], [19] solutions are the following: it is general, works on complex environments and produces short trajectory.

ISOC uses discrete grid maps and targets the complete coverage. The specific feature of our approach is the combination of two ideas, considering the whole area as one unit and using the BCDC-based motion.

The ISOC approach uses the concept of main lines. These main lines are actually a beam of parallel lines, which have a particular orientation in the map. This orientation ensures a set of straight lines with maximum length, surrounded by the map and interrupted by the obstacles, see Figure 61. By composing (or linking) these lines we obtain the minimum

length path. The composition stage relates the solution with optimization problems. For example, it is associated with the traveling salesman problem (TSP), evolutionary-based algorithms, and its characteristics represents the advantage of the ISOC approach with respect to the state-of-the-art reported in [12], [13], [14], [15].

Three solutions to obtain the main lines are proposed. These solutions are inserted in the ISOC strategy resulting in three new ISOC approaches. All the suggested approaches are validated by simulation and experimental results.

As mentioned the ISOC approach uses the concept of main lines to cover the map (domain). The main lines are parallel to each other, their distances are related on the sensory range, because the coverage is connected to the range sensor's perception. Based on the main lines, the main segments can be created concluded by the structure of the map. The final trajectory is composed with the help of the auxiliary segments. These auxiliary segments intend to connect the main segments and thus make one single continuous trajectory. According to this approach, the movement mostly consists of the long main segments, which is one of the reasons why the Iterative Structured Orientation Coverage can lead to a shorter trajectory compared to the widely used Boustrophedon Cellular Decomposition Coverage [13]. Figure 61 (a) shows the creation of main lines and Figure 61(b) shows how to construct the main segments and connect it with the auxiliary segments to compose the trajectory.

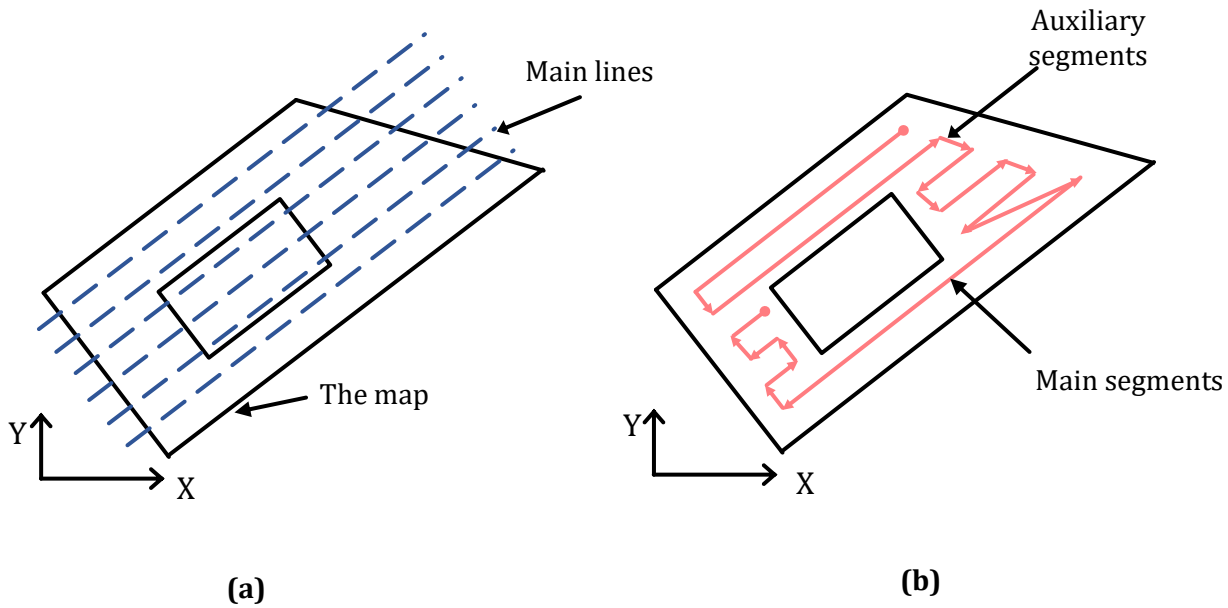


Figure 61 - Illustration of the main lines (a) and the main and auxiliary segments (b)

According to our best knowledge there was no realization of the Boustrophedon cellular decomposition for grid maps, even the realization is straightforward from line map to grid map. In order to appropriately compare the iterative structured orientation coverage (ISOC) with boustrophedon cellular decomposition coverage (BCDC) the same input set (maps) and the same programming environment (MATLAB) needed to be used. Figure 62 shows how important is to choose the right orientation. On Figure 62 (a) shorter but more main segments are determined in contrast on Figure 62 (c) where longer but less main lines are present. The possible connection is visualized on (b) and (d) which clearly shows that even in this simple case, the less main lines chosen, the shortest final trajectory may reached.

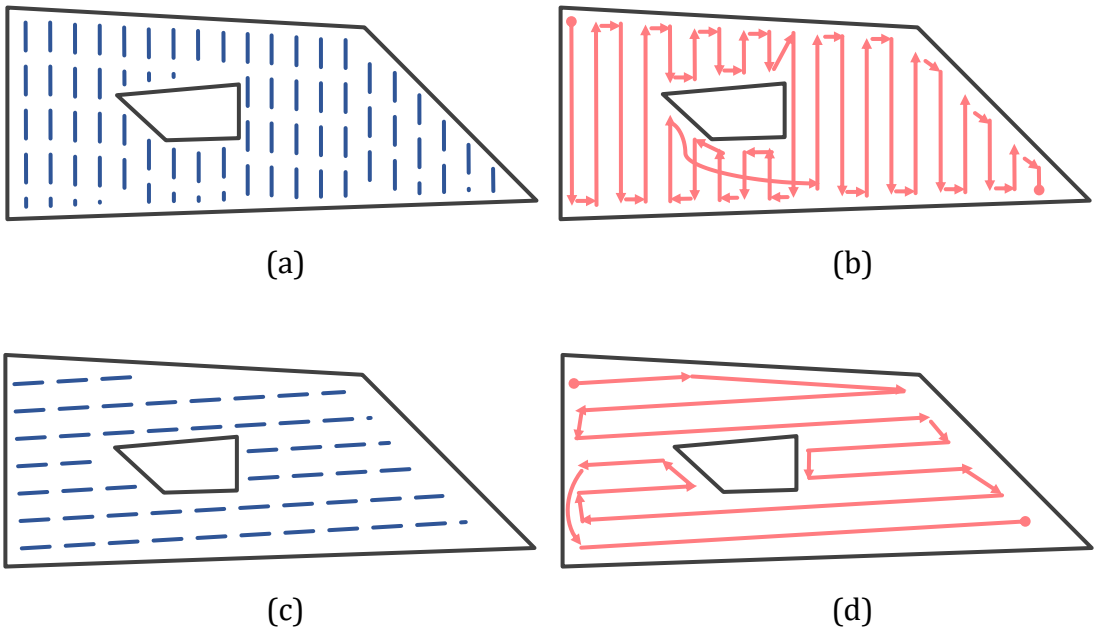


Figure 62 - Visualization of ISOC with two orientations, main lines are presented on (a) and (b), the whole path is on (b) and (d)

The initial data of the CPP approach is the map. The map is generated from a picture and modeled by the map matrix $M = [M_{\mathrm{ij}}]_{i=1\ldots n, j=1\ldots m} \in \Re^{n\times m}$, with the elements:

$$M_{\mathrm{ij}} = \begin{cases} 0, & \textit{if} \text{ cell } (i,j) \text{ is free } (\textit{white}) \\ ]0,1[& \text{likelihood of occupancy} \\ 1, & \textit{if} \text{ cell } (i,j) \text{ is occupied } (\textit{black}) \end{cases} \tag{65}$$

where n is the number of horizontal pixels and m is the number of vertical pixels.

Using the main lines concept, the problem of finding a minimum length path, which covers the whole map, reduces to the following 3 steps:

1. Find the appropriate main line, i.e., the orientation of the beam of parallel lines.
2. Construct main segments.
3. Link these segments with auxiliary segments such that the final (continuous) path has a minimum length.

These steps will be detailed in the following and later as a unified algorithm which is a part of the ISOC. For step 1 three different solution will be presented noted with solution "A", "B" and "C". The equation of the beam of parallel lines expressed in the discrete domain is:

$$\begin{cases} L_{q_1 k_1} \equiv i = \left\lfloor j \frac{q_1}{(m+1)} \right\rfloor + k_1 + 1, & \text{if } \alpha \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right], \\ L_{q_2 k_2} j = \left\lfloor i \frac{q_2}{(n+1)} \right\rfloor + k_2 + 1, & \text{if } \alpha \in [-\frac{\pi}{2}, -\frac{\pi}{4}) \cup (\frac{\pi}{4}, \frac{\pi}{2}] \end{cases} \tag{66}$$

where α is the line slope in the continuous domain, $\alpha = \tan^{-1}\left(\frac{q_1}{m+1}\right)$ or $\alpha = \tan^{-1}\left(\frac{q_2}{n+1}\right)$, $q_1 = 1...m$ or $q_2 = 1...n$ have a direct effect on the slope in the discrete domain, with the unified notation $q \in \{q_1, q_2\}$, $\lfloor x \rfloor$ indicates generally the integer part of $x \in \Re$, $k_1 = \beta_1 \delta_1$ or $k_2 = \beta_2 \delta_2$ is the intercept with the unified notation $k \in \{k_1, k_2\}$ for both horizontal and vertical axes, $\beta_1 = 0...(n-1)/\delta_1$, $\beta_2 = 0...(m-1)/\delta_2$ the integer steps δ_1 and δ_2 are computed in terms of

$$\delta_1 = \left\lfloor b \frac{\sqrt{q_1^2 + (m+1)^2}}{m+1} \right\rfloor, \ \delta_2 = \left\lfloor b \frac{\sqrt{q_2^2 + (n+1)^2}}{n+1} \right\rfloor, \tag{67}$$

where b is the distance between the lines (the robot width), $L_{q_1 k_1}$ and $L_{q_2 k_2}$ are the lines that belong to the beam with the unified notation $L_{\mathrm{qk}} \in \left\{L_{q_1 k_1}, L_{q_2 k_2}\right\}$ for both axes. Figure 63 illustrates an example of lines in the discrete domain.

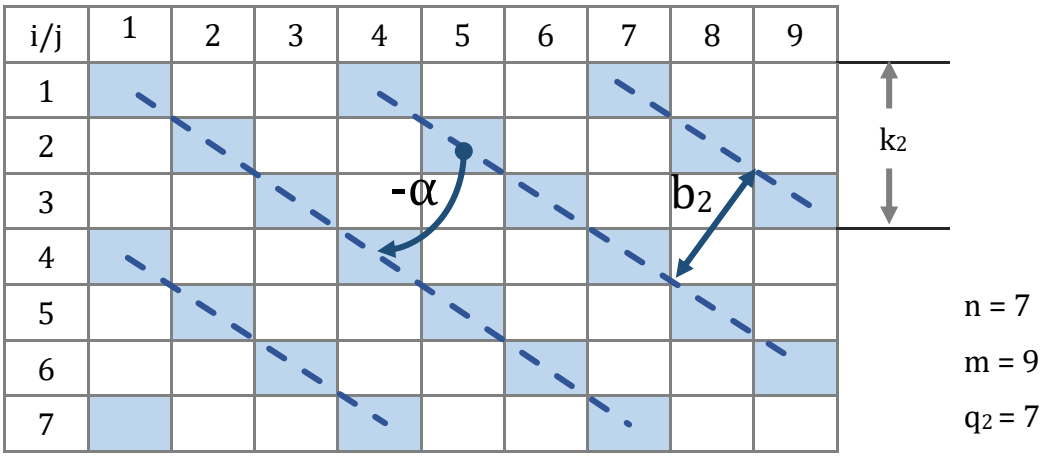


Figure 63 - Illustration of lines in the discrete domain

Each line can be associated with a matrix $\Lambda^{\mathrm{qk}} = [\Lambda_{\mathrm{ij}}^{\mathrm{qk}}]_{i=1\ldots n, j=1\ldots m} \in \Re^{n\times m}$ with the elements $\Lambda_{\mathrm{ij}}^{\mathrm{qk}}$

$$\Lambda_{\mathrm{ij}}^{\mathrm{qk}} = \begin{cases} 1, & \text{if } (i,j) = \left(L_{q_1 k_1}, j\right) \wedge (i,j) = \left(i, L_{q_2 k_2}\right), \\ 0, & \text{otherwise.} \end{cases} \tag{68}$$

Three solutions for the computation of the main lines are proposed in this thesis. The first two solutions generate a beam of lines and define the main segments as the intersection between the map and the beam of lines, compute the segments length and compute (or approximate) the path length in terms of the sum of these lengths. An iterative process is conduced to compute the slope of the beam of lines, which generates a set of path lengths. The result, represented the main segments, is related to the minimum length path as the solution to the optimization problem

$$q^* = \underset{q\in D_q}{\operatorname{argmin}} \sum_{k\in D_k} \Gamma\left(\Lambda^{\mathrm{qk}}, M\right), \tag{69}$$

where q^* gives the optimum slope in the discrete domain, D_q is the discrete domain of slope, D_k is the intercept domain, the general notation $\Gamma(\Lambda^{\mathrm{qk}}, M)$ is the general notation for the path length:

$$\Gamma\left(\Lambda^{\mathrm{qk}}, M\right) = \lambda \sum_{i=1}^{n} \sum_{j=1}^{m} p_{\mathrm{ij}}, \quad p_{\mathrm{ij}} = \begin{cases} 0, & \text{if } M_{\mathrm{ij}} = 1, \\ 1, & \text{if } \Lambda_{\mathrm{ij}}^{\mathrm{qk}} = 1, \end{cases} \tag{70}$$

where the general notation $\lambda \in \{\lambda_1, \lambda_2\}$ is used for the distance between the points calculated as

$$\lambda_1 = \sqrt{\frac{(m+1)^2+q_1^2}{m+1}},$$
$$\lambda_2 = \sqrt{\frac{(n+1)^2+q_2^2}{n+1}}. \tag{71}$$

3.1.1 Solution "A" for construction of the main lines

Solution "A" is based on the parallel lines rotation, this solution consists of the following steps:

1.1 The lines expressed in (66) which depend on q and k, are generated.

1.2 The matrices Λ^{qk} with the elements $\Lambda_{\mathrm{ij}}^{\mathrm{qk}}$ expressed in equation (68) are generated.

1.3 The path length $\Gamma(\Lambda^{\mathrm{qk}}, M)$ is computed according to (70).

1.4 The objective function in (69) is computed in terms of the sum $\sum_{k\in D_k}\Gamma(\Lambda^{\mathrm{qk}}, M)$ for $q = \text{const}$ and variable $k,\ k \in D_k$.

1.5 The optimization problem defined in (69) is solved considering that the objective function in the right-hand term of (68) depends on the variable $q,\ q \in D_q$, and the solution to this optimization problem, i.e., the variable that gives the minimum path length, is q^*.

The first two solutions differ by the slope domain and by the map definition. The first solution preserves the initial map and defines a continuous domain of slope $D = [0, \pi]$ in order to include all possible orientations of the beam of parallel lines.

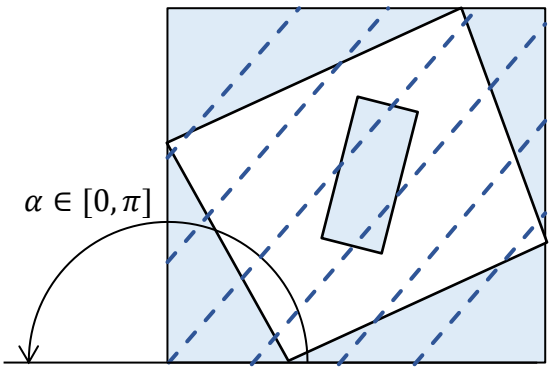


Figure 64 - Illustration of solution "A"

Figure 64 illustrates an example of beam of parallel lines used in the first solution ($D = [0, \pi]$).

3.1.2 Solution "B" for construction of the main

Solution "B" is also based on parallel lines rotation. Tis solution defines a new map using a composition of the initial map and uses a smaller domain of slope, i.e. $D = [0, \pi/4]$. Figure 65 exemplifies a beam of parallel lines used in the second solution.

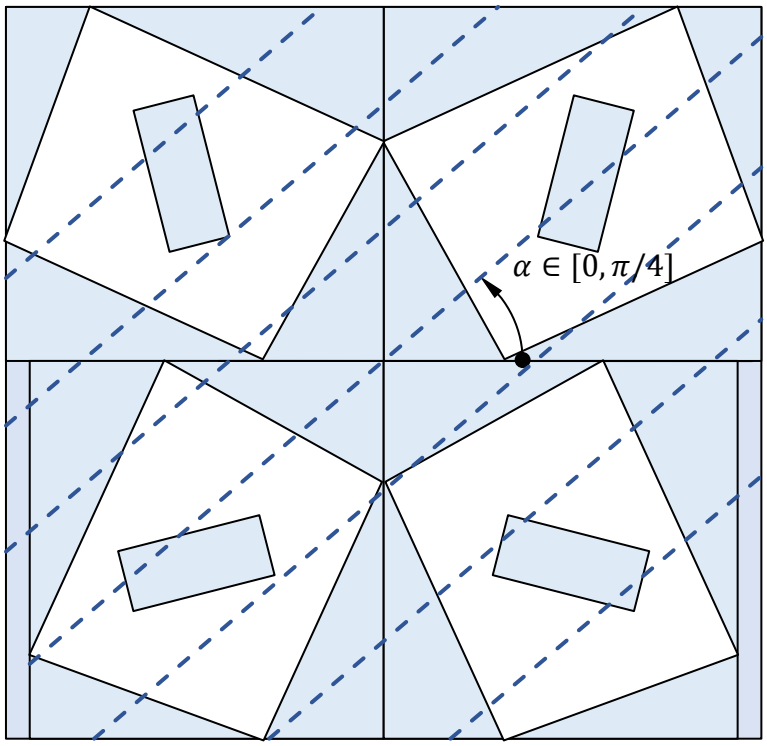


Figure 65 - Illustration of solution "B"

Solution "B" is based on the generation of a new map by the union of four maps that are rotated as shown in Figure 65. These four maps correspond to the four quadrants I, II, III and IV, obtained as follows.

The map in the quadrant I is $M_I = [M_{I,\mathrm{ij}}]_{i=1\ldots n, j=1\ldots m} \in \Re^{n\times m}$, with the map matrix elements

$$M_{I,\mathrm{ij}} = M_{\mathrm{ij}}, i = 1\ldots n, j = 1\ldots m. \quad (72)$$

The map in the quadrant II is $M_{\mathrm{II}} = [M_{\mathrm{II},\mathrm{ij}}]_{i=1\ldots n, j=1\ldots m} \in \Re^{n\times m}$, with the map matrix elements

$$M_{II,\mathrm{ij}} = M_{\mathrm{im}-j+1}, i = 1\ldots n, j = 1\ldots m. \quad (73)$$

The map in the quadrant III is M_{III}, obtained in terms of the composition

$$M_{\text{III}} = [P|M^T] \in \Re^{m\times m},\ P = [P_{\text{ij}}]_{i=1...m,j=1...m-n},\ P_{\text{ij}} = 1, \tag{74}$$

where the subscript T indicates matrix transposition.

The map in the quadrant IV is $M_{\text{IV}} = [M_{\text{IV,ij}}]_{i=1...n,j=1...m} \in \Re^{n\times m}$, with the map matrix elements

$$M_{I,\text{ij}} = M_{\text{in}-j+1}, i = 1...n, j = 1...m. \tag{75}$$

Solution "B" consists of the following steps:

2.1. The map matrix in the four quadrants is computed using (72) to (75).

2.2.-2.6. These are the steps 1.1 to 1.5 in the first solution.

3.1.3 Solution "C" for construction of the main lines

Solution "C" approximates the main lines with the map axis, which is inspired from the properties specific to mechanical inertia. The map axis slope is obtained in terms of

$$\alpha = \frac{1}{2}\tan^{-1}\left(\frac{2I_{\text{xy}}}{I_y - I_x}\right), \tag{76}$$

where the following center of gravity-type relationships are employed:

$$I_{\text{xy}} = \sum_{i=1}^{n}\sum_{j=1}^{m} i_c j_c M_{\text{ij}},\ I_x = \sum_{i=1}^{n}\sum_{j=1}^{m} j_c^2 M_{\text{ij}},\ I_y = \sum_{i=1}^{n}\sum_{j=1}^{m} i_c^2 M_{\text{ij}},$$

$$x = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} \text{jM}_{\text{ij}}}{\sum_{i=1}^{n}\sum_{j=1}^{m} M_{\text{ij}}},\ y = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} \text{iM}_{\text{ij}}}{\sum_{i=1}^{n}\sum_{j=1}^{m} M_{\text{ij}}}, \tag{77}$$

$$i_c = i - y,\ j_c = j - x,\ i = 1...n,\ j = 1...m$$

and M_{ij} are the elements of the map matrix M defined in (65). The beam of parallel lines is next computed using (66). An example of application of the third solution is given in Figure 66.

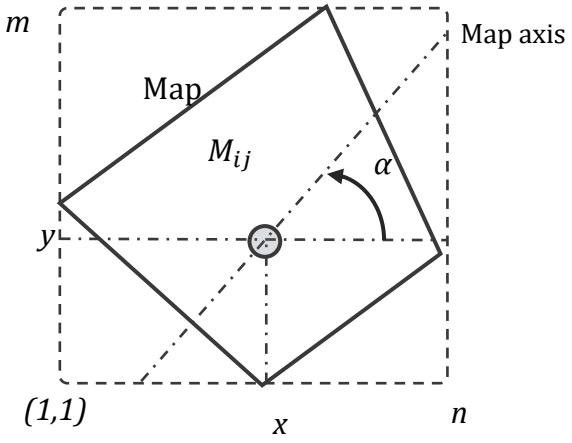


Figure 66 - Illustration of solution "C"

Solution "C" consists of the following steps:

3.1. The map axis slope is computed using (76) and (77).

3.2. - 3.6. These are the steps 1.1 to 1.5 in the first solution.

3.1.4 Construction of the auxiliary segments

In the previous section three solution is presented (A, B, C) and all of the solution provides an organised set of main lines. The lines order has been defined in the process of definition of the main lines using the intercept of each line expressed in (66). The definition of the main segments splits the lines in several segments producing a list of segments ordered (in their turn) using a left to right convention. This means that each segment has two kinds of neighborhoods, i.e., the segments that belong to the lists of neighbor main lines and the segments that do not belong to the same list. The second type of segments is eluded because of the obstacles between these segments. An ordered beam of segments is illustrated in Figure 67.

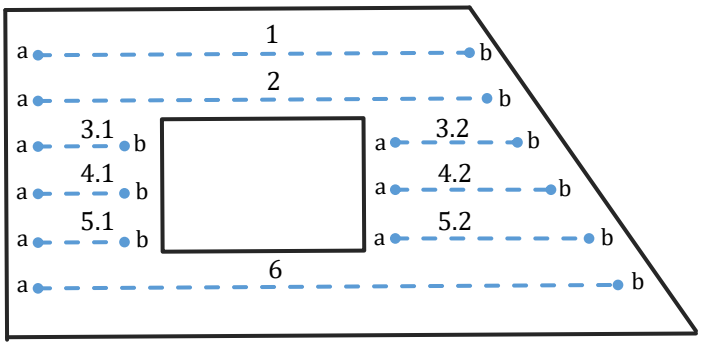


Figure 67 – Illustration of ordered beam of segments

The generated main lines and the line graph object have some unique features which the algorithm takes advantage of. The problem of visiting all the nodes can be considered as a classic travelling salesman problem (TSP) [68]. The classic TSP approach assumes that all the nodes are connected to each other and all of them needs to be visited once. In our case there is not necessary a limitation to visit a node only once, so it is related but not the classic travelling salesman problem. The generated connectivity graph is a unique one and it consists of three types of connections. The first is where the nodes are connected to each other as a single line, which means the in-degree and the out-degree of each node is one. The second type is where the main lines splits which means the out-degree is greater than one. The third type is the opposite where the nodes merge. In general, considering the mentioned representation usually the degree of a node is a small number, usually under 10, in contrast with some TSP datasets can contain nodes with a degree of 100 or more. The classic TSP approach requires to visit a node only once which is not a requirement in our case. For a TSP problem numerous [68] solutions are available which are either exact or heuristic algorithms. Considering the mentioned uniqueness, a simple heuristic greedy algorithm is used to visit all the nodes which are basically start and end points of the main lines.

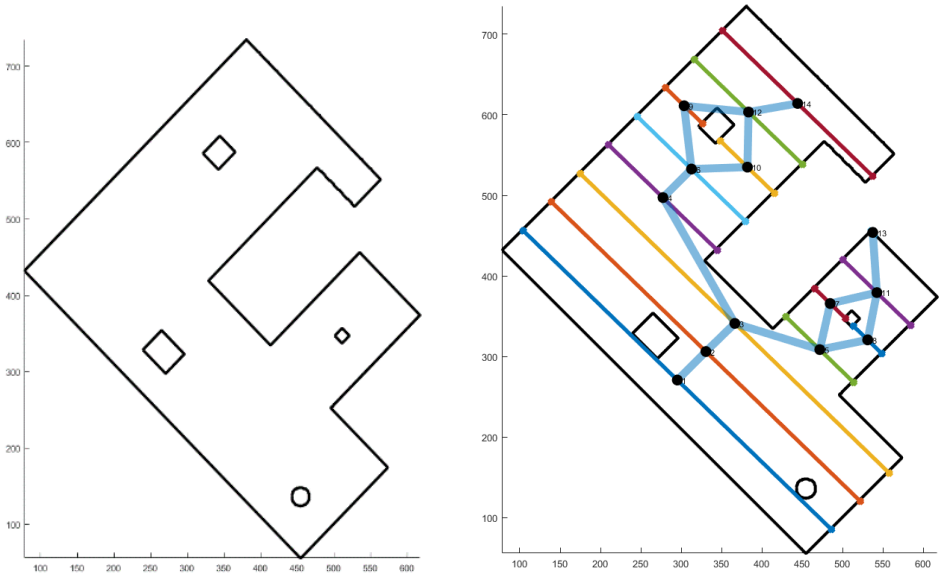


Figure 68 - Illustration of the method to connect the main lines

All segment starts with a and ends with b. The nodes G graph is determined between the a and the b point:

$$G = (V, E), \tag{78}$$

Where the set of nodes are denoted with V

$$V = \{i.j.k \vee i = 1...n_s,\ j = 1...n_i,\ k = a \text{ as well } b\}, \tag{79}$$

n_s is the number of main segments, and n_i is the number segments generated from i^{th} main line, and the set of edges is E:

$$E(i.j.k, p.r.l) = \begin{cases} 1 & \text{if } p = i + 1 \text{ or } i.j = p.r, \\ 0 & \text{else.} \end{cases} \tag{80}$$

The equation (80) shows weather, an edge between nodes exists. The edge exists if $E(i.j.k, p.r.l) = 1$, this means that one of the conditions need to be true: either if $p = i + 1$, meaning that the segments are between two successive lines (the line is not skipped) or, if $i.j = p.r$, meaning between the points of the same segment. The approach basically handles the segments as nodes with one condition, if the entrance happened on one point of the segment, the depart should be on the other point. In order to avoid this complexity, a heuristic is proposed to connect only neighbor segments. The graph related to the segments illustrated in Figure 69.

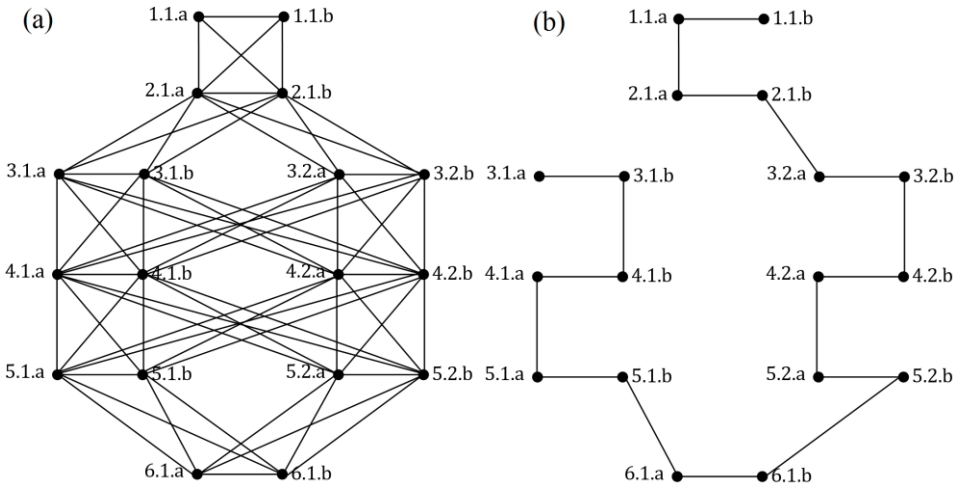
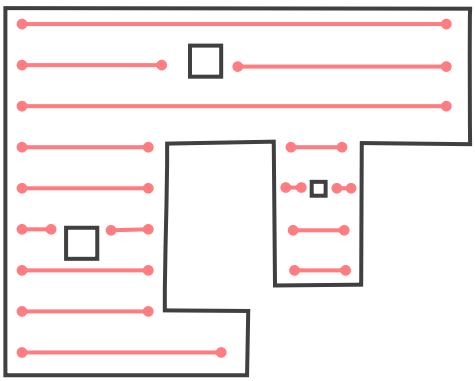


Figure 69 - All possible ways to connect the nodes (a) and a possible traversal solution (b)

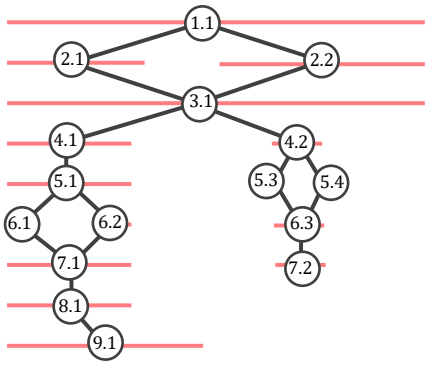
Each edge of the graph is associated to a cost function. The simplest cost function definition is the distance between the nodes. If the nodes can be linked with a straight line, the computation of the distance is simple. Contrarily, if obstacles interfere, a trajectory between nodes must be defined. The graph can be further simplified by trimming these edges in order to fulfil the objective to minimize the cost function, i.e., to minimize the path length.

The problem of visiting all segments (nodes) is related to the TSP. In contrast to the classical TSP approach, which assumes that all of them need to be visited once, the ISOC approach proposed in this paper does not have this constraint. The only constraint imposed here is that after a node is reached it is mandatory to visit the second node of the segment so the covering of the main segments is ensured. Figure 69 points out a possible solution that starts with the node 1.1.b and ends with the node 3.1.a.

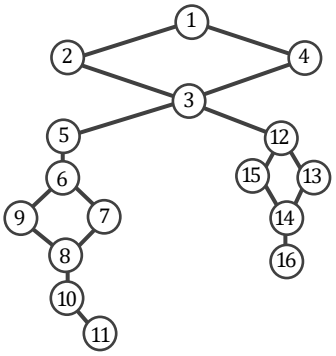
Concluding, the auxiliary segments are the lines (or trajectories) that connect the main segments and ensure a minimum path length.



(a)



(b)



(c)

Figure 70 - The main segments (a), the generated traversal graph (b) and a possible traversal created by the greedy algorithm (c)

The aim of the greedy graph traversal algorithm is to *specify the line visiting* sequence has the following features. It is obvious that every main line needs to be visited, and every line has two sub-nodes. One sub-node will be the start node the other the end, so this can be considered as one of a constrains regarding the visiting order. In other words, the way inside a main line cannot be the same as the way outside the main line. With this constrain it is guaranteed that the map will be covered entirely. The line graph object which connects all lines together (`lineGraph`) is generated from the opposite direction as the

main line orientation. This means if a slice of map is covered with only one main line, the slice before and after is connected to only this line. In other words, every lines in a specific slice can be only connected to the lines of slices before and after but not to any other slice of lines. The connection between these lines are specified how these lines overlay. An overlay in this context means that a line is available from its neighbour's slice. After the `lineGraph` object is generated the mentioned greedy algorithm visits all the nodes. The greediness of this algorithm is composed from two things. Firstly, if the node has only one out-degree, the algorithm chooses it. Secondly, if the node has more than one out-degree, so basically where the node splits, it continuously chooses the first node, if not already visited. After it reaches a merge node it goes back, until it finds an already visited node. Broadly speaking this is how the algorithm works, which greedy behaviour may not optimal but in this unique featured graph it can perform well.

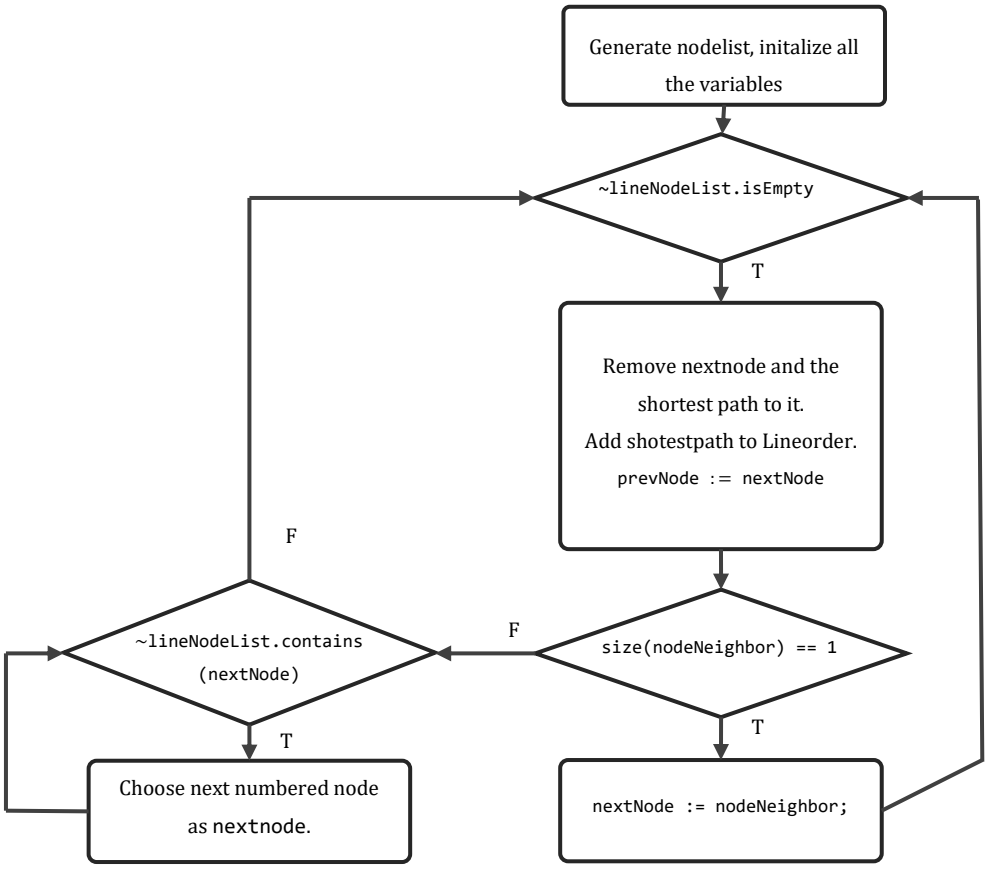


Figure 71 - Flowchart of the greedy algorithm

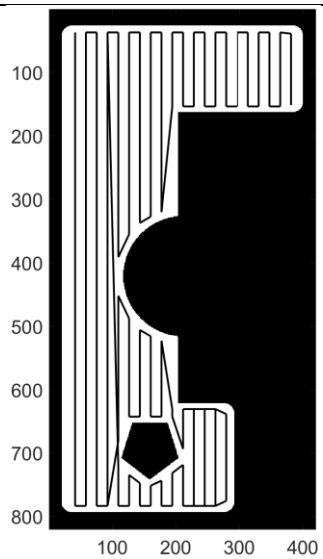
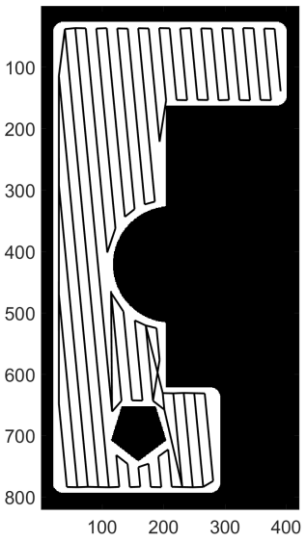
In more details the greedy graph traversal algorithm works as follows. The `lineGraph` object is considered as an undirected graph. The G undirected graph is a set of objects -

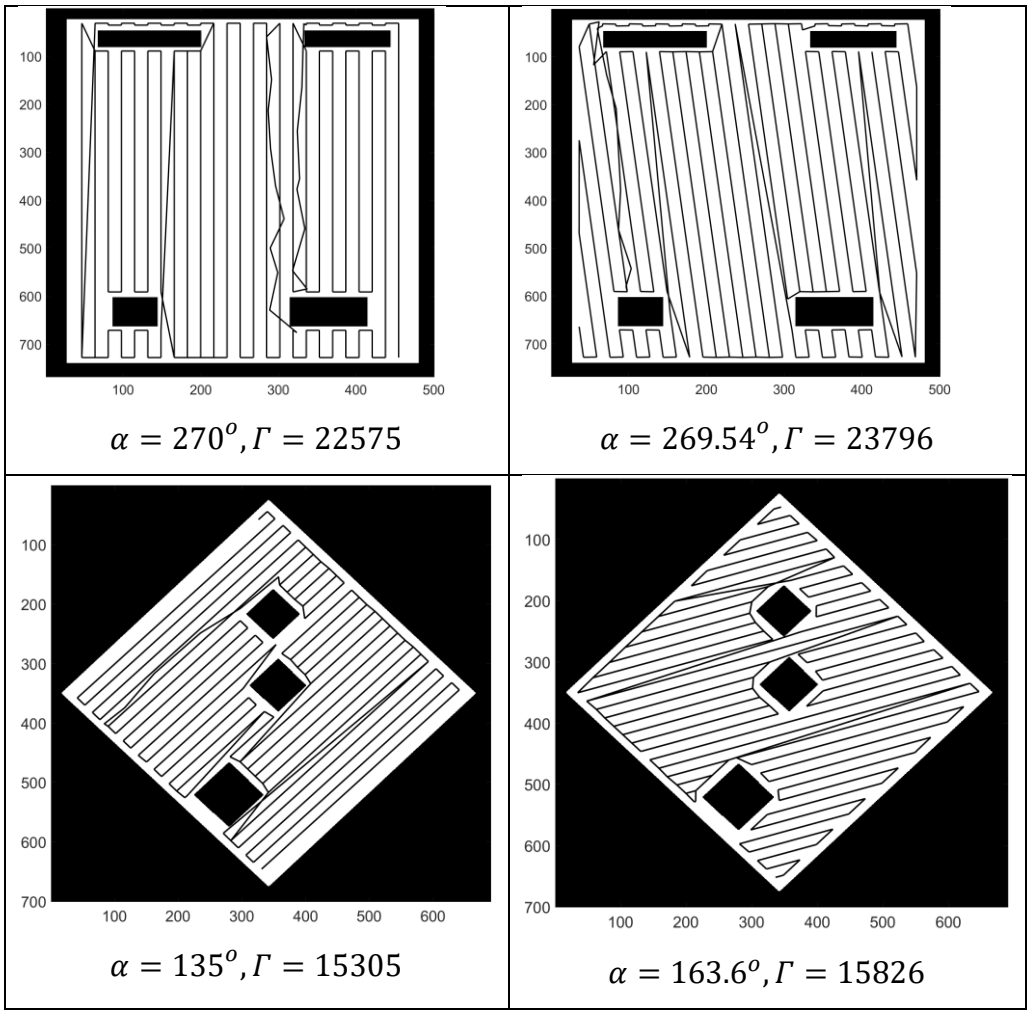
edges and nodes - that are connected together, where all the edges are bidirectional. This can be formulized as $G = (E, N)$ where $N \neq \emptyset$ and $N = \{n_1, n_2, \dots\}$ and $E = \{e_1, e_2, \dots\}$ thus $E \subseteq N$. All the N nodes of the `lineGraph` object should be included into a dynamic list datatype called the `lineNodeList`. In MATLAB or Java this can be an ArrayList, in C# a List or in C a Linked List. The main loop of the algorithm removes all the elements from the list, so it terminates when the list is empty. The path initializes with the first element of the graph (n_1). After that it chooses the obvious way, if only one neighbour is present. If there are multiple neighbours, then it first goes down until the sub-branch is covered than it comes back and repeats the mentioned two cases. The flowchart of algorithm is visible in Figure 71.

3.1.5 Conclusion and evaluation

This section validates the ISOC approaches presented in the previous section by simulations and experiments conducted on mobile robots. The validation by simulation includes a comparison between the proposed approaches, and of the proposed approaches with another well-known approach discussed in Section 1, namely the BCDC approach. The validation by experiments is focused on a Khepera mobile robot.

Table 2 - Comparison between the proposed approaches

Results using solution "A" and "B"	Results using solution "C"
 $\alpha = 90^o, \Gamma = 14443$	 $\alpha = 84.52^o, \Gamma = 13684$



Three types of maps have been used to validate the ISOC algorithm presented in the previous section: a *simulated map* obtained from V-REP, a *real-life measurement* map, and an *artificially generated* map. The maps developed in V-REP and imported to MATLAB are called simulation maps. The real-world measurement maps are available datasets, which we have been downloaded from [69]. These maps represent the third floor common area of the MIT Stata Center (Dreyfoos Center) [69]. The artificially generated maps have been obtained by either hand drawing or randomly using MATLAB. These maps are illustrated in Fig. 9 as follows: three artificial maps in Figure 72 (a), (b) and (d), and a real-world map, i.e. the test bench taken from our laboratory snapshot in Figure 72 (c).

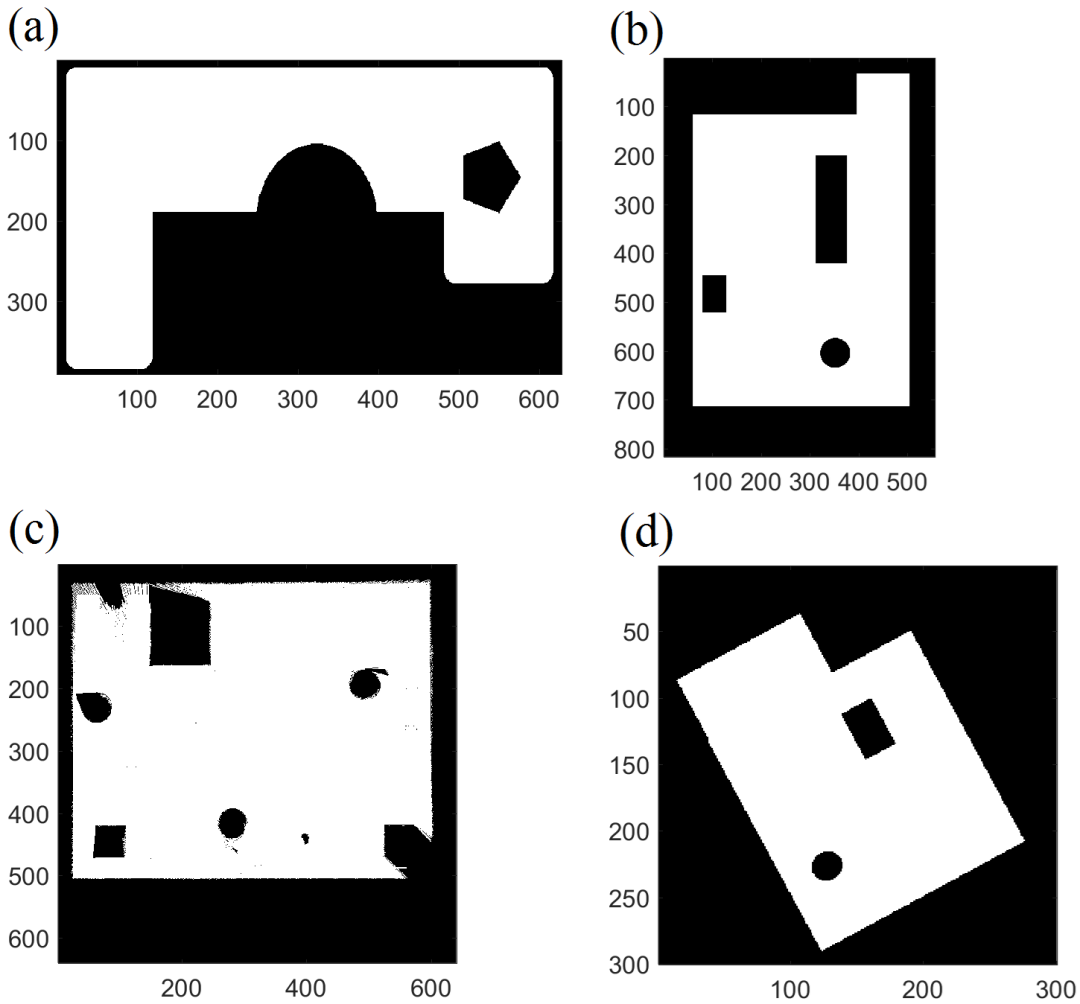


Figure 72 - Real-world (c) and artificially generated maps (a, b, d)

The proposed approaches are next compared to the BCDC approach, and the results are presented in Table 2. The comparison shows that the ISOC algorithm is always slower than the BCDC algorithm, but most of the time it generates a shorter path. This confirms the results that have been foreseen at the very beginning of the creation of the ISOC approaches and algorithm: ISOC deals with complex maps better, meaning that it generates shorter paths, but this is reflected in its increased computational complexity. The comparison was done on a computer with i7 3.7 Ghz processor and 16 GB RAM. Form point of computational complexity the three proposed solutions has the following characteristics. Solution "A" is the most time consuming, solution "B" is less time consuming but it uses more memory. Solution "A" and "B" always generates the same results. Solution "C" is usually the fastest but it doesn't guarantee the best orientation.

Map size (pixel)	Number of holes	Occupied ratio (%)	BCDC-based path length (mm)	ISOC-based path length (mm)	BCDC-based computation time (s)	ISOC-based computation time (s)	Image
435600	1	73.04	9984.41	9625.15	15.65	32.17	Figure 72 (a)
453696	3	55.77	21392.96	18449.71	10.79	45.65	Figure 72 (b)
409600	5	60.06	27214.68	22278.05	14.82	96.97	Figure 72 (c)
90000	2	39.23	2879.35	2880.04	10.54	49.48	Figure 72 (d)

A Khepera III differential drive robot has been used in the experiments. The map has been previously known, and after the path commutation, an open loop control was applied for the robot. The first phase of the experiments consists of a simulation, and the result is given in Figure 73.

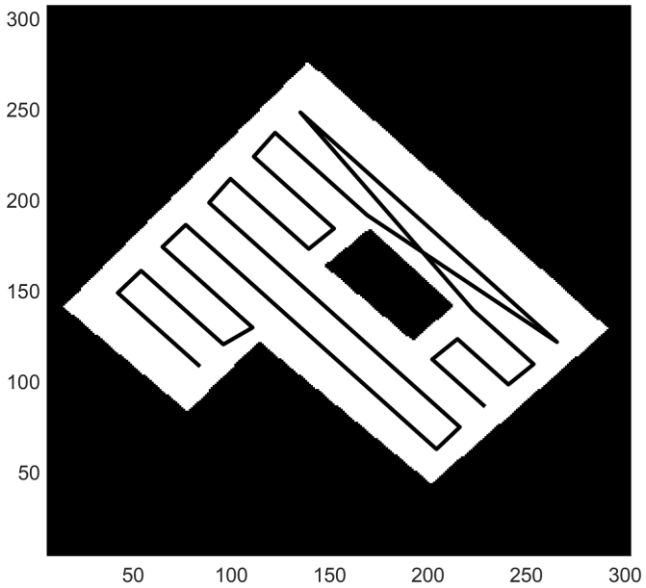


Figure 73 - Simulation result of the known map and the robot path

A map was evolved in the second phase and highlighted with yellow marker in order to visualize the real-world experiments as shown in Figure 74 (a). The robot path was measured with a camera applied above the robot path. This camera took photos in approximately equivalent periods of time. Figure 74 (b) shows a merge of several images

that were taken during the experiments, and suggestively illustrates that the robot covers the path. The entire commented source code is available online.

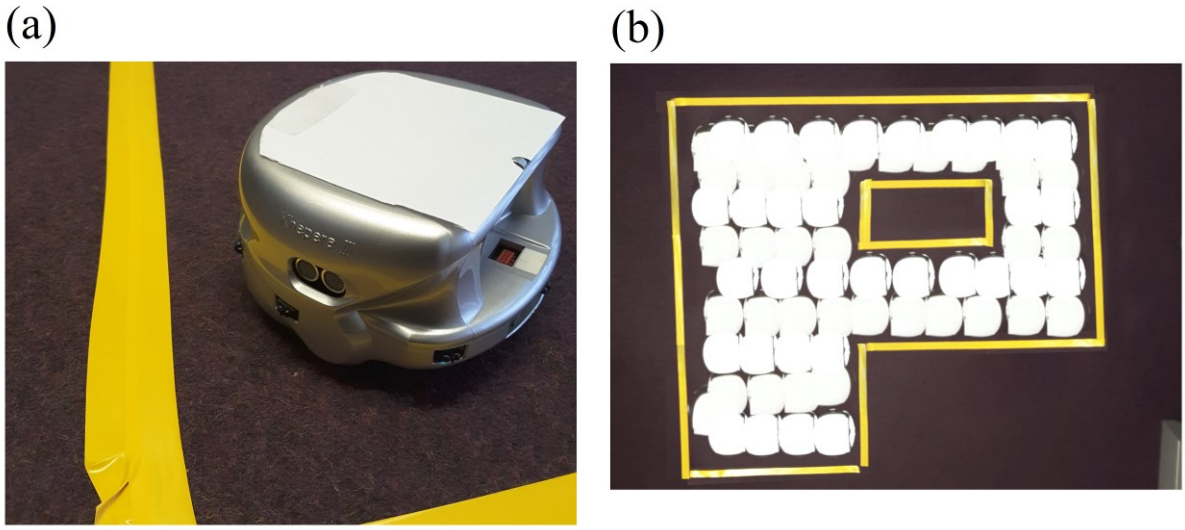


Figure 74 - Image that shows the Khepera robot (a) and merged images illustrating the robot path (b)

Figure 74 visualizes the measurement, on the left side the Khepera robot is visible on the right side the traversal trajectory is visualized as merged images, which were taken in different time.

3.2. Thesis 2.

I created a kinematic trajectory following approach, called the multiple goal pursuit. One of the benefits of this algorithm is the more precise tunability in curved areas; and the more human-like reasoning involved. The human-like thinking comes from tracking and following more goal points on the road simultaneously. I investigated more versions of this algorithm; I also created the windowed version which divides the trajectory into more working sets and fits the curves to those sets.
References: [H26], [H27], [H28]

The motivation of the research behind this thesis was to construct a simple, but flexible kinematic trajectory following approach. In addition, I wanted to implement a *more human-like thinking*, meaning to track and follow more goal points on the road simultaneously. As a driver we naturally take multiple considerations into account, without even considering it consciously. E.g. not only lane keeping is an important task but the cyclist in our lane also influences our planning. The proposed approach is encouraged by the pure-pursuit algorithm. Pure-pursuit algorithm is a popular trajectory tracking algorithm, widely used in mobile robotics and vehicular control for numerous reasons. The operation is simple and straightforward as it depends only on the kinematic model of the target mechanical system. The algorithm can be tuned by choosing look-ahead distance of points of the reference trajectory. In this thesis, I propose a low-computational complexity therefore fast trajectory following approach which was inspired by the pure-pursuit algorithm.

3.2.1 Description of multiple goal pursuit algorithm

The multiple goal pursuit is based on a simple assumption to follow more than one selected goal point at once and to select the best fitting (see Figure 75). The realization is based on the finite number of discrete possible curves. These curves are implicitly determined by the wheel angle, which is constrained by the kinematic properties of the vehicle. In my realization first instead of using an optimizer I used a finite number of discrete possible curves and defined the best fit from this set of possibilities. Later I also implemented a simple optimizer for the task.

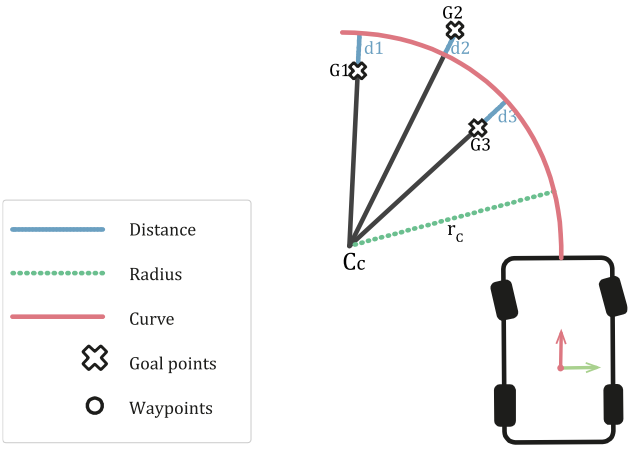


Figure 75 - High-level overview of the proposed multiple goal points-based pure-pursuit algorithm

The algorithm extends Step 4 of the original pure-pursuit algorithm (described in the Trajectory and path following approaches section). Assume that the reference trajectory is given, with a set of T geometric points, and the P position and θ orientation of the vehicle. Define $G \subseteq T$ as the set of selected goal points, with a preset length $N = |G|$. Goal points are selected starting from the closest reference point to the vehicles current position (index denoted as j), shifted with an additional preset $o \in \mathbb{Z}$ offset:

$$G = \{G_1, G_2, \dots G_N \mid G_k, (j + o) \leq k < (j + o + N)\} \tag{81}$$

Assume an angle α_i from a domain of possible angles $[\alpha_{min}, \alpha_{max}]$ (presumably the wheel angle limits). A sequence of curves can be calculated each with radius $\rho_i = -2/\alpha$, and a centre point C_i. A G_k goal point from set G to any C_i determines a line segment: the normalized difference of the length of this segment and ρ_i radius is the $d \in \mathbb{R}^+$ distance from the curve. The metric for selecting a good angle is the sum of this difference for each goal point. Summarizing the calculation of d_{sum} (equation 82) distance for the goal point set G:

$$d_{sum_i} = \sum_{k=0}^{N} \left| \left\| \overrightarrow{C_i G_k} \right\| - \rho_i \right| \tag{82}$$

Finally, the curve with the minimal d_{sum} is chosen. Figure 75 shows the steps of calculating the best angle given multiple selected goal points. This modification naturally introduces some issues. The most obvious one that at a given time there is no guaranteed curve that reaches all the selected multiple-goal points. Another example for issues is at the reaching the end of the reference trajectory, where there are physically no multiple goal to select. In the following I will go into the details why these issues emerge and how the algorithm overcomes these matters. As mentioned in the introduction involving multiple goal instead of one derives issues. The most important is that there is no definitive solution for the curve so a solution for that should be constructed. I created two different version for this particular problem. The first solution is a kind of greedy approach. In this case a discrete set of possible curves is created (similarly to Figure 77) and a simple loop iterates trough this set. The algorithm in this case is a simple minimum search, the minimal Euclidian distance between the given goal points and the curve determines the solution.

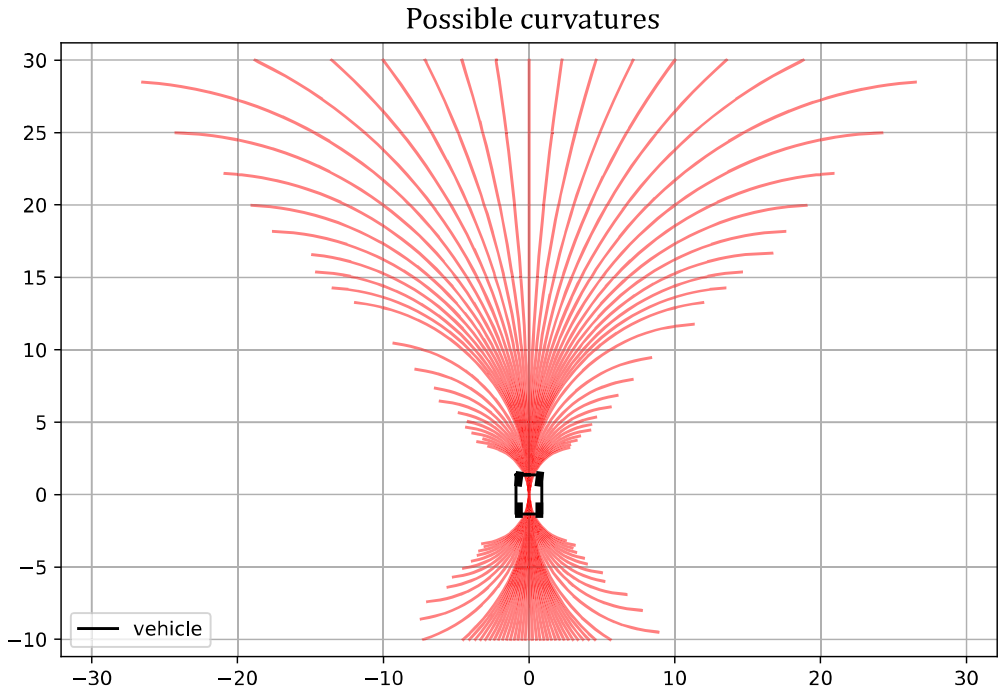


Figure 76 - Visualization of the possible curves

On the other hand, this solution is rather a trade-off between the resolution of the curve set and the speed of the algorithm. The more precise result emerges the higher the resolution is, but of course this requires more computation and more time. For this

reason, an optimization algorithm is created too. This algorithm iteratively approximates the best solution and will be described in the next section. The next pseudo-algorithm may clarify the working principle of this greedy algorithm, which finds the most suitable angle.

```
1: function MultiAngleGoalSelectBestAngle(T: trajectory, N: number of goal
   points, o: offset of selection, P: position of the vehicle, θ: yaw of the
   vehicle, [α_min, α_max]: angle domain)
2:  k := index of closest ref. point of T to P;
3:  G := T[o+k:o+k+N];
4:  distpairs ← Empty list of 2-tuples
5:  for all α_i in [α_min, α_max] do
6:    p_i := -2/α_i
7:    C_i := P-R(θ)r
8:    d_sum := 0
9:    for all G_j ∈ G do
10:     d_sum := d_sum + distance(C_i,G_j)- p_i
11:   end for
12:   push (d_sum, α_i) into distpairs
13: end for
14: α_best ← select from distpairs α_i where d_sum is min
15: return α_best
16: end function
```

3.2.2 Description of windowed multiple goal algorithm

In this section a variant of the original multiple goal pure pursuit will be described, i.e. which is based on *windowing*. This algorithm is rather not a classic trajectory follower in terms of slightly redesigning, replanning the original trajectory. With this method a similar to the original trajectory is created, but the slight modification makes it more suitable to the vehicle or robot to follow. The design strategy consists of finding the optimum trajectory which approximates a desired trajectory defined by a set of chosen points. This means that the mentioned set is an initial data and a minimum problem must be defined.

From the multitude of possibilities, the minimum problem will be defined starting with the following hypothesis:

1. During the approximation the steering angle γ is constant;
2. Changing the steering angle γ is an instantaneous process.

If the previous hypotheses are correlated, we can conclude that the initial set of points is approximated by a trajectory composed from circle arcs. The transition from one arc to another is instantaneous. For a particular circle arc a subset (named here *the working set*) is selected from the initial set.

The following decisions are subjects of debates:

- The number of points included in the working set;
- The number of common points which belong to different working sets;
- The car position when the next working set is defined.

Figure 77 illustrates the mentioned observations.

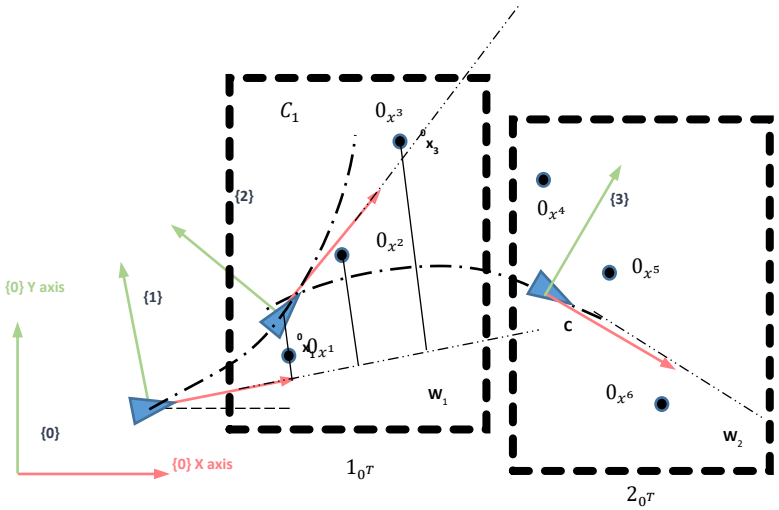


Figure 77 - The windowed goal pursuit strategy

In Figure 77. a possible strategy is illustrated. The figures contain two referential systems. The first 0 is the global (fix) referential system the second 1, 2, 3 is the mobile referential system attached to the vehicle. The blue triangle is the abstraction of the vehicle. Two windows are illustrated $w_{1,2}$. Each of them contains a *working set* of three points.

The trajectory is a composition of two circle arcs C_1 and C_2. The first (C_1) is defined in 1 using the window w_1 i.e. the points ${}^0x_1$, ${}^0x_2$, ${}^0x_3$. The second (C_2) is defined in 2 and use the window w_2 i.e. the points ${}^0x_4$, ${}^0x_5$, ${}^0x_6$. Each of the mentioned point is defined in the 0 referential frame.

Using m windows and n points for each window, the mathematical definition of the problem is the following:

$$R_j = \arg\left(min \sum_{i=1}^{m} e_i\right) \tag{83}$$

where:

- R_j is the radius of the arc C_j;
- e_i is the distance from point i to the arc C_j;
- j is the current window number $j = 1, \dots, m$;
- i is the current point number which belong to the widow j.

The distance from point $\boldsymbol{x}_i$ to C_j is also a problem worth to discuss. Figure 78. illustrates two possibilities: a.) considers only one coordinate of the current point. By contrary in Figure 78 b.) the shortest distance is considered.

For the first choice:

$$e_i = y_i - C_j(x_i) \tag{84}$$

where:

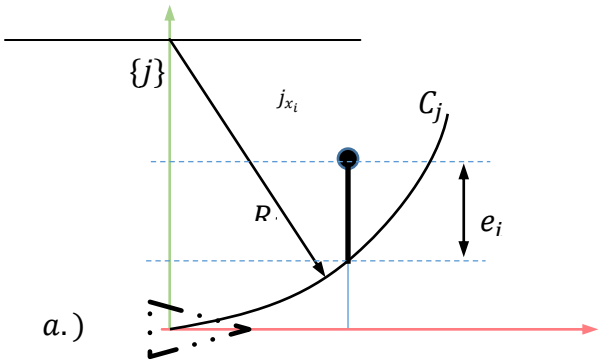
$$\begin{bmatrix} {}^{j}x_i \\ {}^{j}y_i \\ 1 \end{bmatrix} = {}^{j}_{0}T\,{}^{\mathbf{0}}\mathbf{x_i} = \begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} {}^{\mathbf{0}}\mathbf{x_i}; \tag{85}$$

θ is the orientation of the j referential system; x_j, y_j are the coordinate of the origin for j referential system.

If we assume this definition, we will recognize that this is a quadratic regression problem which has the analytical solution.

$$\frac{1}{R} = \kappa = (\mathbf{X^t X})^{-1}(\mathbf{X^t Y}) \tag{86}$$

where:



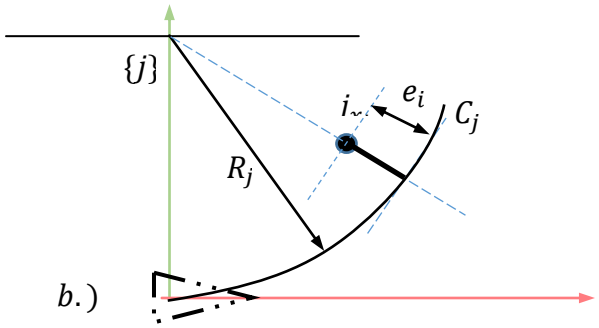


Figure 78 - Possible ways of computing the distances

$$\mathbf{X} = \begin{bmatrix} x_1^2 + y_1^2 \\ \dots \\ x_n^2 + y_n^2 \end{bmatrix} \tag{87}$$

$$\mathbf{Y} = \begin{bmatrix} 2y_1 \\ \dots \\ 2y_n \end{bmatrix} \tag{88}$$

For the second choice:

$$e_i = \left| R - \left\| \begin{bmatrix} x_i \\ y_i - R \end{bmatrix} \right\| \right| = \left| R - \sqrt{x_i^2 + (y_i - R)^2} \right| \tag{89}$$

which has a numerical solution.

In the following a flowchart (Figure 79) is presented in order to understand the approach more in detail.

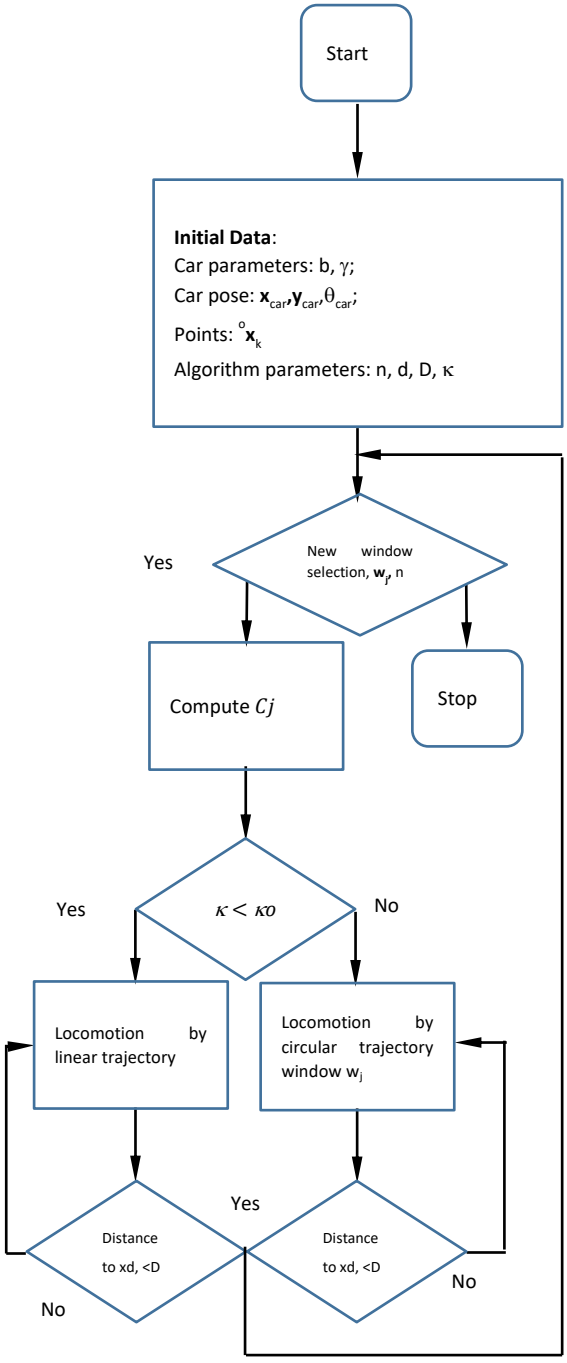


Figure 79 - Algorithm flowchart

The algorithm uses the vehicle parameters, pose, the trajectory points and the input parameters as an input. The vehicle parameters are: b wheelbase and γ steering angle, and the pose. The points: ${}^{o}\boldsymbol{x_k}$. The algorithm parameters are: n working set, d skyline and κ curve limit. This limit decides whether the locomotion is based on linear trajectory or on circular trajectory window. In every iteration a new window selection is calculated.

The simulation uses the previous algorithm for a kinematic model which is in accordance with the starting hypothesis i.e. the steering is instantaneous. In the first simulations, the following kinematic bicycle model is used:

$$\begin{cases} \dot{x} = v\cos(\theta) \\ \dot{y} = v\sin(\theta) \\ \dot{\theta} = \frac{v}{L}\tan(\gamma) \\ \gamma = ct \end{cases} \tag{90}$$

To analyse the results, we define the lateral approximation error.

- The lateral deviation error of the current position $e(k)$ is the distance between this position and the current segment of the desired trajectory,
- The total deviation error e_T is the sum of the error during the all simulation from 1 to N,
- The relative deviation error e_r is the ratio between the total error and the total number of simulated points.

$$e(k) = distance\left(\boldsymbol{x}(k), \overline{{}^{0}\mathbf{x}_{\boldsymbol{u}}{}^{0}\mathbf{x}_{\boldsymbol{u+1}}}\right) = \left|\overline{\mathbf{x}(\mathrm{k})\mathbf{x}_{\boldsymbol{u+1}}} - \mathbf{x}_{\boldsymbol{u}}\widehat{\mathbf{x}_{\boldsymbol{u+1}}}\left(\overline{\mathbf{x}(\mathrm{k})\mathbf{x}_{\boldsymbol{u+1}}} \cdot \overline{\mathbf{x}_{\boldsymbol{u}}\mathbf{x}_{\boldsymbol{u+1}}}\right)\right| \tag{91}$$

$$e_T = \sum_{k=1}^{N} e(k) \tag{92}$$

$$e_r = \frac{e_T}{N} \tag{93}$$

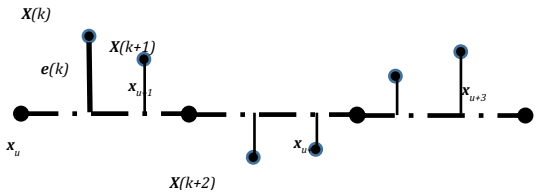


Figure 80 - Lateral deviation definition

These errors depend on number of the points included in the working set (n) named the skyline and on the position of the decision point (d). For the desired trajectory we have

made a study relative to this dependency and we obtained the results relative error value for each case (n, d).

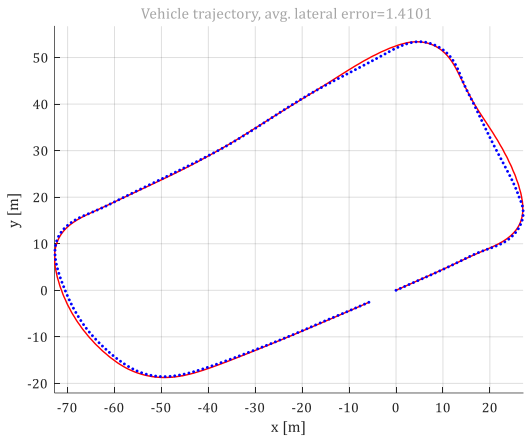


Figure 81 - Simulation result, the blue dots are the waypoints, the red line is the trajectory generated by the model

We also prepared with the simulation of the non-instantaneous steering. In this case the kinematic bicycle model can be modified as follows:

$$\begin{cases} \dot{x} = vcos(\theta) \\ \dot{y} = vsin(\theta) \\ \dot{\theta} = \frac{v}{L}\tan(\gamma) \\ \gamma = \gamma_0 + \dot{\gamma}t \\ \dot{\gamma} = ct \end{cases} \tag{94}$$

The steering angle is a linear function where the steering velocity is constant during the steering process. The robot trajectory is no longer an arc of circle but a combination of two curves. First one is a clothoid (also known as Euler spiral) when $\gamma = \gamma_0 + \dot{\gamma}t$ and an arc of circle when $\gamma = ct$. In this case the simulation result shows a lower quality of approximation. Because of this phenomenon we define the robot orientation angle error like the difference between the orientation angle in the hypothesis that the steering is instantaneously modified to the final value and the orientation in the hypothesis that the steering angle is continuously modified from the initial to the final value.

$$e_\theta = \frac{v}{L}\left(tan(\gamma_f)\Delta t - \int_0^{\Delta t} \tan(\gamma_0 + \dot{\gamma}\tau)d\tau\right) = \frac{v}{\dot{\gamma}L}\left(tan(\gamma_f)\Delta\gamma - log\left(\frac{\cos(\gamma_0)}{\cos(\gamma_f)}\right)\right) \tag{95}$$

where: $\gamma_0$is the initial steering angle;

γ_f is the final steering angle $\gamma_f = \text{atan}\,(L\kappa)$;

$\dot{\gamma}$ is the steering angle velocity;

Δt is the steering time;

It can be observed that (95) is in accordance with the intuitions: if the steering angle velocity increase the orientation error angle will decrease; by contrary the velocity is proportional with the error.

Now a strategy where a maximum orientation error e_{θ_max} is allowed is straightforward to construct. Because the maximum steering velocity $\dot{\gamma}$ is an actuator parameter (which is no subject of changes) equation (96) is proposed for the car velocity.

$$v_{new} = \frac{e_{\theta\_max}}{e_\theta} v_{old} \tag{96}$$

In conclusion the algorithm must be improved in order to offer better results. We think to the following possibilities:

- Refine the input set of points: interpolate new points;
- Change the vehicle velocity during the locomotion (96);
- Define the new windows by including in the new working set the points which succeed the decision point.

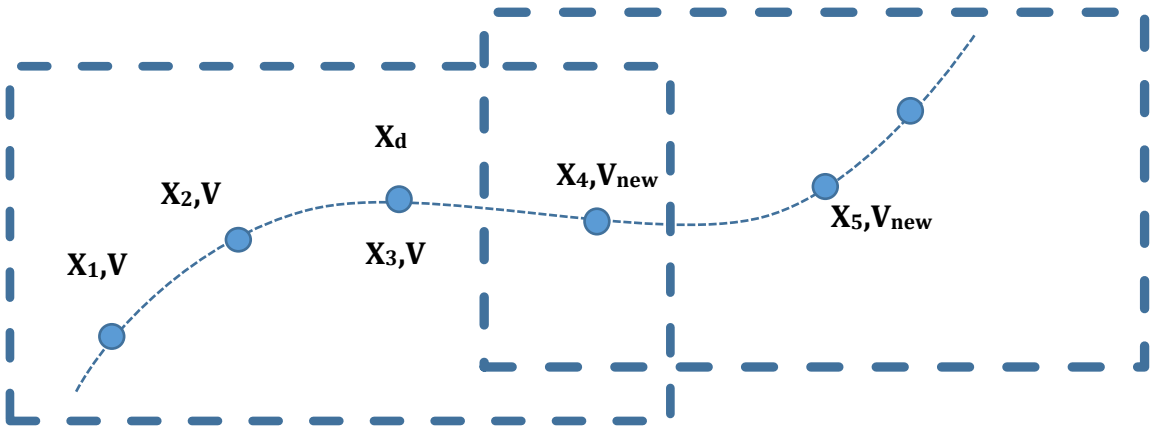


Figure 82 - The proposed new strategy

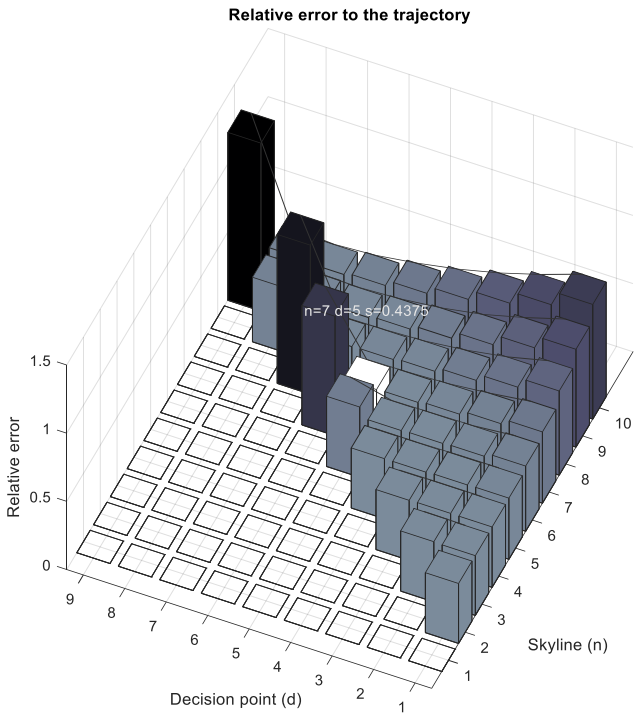


Figure 83 - The relative error depends on the skyline (n) and on the decision point (d)

Regarding an examined sample trajectory, a 3D graph is provided (Figure 83) which shows relative the dependency what we obtained as a results relative error value for each case (n, d). In this figure the x axis is the skyline (n), the y axis is the decision point (d) and the z axis is the relative error.

3.2.3 Metrics

In order to evaluate these algorithms two metrics were elaborated: the average of the lateral deviation, and maximum lateral deviation.

To validate and investigate the performance of the proposed algorithms, error metrics must be defined. Two basic metrics were used, the average lateral deviation and the maximum lateral deviation. It can be assumed that the reference trajectory is composed of $n > 2$ reference points and is defined at any t time step. In this case, the lateral deviation is the perpendicular distance between a line l and the position P of the vehicle.

Line l is determined by two L_1, L_2 points of the reference trajectory, a direction vector $\overrightarrow{L_U} = \overrightarrow{L_1 L_2}$. Using the general formula to calculate the perpendicular distance between point P and line l, the lateral deviation error is calculated as:

$$e_{lat}(P,(l)) = \frac{\left\|\overrightarrow{L_A P} \times \overrightarrow{L_u}\right\|}{\left\|\overrightarrow{L_u}\right\|} = \frac{\left\|\overrightarrow{L_2 L_1} \times \overrightarrow{P L_1}\right\|}{\left\|\overrightarrow{L_2 L_1}\right\|} \tag{97}$$

The algorithm is supposed to minimize the perpendicular distance during its operation. It can be validated with the average and the maximal lateral distance over a total of N_m measurement points. As a benchmark a simple average metric (98) can be used. I used the arithmetic mean of measured lateral distances (assuming P_i is the vehicle position at a given time):

$$e_{avg} = \frac{\sum_{lat}^{N_m} D_{lat}^{i}\,(P_i, \{L1, L2\})}{|N_m|} \tag{98}$$

Over this set of measurements, calculating the maximum lateral deviation (99) is also straightforward:

$$e_{max} = max\,\{\, D_{lat}^{i}\,(P, \{L1, L2\})\} \tag{99}$$

3.2.4 Test-environment used at real-world measurements

This section is devoted describing the automotive case study in which the implementation was used. The main task of this case study was to instrument an electric vehicle (Nissan Leaf, Figure 85) with autonomous functionalities. The prototype of this system performed satisfactorily on the opening of the ZalaZone vehicle industry proving ground. Figure 85 illustrates the vehicle in operation in Zala. The operational domain of this vehicle is limited to the proving ground and the university campus and is required to reach a relatively slow speed of 25 km/h. This vehicle is front-wheel driven, with a wheelbase of 2.7 meters and a track width of 1.77 meters. To ensure instantaneous localization of the vehicle we used a high accuracy RTK capable D-GPS sensor (KVH GEO-FOG 3D Dual), which was also used to capture reference trajectories. The software was based on OSRF Robot Operating System (ROS) [70]. The computing platform features first the Nvidia Jetson TX2, later the Nvidia Jetson AGX Xavier. The low-level control is realised on a National Instruments CompactRIO Controller: cRIO-9039 as the Real-time and FPGA

module, NI 9853 CAN module, NI 9403 DI module, NI 9205 Voltage Input Module, NI 9264 Voltage Output Module. After the algorithm showed satisfying behaviour in the simulation environment, we decided to test it in a real-world scenario. During the experiment we used the Nvidia Jetson TX2 in the Nissan Leaf. The Nvidia Jetson TX2 is a 7.5-watt embedded controller where we used Ubuntu 18.04 (Bionic) along with ROS Melodic Morenia. The embedded controller has 8 GB of memory and 59.7 GB/s of memory bandwidth, NVIDIA Denver2 and quad-core ARM Cortex-A57 CPU, and an integrated 256-core Pascal GPU. During the experiments we only used the KVH GEO-FOG 3D sensor as the source of localization we provided the wheel angle reference signal alongside with the speed reference via CAN and we measured the wheel angle and speed back in the same protocol. Nevertheless, during the data acquisition we logged all the sensory information, such as the 2x16 channel Velodyne LIDAR, the single channel SICK LMS 111 laser scanner, the camera stream into rosbag files. Figure 86 shows an example how the algorithm generated reference trajectory for two turns in the routine track, and also the measured wheel angle is shown.

3.2.5 Conclusion and evaluation

This thesis described a novel approach to improve the widely used pure-pursuit algorithm. The proposed algorithms are usable at low vehicle speed with a reference trajectory generated by planner components or with an explicit reference trajectory (e.g. GNSS-based waypoints). All resources and descriptions needed to reproduce the algorithm are available at our organization's public repository.

Table I summarizes the validation results and comparison with other versions of pure pursuit. Two tracks were selected: a segment of the ZalaZone proving ground, and a driving school test track near our university's campus. We used three different methods to validate the implementation. The first two methods are simulation-based testing of the algorithm: first is a simple simulation with simple graphical implementation. This simulation is capable of 2D visual feedback, alongside monitoring lateral deviation from the preset trajectory. This Python-based implementation is shared at our organization's GitHub repository. In the next step, we implemented this algorithm as a Robot Operating System (ROS) node. We embedded this node into a simulated version of our testing car. The simulation was based on OSRF Gazebo, which allowed us the versatile software

integration and to test with dynamic properties involved. This implementation is currently available at private repositories and might be released to the public in the future. The implementation is ROS-based, we could test this software both in simulation on our testing vehicle, without significant modification in the code. Simulation-based execution proved no significant deviation from the simulated tests. A working implementation of the presented algorithm is shared at our organization's public GitHub repository.

Table 3 - Some results accordance to the stated metrics

Method	Parameters	Deviation [m] ZalaZone track	Deviation [m] driving school
Follow-the-carrot	10.00	Avg: 0.65 Max: 2.76	Avg: 0.55 Max:2.76
Follow-the-carrot	10.50	Avg: 0.74 Max: 3.12	Avg: 0.63 Max: 2.95
Pure-pursuit	2.00	Avg: 0.10 Max: 0.56	Avg: 0.08 Max: 0.51
Pure-pursuit	2.33	Avg: 0.12 Max: 0.64	Avg: 0.09 Max: 0.60
Pure-pursuit	2.67	Avg: 0.13 Max: 0.74	Avg: 0.11 Max: 0.66
Pure-pursuit	3.00	Avg: 0.15 Max: 0.97	Avg: 0.13 Max: 0.83
Pure-pursuit	10.00	Avg: 0.52 Max: 1.90	Avg: 0.46 Max: 1.90
Look-ached distance according speed	2.33; 2.0	Avg: 0.13 Max: 0.65	Avg: 0.17 Max: 0.72
Look-ached distance according curves	2.33	Avg: 0.12 Max: 0.64	Avg: 0.10 Max: 0.60
Multi goal pursuit	4; 5; 6	Avg: 0.48 Max: 1.40	Avg: 0.30 Max: 1.34

Further work This algorithm can be further improved, besides typical enhancements (e.g. testing, code optimization, etc.). A very obvious but rather technical improvement would be the migration of the ROS-based code to the more recent ROS 2 basis, which allows real-time communication between components. One of these possible improvements is to use an optimization framework to select the best curve out of the generated curves. As my first experiments, I tested the behaviour of the algorithm in own simulation environments; the two presented approaches (the plain multigoal pursuit and the windowed multigoal version) were parallelly developed in Python and in MATLAB. These

simulations also used real-word trajectories and the vehicle model was first a kinematic, later a dynamic one. After that, a requirement emerged to test the algorithms even in OSRF Gazebo for two reasons. First, we wanted to implement the algorithm as an ROS node and second there was already a more precise dynamic model of the vehicle in Gazebo. Gazebo also allowed us the useful software integration, agile migration between development and target. For these purposes we had to recreate the algorithm in C++ not only because the ROS compatibility but also for computational resources. Simulation-based execution proved no significant deviation from the simulated tests. In Figure 84 some of the results is shown regarding the examined trajectories. The x and the y axis is in meters, this simulation is based on a real-word measurement in the ZalaZone vehicle industry proving ground and the GPS coordinates are in Universal Transverse Mercator (UTM), because of the minimal distortion it involves. We had to implement the follow the carrot, the speed ratio based pure pursuit, in order to experiment with them in the same environment.

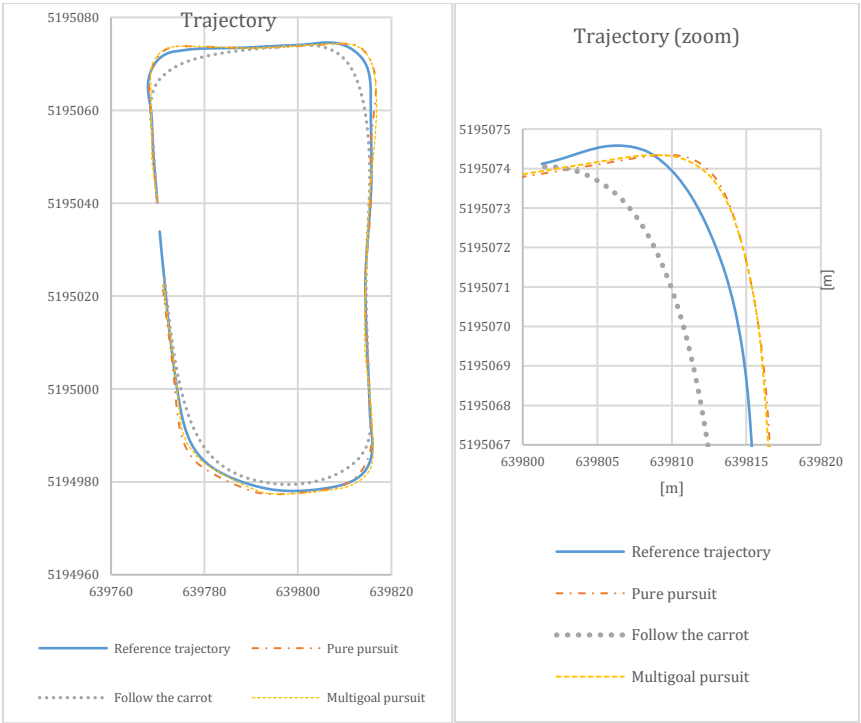


Figure 84 - Typical behaviors of the examined approaches

As another experiment we instrumented an electric vehicle (Nissan Leaf) with autonomous functionalities. Figure 85 represents the vehicle in operation. The

operational domain of this vehicle is limited to the ZalaZone vehicle industry proving ground and the university campus and is required to reach a relatively slow speed of 25 km/h. This vehicle is front wheel driven, with a wheelbase of 2.7 meters and a track width of 1.77 meters. To ensure instantaneous localization of the vehicle we used a high accuracy RTK capable GPSs. One of them was the KVH GEO-FOG 3D the other was Swift Navigation Duro Inertial RTK. These sensors were also used to capture reference trajectories.

Our software was based on OSRF Robot Operating System (ROS) [70]. Our computing platform features first the Nvidia Jetson TX2, later the Nvidia Jetson AGX Xavier. The low-level control is realized on a National Instruments CompactRIO Controller: cRIO-9039 as the Real-time and FPGA module, NI 9853 CAN module, NI 9403 DI module, NI 9205 Voltage Input Module, NI 9264 Voltage Output Module.

Figure 85 - The vehicle used in the experiments

After the algorithm showed satisfying behaviour in the simulation environment, we decided to test it in a real-world scenario. During the experiment we used the Nvidia Jetson TX2 in the Nissan Leaf. The Nvidia Jetson TX2 is a 7.5-watt embedded controller where we used Ubuntu 18.04 (Bionic) along with ROS Melodic Morenia. The embedded controller has 8 GB of memory and 59.7 GB/s of memory bandwidth, NVIDIA Denver2 and quad-core ARM Cortex-A57 CPU, and an integrated 256-core Pascal GPU. During the experiments we only used the the KVH GEO-FOG 3D sensor as the source of localization we provided the wheel angle reference signal alongside with the speed reference via CAN and we measured the wheel angle and speed back in the same protocol. Nevertheless during the data acquisition we logged all the sensory information, such as the two 16

channel Velodyne LIDAR, the single channel SICK LMS 111 laser scanner, the camera stream into rosbag files. The following chart (Figure 86) shows an example how the algorithm generated reference trajectory for two turns in the routine track, and also the measured wheel angle is shown.

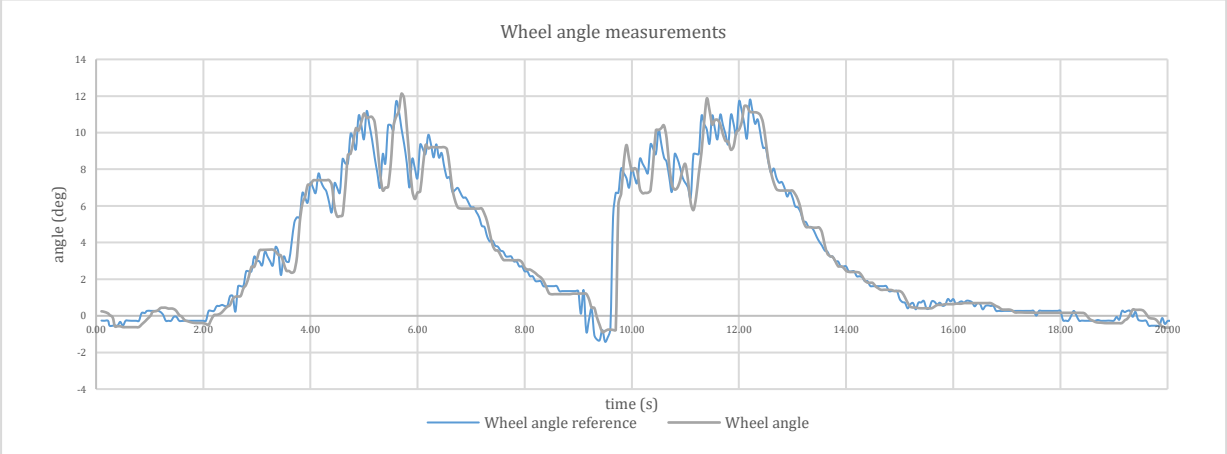


Figure 86 - The wheel angle change around two corners

The waypoints for the algorithm were provided by driving through a predetermined path, saving and filtering the position and speed data via the waypoint_saver node. The waypoints were somewhere inside a 3.5 m wide traffic lane, and as listed in Table 3 the algorithm's average deviation was 0.3 and 0.48 m, which was enough to keep the vehicle in track smoothly. In this experiment we only used the multigoal pursuit algorithm with one parameter set. This set is better compared to the Autoware pure pursuit realization in terms of lateral deviation, but the speed ratio-based version worked even better. As a result, we obtained that the multigoal pursuit is able to drive the vehicle with the mentioned embedded controller at slow speeds, so about 25 km/h. The algorithm is able to perform smooth and continuous movement along a predefined trajectory, even manoeuvring around corners and edges. The main advantage of the proposed algorithm is the human-like reasoning involved during the trajectory following.

3.3. Thesis 3.

I showed that probabilistic occupancy grid map construction is possible to be done via the "crisp signatures" sensory model. The novelty of this approach consists of involving the concept of signatures in map construction and the elimination of Bayesian sensory, so the probabilistic characteristics of the environment model could be involved in the mapping phase. This approach is considered as another mathematical sight of sensory model realization and works well especially with time-of-flight-based sensors such as a LIDAR. The proposed approaches are validated by simulation and experimental results, the main benefits of the proposed approach consist of the signature-structured handling of complex sensor data and the algorithmic separation of the probabilistic characteristics.
References: [H1], [H3], [H12], [H14], [H23], [H24], [H25]

Nowadays the low-level tasks of self-driving - or other in words autonomous - vehicles and robots are intensively researched. These low-level tasks include for example the path-planning and obstacle avoidance task. Both path planning and obstacle avoidance usually relies on map building. In the following the occupancy grid based probabilistic map building will be examined. This means that the localization algorithm builds the map in probabilistic manner. In 1998 Thrun, Burgard and Fox proved, that the probabilistic approach can be used as a method involved in the SLAM (Simultaneous Localization And Mapping) problem [9]. Various sensory models are calculating the probability in different ways. In the occupancy grid map every grid cell is represented as probability, most often a number between 0 and 1. Several methods such as Bayesian, Dempster-Shafer, Fuzzy, Borderstein or TBF sensor representation [10] can be mentioned. The proposed solution originality consists of two things. The first is defining a probabilistic function for each element of the grid, the second is involving the concept of signatures in the process. The proposed sensory model is called as the *crisp* model. Figure 90 and Figure 91 shows the difference between the Bayesian and the crisp representation. Here the discrete 2D measurement was taken and the 3rd dimension comes from occupied feature of the given coordinate. On the figures the numbers close to or equal to 1 means that the given coordinate (grid cell) is occupied. Accordingly, if the coordinate is empty numbers close to or equal to zero is chosen. All other areas are initialized with 0.5 because there is no prior knowledge about this grid cells. On the visualization of the crisp sensory model

(Figure 90) it is visible that there are no values other than 0, 0.5 and 1. On the other hand Bayes sensory model (Figure 91) has several other values between 0 and 1. According to Sebastian Thrun [71] LIDAR range finders do not always require the probabilistic – for example Bayesian – sensory model for a single measurement.

3.3.1 Application of signatures

The classical, Bayesian sensory model does not involve signatures. The signatures are flexible mathematical constructions which can contain multiple measurements of a single-channel LIDAR (2D LIDAR) or even multiple measurements of a multiple-channel LIDAR (3D LIDAR). Considered $A_S(x)$ as signature, which corresponds to a single-channel LIDAR measurement.

$$A_S(x) = \begin{bmatrix} \begin{bmatrix} a_1 \\ \begin{bmatrix} a_{1,1} \\ a_{1,2} \\ a_{1,3} \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} a_2 \\ \begin{bmatrix} a_{2,1} \\ a_{2,2} \\ a_{2,3} \end{bmatrix} \end{bmatrix} \\ \cdots \\ \begin{bmatrix} a_n \\ \begin{bmatrix} a_{n,1} \\ a_{n,2} \\ a_{n,3} \end{bmatrix} \end{bmatrix} \\ p \end{bmatrix} \tag{100}$$

Where at given m measurement a_m tells if the cell is occupied, $a_{m,1}$ is the ID, $a_{m,2}$ is the distance $a_{m,3}$ angle, and p is an optional parameter for the pose of the sensor. This gives us the possibility to easily represent the LIDAR measurement. The other benefits of this approach is that it can be easily extended to a multiple-channel LIDAR too and the available signature operations can be applied too. For example, if a resolution is changed or another range sensor is applied, the extension and contraction operation is available. The generalization of the range sensor (Figure 87) can a viewed as an $A_S(x)$ signature described in equation (100).

$$A_M(x) = \begin{bmatrix} A_{S_1} \\ A_{S_2} \\ \dots \\ A_{S_C} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} a_{1,1} \\ \begin{bmatrix} a_{1,1,1} \\ a_{1,1,2} \\ a_{1,1,3} \end{bmatrix} \end{bmatrix} \begin{bmatrix} a_{2,1} \\ \begin{bmatrix} a_{2,1,1} \\ a_{2,1,2} \\ a_{2,1,3} \end{bmatrix} \end{bmatrix} & & \begin{bmatrix} a_{c,1} \\ \begin{bmatrix} a_{c,1,1} \\ a_{c,1,2} \\ a_{c,1,3} \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} a_{1,2} \\ \begin{bmatrix} a_{1,2,1} \\ a_{1,2,2} \\ a_{1,2,3} \end{bmatrix} \end{bmatrix} \begin{bmatrix} a_{2,2} \\ \begin{bmatrix} a_{2,2,1} \\ a_{2,2,2} \\ a_{2,2,3} \end{bmatrix} \end{bmatrix} & \dots & \begin{bmatrix} a_{c,2} \\ \begin{bmatrix} a_{c,2,1} \\ a_{c,2,2} \\ a_{c,2,3} \end{bmatrix} \end{bmatrix} \\ \dots \quad \dots & & \dots \\ \begin{bmatrix} a_{1,n} \\ \begin{bmatrix} a_{1,n,1} \\ a_{1,n,2} \\ a_{1,n,3} \end{bmatrix} \end{bmatrix} \begin{bmatrix} a_{2,n} \\ \begin{bmatrix} a_{2,n,1} \\ a_{2,n,2} \\ a_{2,n,3} \end{bmatrix} \end{bmatrix} & & \begin{bmatrix} a_{c,n} \\ \begin{bmatrix} a_{c,n,1} \\ a_{c,n,2} \\ a_{c,n,3} \end{bmatrix} \end{bmatrix} \end{bmatrix}^T \tag{101}$$

The signature consists of multiple A_S signatures: $A_M(x) = [A_{S_1} \quad A_{S_2} \quad \dots \quad A_{S_C}]^T$, where C is the number of channels. In the thesis only this approach is examined, but note that one of the benefits of this approach is its flexibility to extend and write mathematical operations which can adapt to this flexibility.

3.3.2 Mapping

Map construction is a process that intends to spatially represent the objects in a particular space. At the limit, not in the objects themselves is interesting, but particularly in the obstacles, that is, those locations that cannot be accessed by the robot. These locations are called occupied. It is already understood that the locations mentioned are defined by a position vector, in an absolute reference system. The obstacles discovery is done by a sensory system mounted on a mobile robot or vehicle that moves in the absolute reference system. This means - that finally the measured position of obstacle must be transformed from the sensor referential frame in the absolute referential frame. This is not a trivial problem because of the following:

- The pose (position and orientation) of the robot is modelled in a stochastic way;
- The sensor measurements are also influenced by noise, so once again probabilities are to be considered;
- The obstacle position is assimilated with a grid cell occupancy;
- A particular location can be subject of different measurements whose results are necessary to be correlated.

The problem which is intended to solve is to construct an original mathematical model of this process. The originality of the approach consists on computational simplification. This simplification can be done because of the following:

- The original sensory model was developed with respect to the early, noisy sensors (e.g. sonars). For example, a LIDAR produces more reliable measurement, even the measurement noise is still involved.
- The measurements are more frequent, a grid cell is evaluated many times.
- The probabilistic computation can be involved into the map construction in update phase.

Simulations and experiments proved that these simplifications did not have diminished the usability of the result, but on the contrary the fact that they provided less computation time and an easier use of result. Mapping can be separated in the following steps:

1. *Localization* of the mobile robot (in the absolute referential frame) i.e. the pose computation as a compromise between the robot displacement model and the measurements relative to known environmental elements. Kálmán or particle filters are the most used solutions. The mobile robot navigates and performs localization at time t (see 103-107). After each localization the mobile robot performs several measurements $i \in [1, n]$ see (103-107).
2. *Range measurements* used for obstacles discovering. For each discovered obstacle the measurements consist from the following elements: the distance to the obstacle and the angle between the beam of the measuring and the direction of the robot. The mentioned elements are directly obtainable from the sensor. It is also necessary to consider an additional stage, where these measures are transformed into a position of the obstacle defined in the sensor referential system.
3. *Transformations* of the obstacle position from the sensor referential system into the absolute referential system. Homogenous transformations, which involve the robot pose, solve the problem. The obstacle position is transformed into a grid cell occupancy value. The map is assimilated to a discrete space i.e. a grid map.
4. Finally, the (grid) *map update*, where previous measurements are linked to current measurements. Usually a Bayesian filter is used in order to modify the previous hypothesis concerning the occupancy of the map position.

From [71] the location belongs to the Bayesian localization class, and can be described with the following equations which encompass a two phases: prediction and correction of the actual estimate. This can be formalized as follows.

$\{bel(x_{t-1}), u_t, z_t, m\}$

for _all_ x_t _do

$$\begin{cases} \overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}, m) \cdot bel(x_{t-1}) \\ bel(x_t) = \eta\, p(z_t \mid x_t, m) \cdot \overline{bel}(x_t) \end{cases} \tag{102}$$

endfor

$\{bel(x_t)\}$

where: x_t, u_t, m, z_t are the system state, the command signal, the map and the measurements;

The prediction takes account of the model and the correction of the measurements. Firstly an estimated prediction is established $\overline{bel}(x_t)$. In the second part the estimated prediction is corrected using the measurement model $p(z_t \mid x_t, m)$;

For the localization step the following model is proposed:

$\{bel(\boldsymbol{x}), u_t, \boldsymbol{z}_t, m\}$

for _all_ $\boldsymbol{x}$ _do

$$\begin{cases} \bar{\sigma}_{m,t} = \sigma_{m,t-1}\sigma_m \\ \overline{bel}(\boldsymbol{x}) = \bar{\sigma}_{m,t}\delta(\boldsymbol{x}, \boldsymbol{x}_t) \\ m(\boldsymbol{x}) = \sigma_s \delta(\boldsymbol{x}, \boldsymbol{z}_t) \\ \sigma_{m,t} = (\bar{\sigma}_m + \sigma_s - \bar{\sigma}_m\sigma_s) \\ bel(\boldsymbol{x}) = \sigma_{m,t}\delta\left(\boldsymbol{x}, \frac{\bar{\sigma}_{m,t}\boldsymbol{x}_t + \sigma_s \boldsymbol{z}_t}{\bar{\sigma}_{m,t} + \sigma_s}\right) \end{cases} \tag{103}$$

endfor

$\{bel(\boldsymbol{x})\}$

where:

$\boldsymbol{x}$ is a state variable, the mobile robot pose;

$$\delta(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 \;\; if \;\; \sum_{i=1}^{n} |x_i - y_i| = 0 \\ 0 \;\; if \;\; \sum_{i=1}^{n} |x_i - y_i| \neq 0 \end{cases}$$

$x_t = f(x_{t-1}, u)$ is the mobile robot model;

$\sigma_{m,t}$ is the degree of trust of the robot pose at moment t;

$\bar{\sigma}_{m,t}$ is the estimated degree of trust of the robot pose at moment t

σ_m is the degree of trust of the model;

σ_{st} is the degree of trust of the sensor measuring;

For the corrected degree of trust $\sigma_{m,t} = (\bar{\sigma}_m + \sigma_s - \bar{\sigma}_m\sigma_s)$ the condition is imposed as $\begin{cases} 1 \geq \sigma_{m,t} \geq \bar{\sigma}_m \\ 1 \geq \sigma_{m,t} \geq \sigma_s \end{cases}$.

For the obstacle measurements (identification), the following model is proposed:

$$\begin{cases} \sigma\left({}^1\boldsymbol{x}_{i,t}\right) = \sigma^0 \delta({}^1\boldsymbol{x}_{i,t}, {}^1\boldsymbol{z}_{t,i}) \\ \sigma_t^0 = \sigma_{m,t}\sigma_s \end{cases} \tag{104}$$

where: ${}^1\boldsymbol{z}_{i,t} = \begin{bmatrix} l_i \cos(\alpha_i) \\ l_i \sin(\alpha_i) \end{bmatrix}$; ${}^1\boldsymbol{x}_{i,t} = \lambda\, {}^1\boldsymbol{z}_{i,t}$

${}^1\boldsymbol{x}_{i,t}$, ${}^1\boldsymbol{z}_{i,t}$ are vectors which belongs to the i^{th} measurement and are defined in the robot referential system, $\lambda \in [0,1]$; i refers to the number of the measurement.

This step is represented with the robot pose degree of trust is $\sigma_m$and the occupancy degree of trust is σ_{oi}.

The measurements results must to be transformed from the robot referential system into the absolute referential system:

$$\begin{cases} \boldsymbol{x}_{i,t} = \begin{bmatrix} \cos(\theta_t) & -\sin(\theta_t) & r_{tx} \\ \sin(\theta_t) & \cos(\theta_t) & r_{ty} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{x}_{t,i} \\ 1 \end{bmatrix} \\ \bar{\sigma}\left(\boldsymbol{x}_{i,t}\right) = \sigma({}^1\boldsymbol{x}_{i,t}) \end{cases} \tag{105}$$

where: r_{tx}, r_{ty} and θ_t are the position and the orientation of the mobile robot.

The map update is can be solved e.g. by Bayesian the formula:

$$p(h|o) = \frac{p(h)p(o|h)}{\sum_{i=2}^{2} p(h_i)p(o|h_i)} \tag{106}$$

where: h is the hypothesis that the location x is occupied; o is here the observation (the values of z); in fact, there are 2 hypotheses: x is occupied and x is free.

For the map update, the following model is proposed:

$$\begin{cases} \sigma(\boldsymbol{x}_{i,t}) = \frac{1}{2}\left(\tanh\left(\frac{s}{2} - \frac{1}{2}\right) + 1\right) \\ s = \begin{cases} \sigma(\boldsymbol{x}_{i,t-1}) + 1 & if \quad \sigma(\boldsymbol{x}_{i,t}) \neq 0 \\ \sigma(\boldsymbol{x}_{i,t-1}) - 1 & if \quad \sigma(\boldsymbol{x}_{i,t}) = 0 \end{cases} \end{cases} \tag{107}$$

3.3.3 Occupancy grid

Occupancy grid map consists of discrete array (matrix) of numbers where these numbers specify whether the grid cell is occupied or not.

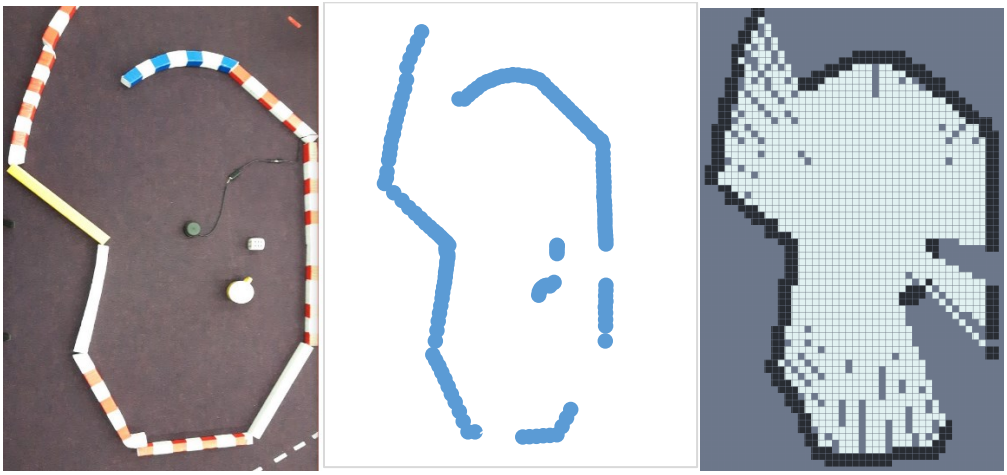


Figure 87 - On the left an image of the measurement, in the middle the measured points, on the right a possible occupancy grid map is displayed, where the occupied area is red, the free area is green

Broadly speaking the creation of an occupancy grid means a conversion from continuous domain to a discrete domain with given resolution and given methods (e.g. Bresenham algorithm or flood fill algorithm). The theoretical background is of course clarified in the "Related work and recent results" section.

3.3.4 Bresenham ray casting

On occupancy grid maps calculating the Bresenham ray casting algorithm can define the elements which can be used to draw a line on a discrete two-dimensional arrays (such as

bitmap image or occupancy grids). It only uses lightweight operations such as integer addition, subtraction and bit shifting, thus it can be reasonably fast. The algorithm accepts two parameters: the start and the end grid cell or in other words, the start and end pixel, both of them consist of an x and a y coordinate.

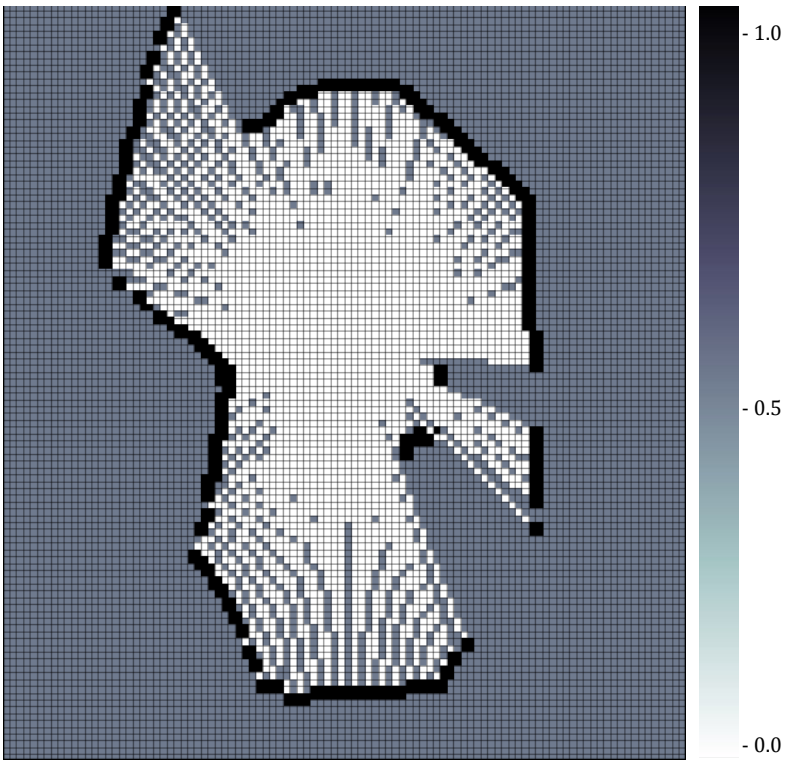


Figure 88 - Bresenham

3.3.5 Flood fill

Like the Bresenham ray casting, flood fill algorithm also works on two-dimensional arrays (such as bitmap image or occupancy grids). Although the purpose is different: flood fill intends to fill the connected areas if they have the same values. Filling in this context simply means to give another value to the connected areas. Therefore, this algorithm accepts the following parameters: the grid map, a start grid cell (start pixel - x and y coordinate), a value to replace, and a new value where all connected grid cell's value will be replaced.

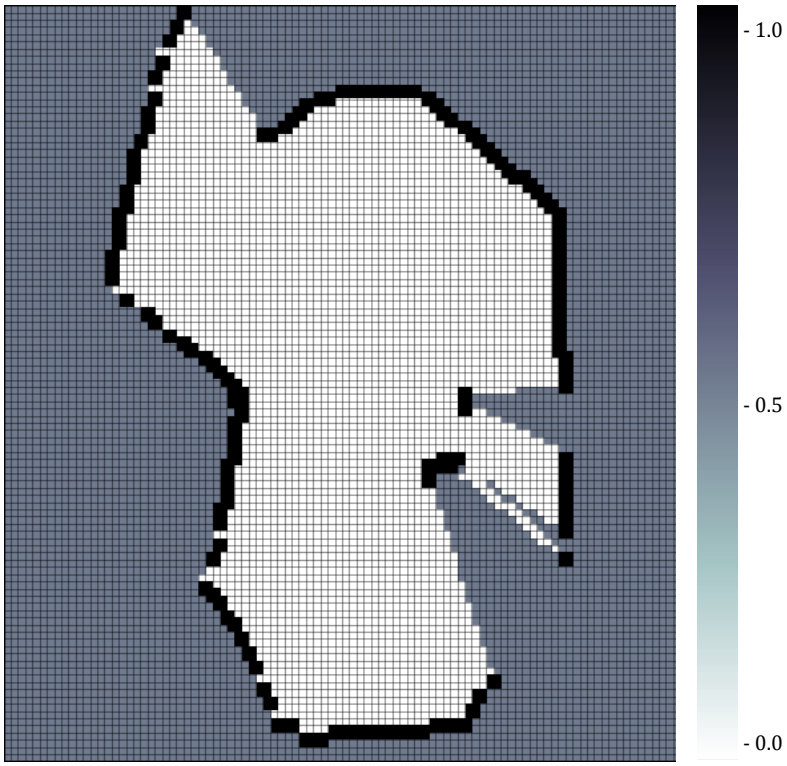


Figure 89 - Flood fill

3.3.6 The crisp sensory model with signatures

Probabilistic map building is necessary because of the localization inaccuracies. These characteristics will be introduced in the map building phase.

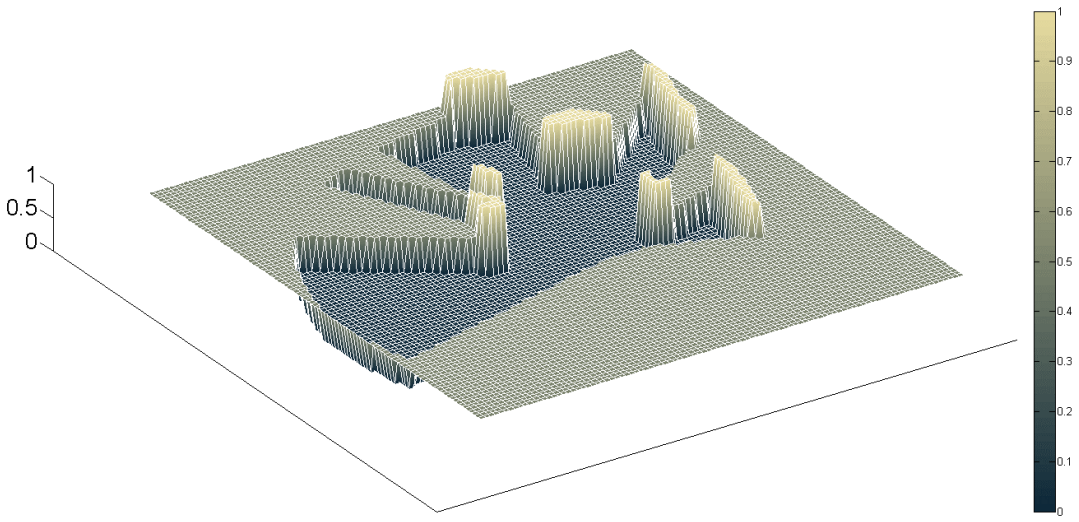


Figure 90 - Visualization of a single LIDAR measurement (scan) with crisp representation in 3D

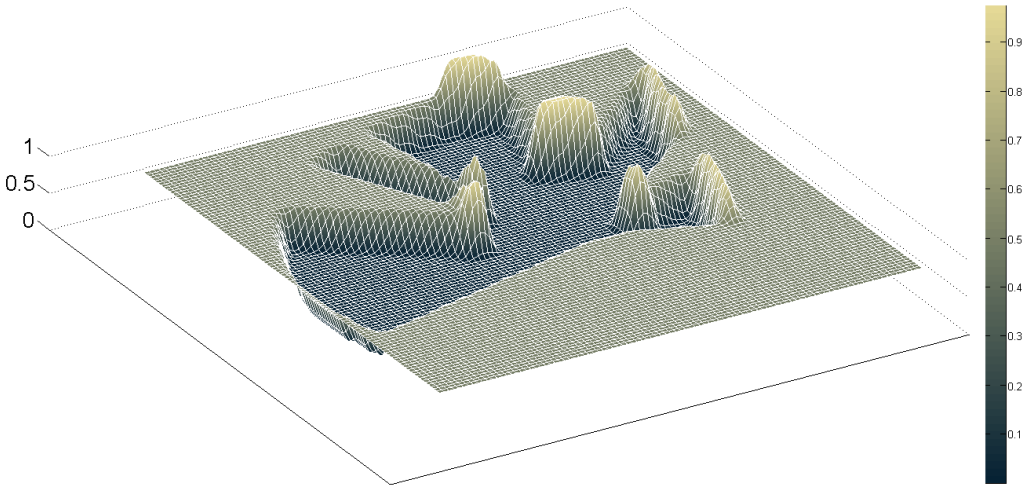


Figure 91 - Visualization of a single LIDAR measurement (scan) with Bayesian representation in 3D

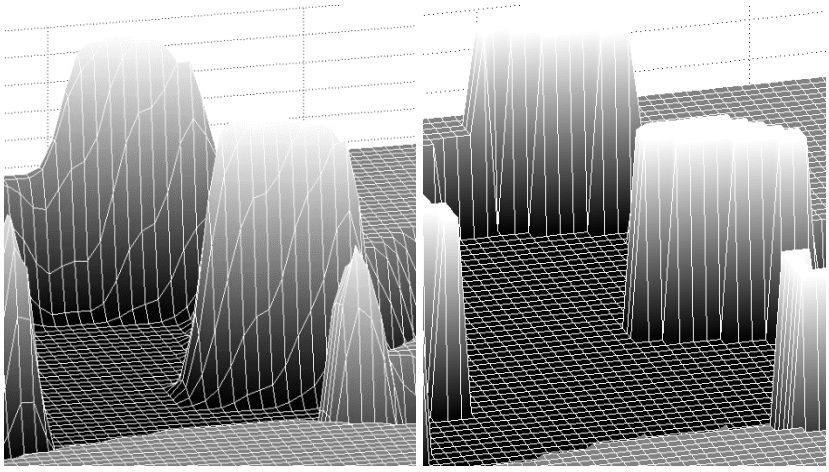


Figure 92 – Closer look to empathise the difference between crisp and Bayesian representation in 3D

An element - a cell - of the map can be associated either with an obstacle or an empty area i.e. two hypothesis can be defined for a cell: h_o *the cell is occupied* or h_e *the cell is free*. The mathematical representation of these hypotheses is probabilistic: $p(h_o)$ respectively $p(h_e)$. The correlation between these probabilities is modelled by equation (108).

$$p(h_o) + p(h_e) = 1 \tag{108}$$

The proposed algorithm stores only the occupied grids. As a result we obtained a i by j matrix or a 2D array $L_w(i,j)$, which will be named the wall layer. Because of (1) is no need to store the empty grids. As an initial step the wall layer has a uniform probability of $p(h_o) = p(h_e) = 0.5$. After each measurement elements of $L_w(i,j)$ are recalculated according to equation (109).

$$\begin{aligned} p(h_o|o) &\propto p(h_o) \cdot p(o|h_o) \\ p(h_e|o) &\propto p(h_e) \cdot p(o|h_e) \end{aligned} \tag{109}$$

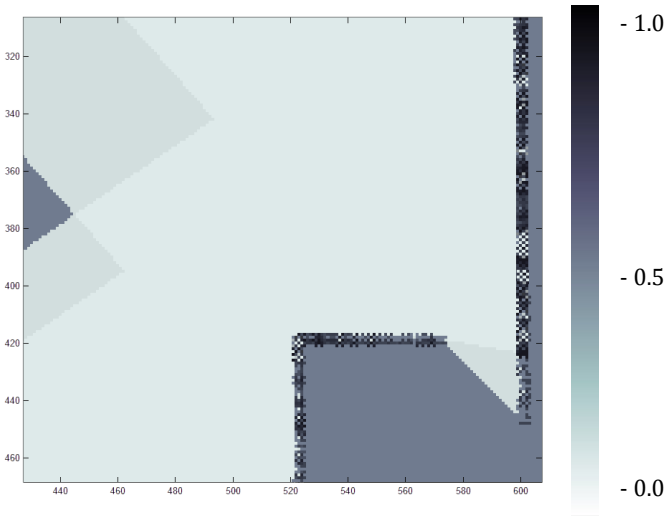


Figure 93 - The probabilistic grid map where colours represent the probability of the wall: 0.5 is grey, close to 0 is white, close to 1 is black. Two measurements were performed in the visible area

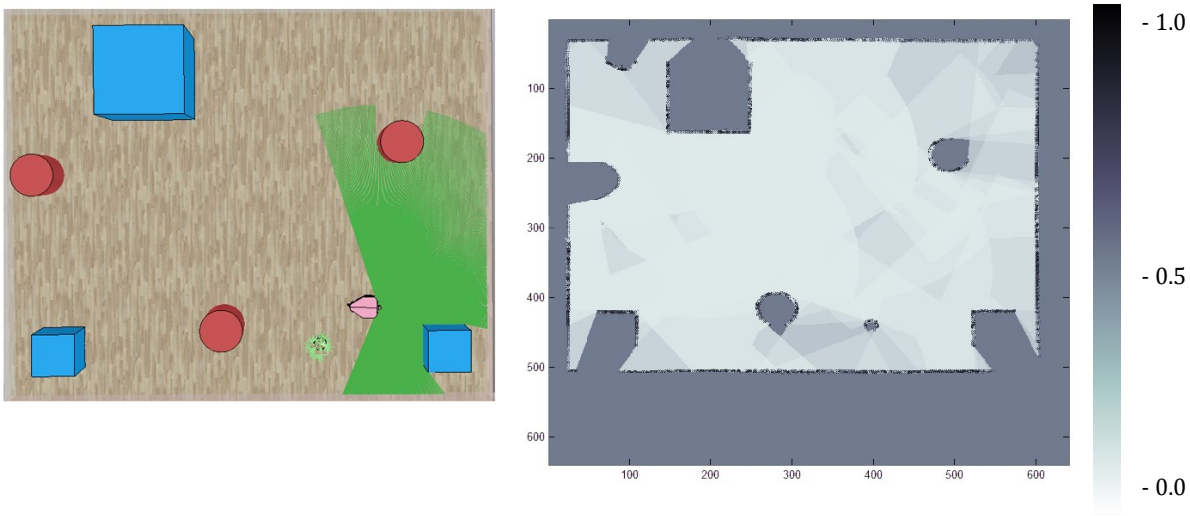


Figure 94 - The visualization of the measurement (MATLAB)

Using Bayes' theorem the probabilities are normalized (110) .

$$p(h_o|o) = \frac{p(h_o) \cdot p(o|h_o)}{p(h_o) \cdot p(o|h_o) + p(h_e) \cdot p(o|h_e)}$$

$$p(h_e|o) = \frac{p(h_e) \cdot p(o|h_e)}{p(h_o) \cdot p(o|h_o) + p(h_e) \cdot p(o|h_e)} \quad (110)$$

This algorithm produces a grid map similar to Figure 94. We have used a convention accordingly the highest probability is red and the lowest probability is blue. On Figure 93. only two independent measurements are visualized for the sake of perspicuity, even the algorithm uses an iterative process with several measurements where each measurement is used for map improvement.

In order to prove my concept, an experiment was carried out, where we can demonstrate that the approach is working. We have chosen a mapping with known poses [22] scenario. In the experiment we have chosen a Gazebo as the simulator, and we created a single-channel LIDAR with 270° field of view. The sensor has a 1° resolution, so a measurement consists of a signature where the 270 measurements and the pose is represented.

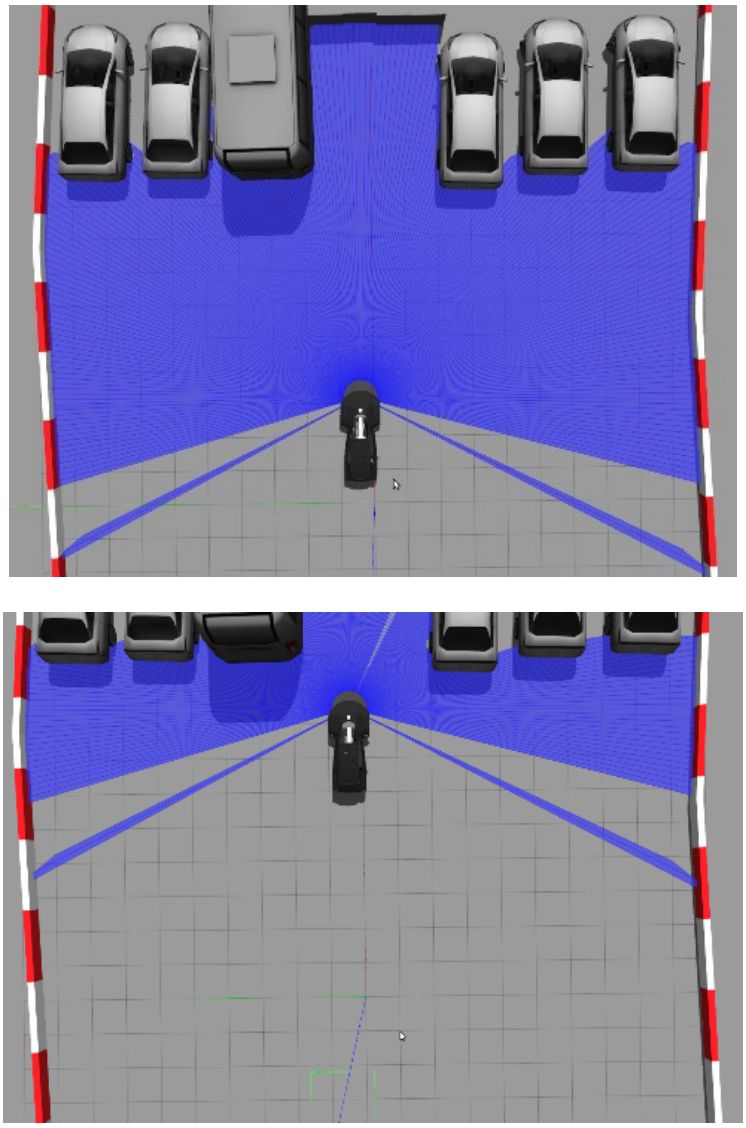


Figure 95 - The scenario to prove the concept

The scene in the simulator is visible on Figure 95, and the corresponding occupancy grid map is on Figure 96. In this scenario a simple parking manoeuvre is carried out, but the focus was not on the manoeuvre itself, rather on the map building capabilities with signatures.

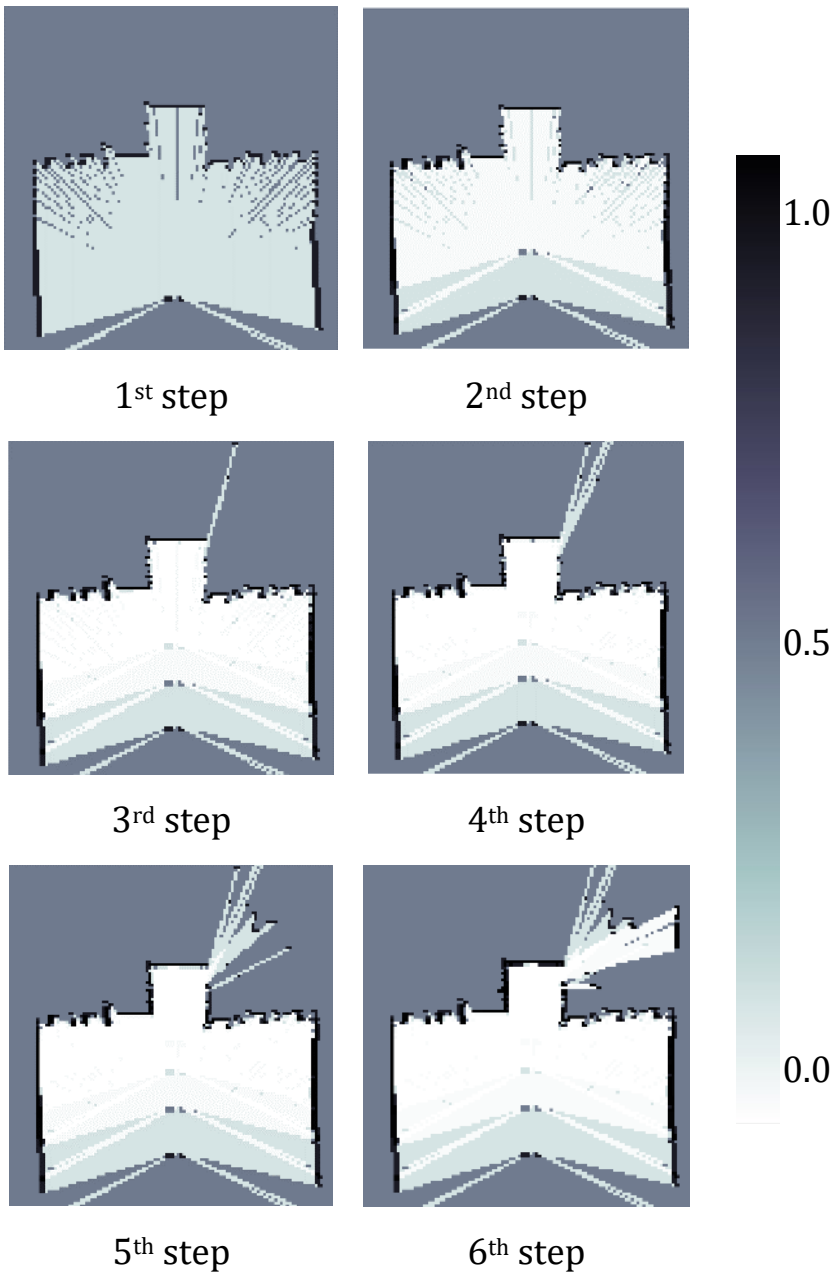


Figure 96 - The results obtained from the simulation (ROS and python)

The grid map is chosen to be 120x120 cells, this is a reasonable resolution given the overall scenario size and the objects. The pose and the range measurements are accessible from Gazebo, we wrote a simple script in order to visualize the mapping.

3.3.7 Conclusions

The proposed crisp signatures sensory model involves signatures as a mathematical tool to structure the sensory data better and also lets the probabilistic characteristics of the environment model involved in the mapping phase. This approach is considered as another mathematical sight of sensory model realization. A showed the sensory model works well especially with time-of-flight-based sensors such as a LIDAR. The main benefits of the proposed approach consist of the algorithmic separation of the probabilistic characteristics, the signature-structured handling of complex sensor data and thus the capability of extension and transfer.

3.4. Thesis 4.

I showed that in narrow subdomain of perception for experimental vehicles a VGG-VD neural network trained on low annotation categories with transfer learning case RMSprop converges slightly faster than Adam optimizer; despite the known fact, that that Adam generally overperforms RMSprop. To prove this phenomenon the results, dataset and necessary conditions have been shared.
References: [H16], [H20], [H21]

As nowadays necessities are arising in the fields of sound and image recognition, self-driving decision making, data processing and understanding, the more attention the neural network (NN) and deep learning (DL) solution gets. From the point of view of my scientific work the subdomain of perception and for mobile robots and road vehicles are particularly important. Instead of examining general neural network disciplines this thesis focuses on a narrow subdomain of this field. In the following the few-annotation category, segmentation-based, perception problems are targeted. Perception in the domain of vehicles or in robotics as general is similar to the conventional computer vision domain tasks [72]. It must be emphatized that this is a similarity not identity, the techical and techological approaches are smimilar but the objective is different [73]. Here, the fundamental problems are called sensing and perception. Sensing is defined as a data acquisition and transfer from a sensor (e.g. camera) whereas perception is more like a multiform process. Basically the understanding or the analysis of the data is done there. As mentioned the research targets the perception where the features are extracted from the sensors data after the acquisition. Compared to the classical robotics problems there are less tasks involved (e.g. there is no grasping or assistant systems) but the objects which are involved are much more diverse. In the classical computer vision (CV) domain, which are often present in robotics conventional algorithms extract information from the raw sensor data based on artificially designed features (pl. SIFT [74], SURF [75], ORB [76], pattern recognition [77], template matching [77]). In contrast of this approach the NN and DL solutions [41], [55] use fundamentally different principle regarding these and other [78] problems. Nowadays there is clearly a trend which supports the solution of complex approximation problems with growing computational powered hardware (GPU, CPU, TPU) and multi layered networks such as convolutional neural networks. The thesis

focuses on neural network-based solutions where the problem characteristics defines few annotation categories. The dataset consists of general traffic, race and model vehicle images.

Figure 97 - Typical images from the Shell Eco-marathon competition

In the following the state-of-the-art techniques are collected and reviewed, these techniques are also applied and measured in the narrow subdomain. This is important because according to my knowledge these techniques in this methodology and scope weren't applied and measured before the work [H16, H20, H21]. Therefore, for example, there are results available of certain optimization techniques but not in terms of the scope and in this kind of profoundness. To be more specific there are already papers who compared optimizers such as Adam versus RMSprop, but on other architectures different result from ours these comparisons may have produced. The thesis tends to highlight the deviation between the general cases and the narrow subdomain and of course not only in the case of optimizers. Thus, the "third wave" of NN is quite new trend; the best way to get knowledge about these matters is to read scientific papers about it. According to my view, there is a gap between the theory of NN and the black box-like application of these techniques. To close this gap a little bit, in this paper besides theory the applied results is presented too.

This case we are dealing with pixel-wise segmentation or in other words semantics segmentation. In vehicles this is related to the conventional computer vision domain tasks. As a consequence of viewpoint variation, scale variation, intra-class variation segmentation is considered as a challenging task [54]. The segmentation problem is

intensively-researched, and nowadays neural network architectures are solving it with the most success [79] [56]. Many of them rely on a special instance of convolutional neural networks (CNN), the fully convolutional neural networks (FCN). To solve a segmentation problem properly you need to make a lot of choices. These choices involve defining the NN architecture, including the number of layers the relation between them, applying a good activation function, choosing a proper hardware, and execute whole training process with a suitable optimization technique. The next sections tend to help in these choices.

3.4.1 Gradient-based optimization techniques

All modern approach of neural network optimization takes advantage of the fact that all operations used in the training are differentiable. Differentiable in this context means that it "can be derived" and the derivative of a tensor operation is called the gradient (∇). The gradient is computable as the loss with respect to the network's coefficients. To decrease the loss the optimization needs to move in the opposite direction from the gradient. Let's consider a continuous function; here a small change in the input can only result in a small change in the output. That can be surely assumed from the definition of continuity. Because of fact that the function is smooth – or in other words it doesn't have any rapid curves - when small enough epsilon (ε) is changed it's possible to approximate the change in the output linear function of slope regarding epsilon. The slope is called the derivative of the function in the chosen point. The generalization of the concept of derivatives to multidimensional function's inputs are functions that take tensors as inputs.
The state of the art optimizers nowadays are: Adagrad, Adam, Proximal Gradient Descent and RMSprop. [80] Each optimizer belongs to the gradient descent optimizer family. The gradient is denoted (with the ∇ Nabla symbol) respect to the function $f(\theta)$ and v vector at each point θ is the directional derivative of f along v:

$$D_v f(\theta) = \nabla f(\theta) \cdot v \tag{111}$$

Each method is a variant of the vanilla gradient descent, where θ are the parameters, η is the learning rate, $J(\theta)$ objective function to the entire training set:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta_t) \tag{112}$$

The first optimizer examined for my case is Adagrad which involves stochastic optimization techniques and online learning which employ proximal functions to control the gradient steps of the algorithm [80] [81]. Adagrad applies different learning rate for every parameter θ_i at every time step t, where $g_{t,i}$ is the gradient of the objective function [80]:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i} \tag{113}$$

Proximal Gradient Descent is another gradient descent method. The algorithm alternates between two phases: [82] the first is performing unconstrained gradient descent and the second is solving an instantaneous optimization. This problem trades off minimization of a regularization term while keeping close proximity to the result of the first phase.
The last two optimizers are discussed together because they have similar approach. The advantage of Adam is that it does not require a stationary objective [83]. Adam works with sparse gradients, and it naturally performs a form of step size annealing. According to the literature [83] Adam slightly outperform RMSprop in general problems but in this case it's the opposite. It is true that Adam and RMSprop are similar in the means of performance but according to our experiments RMSprop is the one who performs slightly better, see figure 6. RMSprop is proposed by Tieleman and Hinton [84] note that this is not a classic publication rather a lecture. In case of RMSprop the equation also involves the he running average $E[g^2]$ [80]:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]t+\varepsilon}} \cdot g_t \tag{114}$$

From the mentioned optimizers RMSprop and Adam were the bests in our case. The heavy fluctuation in figure 6 is due the logarithmic scale, but this enlarges the differences between the loss curves. Choosing a proper learning rate can have fortunate consequences to the training progress.

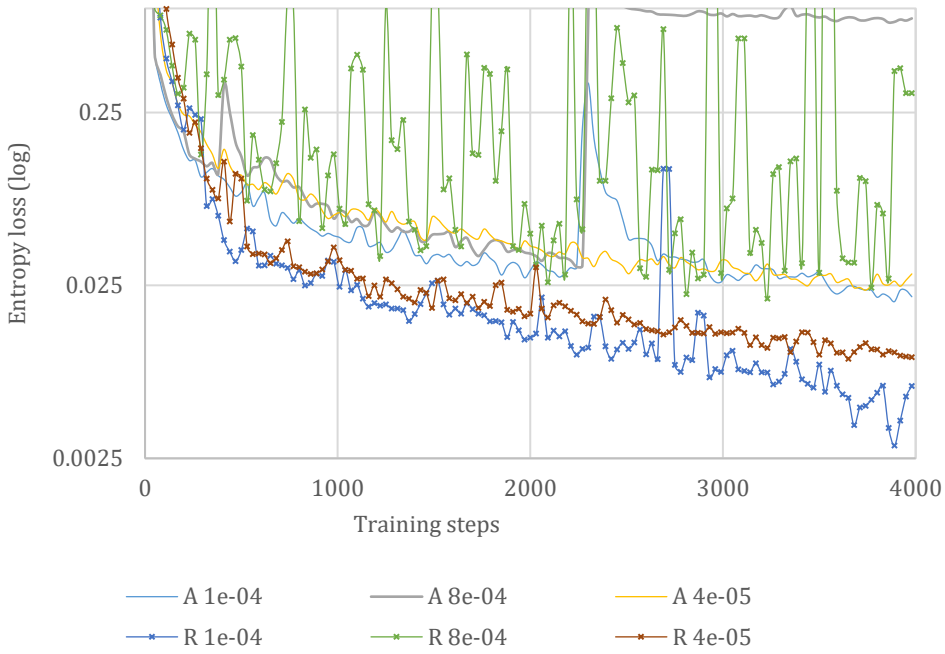


Figure 98 - Adam and RMSprop optimization algorithms (A – Adam, R – RMSprop) each with 3 different learning rates (10^{-4}, 8×10^{-4}, 4×10^{-5})

Figure 99 – Typical characteristics of entropy losses when choosing high / low / appropriate learning rates

High learning rate can have two kinds of effects regarding the entropy loss. The first is slow convergence or not even reaching the desired minimum, the second is fluctuating

loss. However according to our experiences this undesirable fluctuation may lead the loss to a different maybe even better local minimum. This behaviour is of course not easily reproducible; it has random characteristics. In extreme cases very high learning rate can even cause divergence.

3.4.2 Neural-network architectures

FCN in this case means that unlike a simple CNN, FCN only contains convolutional layers. FCN architectures works on feature extraction principle. Every layer is responsible for extracting different features such as colors, contours, edges, see figure 2. One of the well-known networks are the BVLC FCN [57] and their variables (FCN32-s, FCN16-s, FCN8-s), SegNet [85].

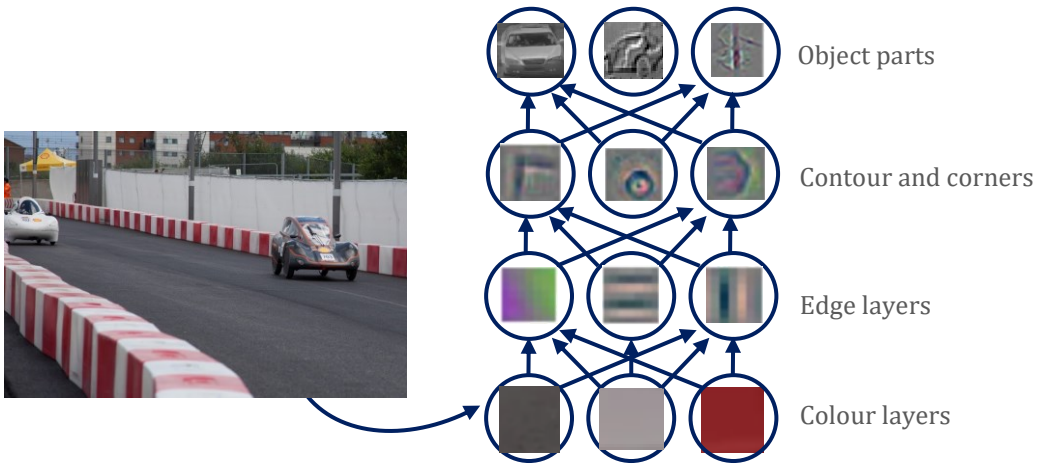


Figure 100 - The visualization of the neural network's logical layers typical image segmentation

Also there are possibilities to use CNN's which were made for classification, but extend them and use as FCN, and transfer their learned representations by fine-tuning. The example for this approach is AlexNet-FCN [57], DeepLab-FCN-CRF [60], the VGG-FCN [57]. Due their simple, but very deep network architecture and high performance the variations of the VGG-FCN networks were chosen to be examined further.

Due to the numerous parameters involved (e.g. VGG-VD19 has 156,351,488 and VGG-VD16 has 138,357,480 trainable parameters) training a convolutional network can be a highly time-consuming process. The training and application of a NN consists typically of various matrix operations such as matrix multiplication or addition, which are parallelizable operations. Since of this phenomenon GPUs (graphical processing units)

can overperfom CPUs (central processing units) because GPUs can handle lots of parallel calculations using thousands of cores. This approach when performing computations instead the traditional CPU on GPU is called GPGPU (General Purpose Computing on Graphics Processing Units). If a GPU is needed to be chosen for NN training and application the following qualities of the GPU needs considered. First the processing power, which means how fast a GPU can work with the NN parameters. This can be approximated as number of CUDA cores multiplied by the clock speed of each core. The second is the memory bandwidth, which is the speed of the memory. It is measured in gigabytes per second (GB/s). The third is the memory size, which is in gigabytes (GB) and determines how big batch size you can choose. If you have small memory try to train only a small batch size in order to fit the training model into memory but this will slow down the learning.

As a rule of thumb for a typical convolutional neural network 8 GB is the minimum recommended. In Table 4 various CPUs and GPUs are compared according how much training step can achieve with 224x224 images.

Table 4 - Step / hour performace on various architectures

#	Step / h	Processing unit	Memory
01.	128	Intel Xeon E5640 CPU	16 GB CPU
02.	162	Intel Xeon E5640 CPU + GeForce GTX 960 GPU	16 GB CPU 4 GB GPU
03.	2780	GeForce GTX 1070 GPU	8 GB GPU
04.	3496	GeForce GTX 1080 Ti GPU	11 GB GPU

Possible benchmarks also could contain accuracy, memory consumption, parallel scaling, but the most important is the performance which can be measured in training step / hour. In practice you either train a network from the beginning, from scratch or initialize your weights as random or - more commonly - you begin with a pretrained network and train that for your special purpose. This method is called transfer learning.

We have chosen transfer learning because all the state of the art networks provides their pretrained weights. The main deep learning frameworks such as Caffe, MXNet, TensorFlow, PyTorch, Keras, Theano, CNTK and so on provides pretrained models. There

are lot of openly available datasets which were trained for hundred or thousand iterations on benchmark datasets such as on PASCAL VOC or on CIFAR or on BSDS. At this experiment we used TensorFlow as or machine learning framework. Of course we considered other frameworks too which may more beneficial from certain point of view.

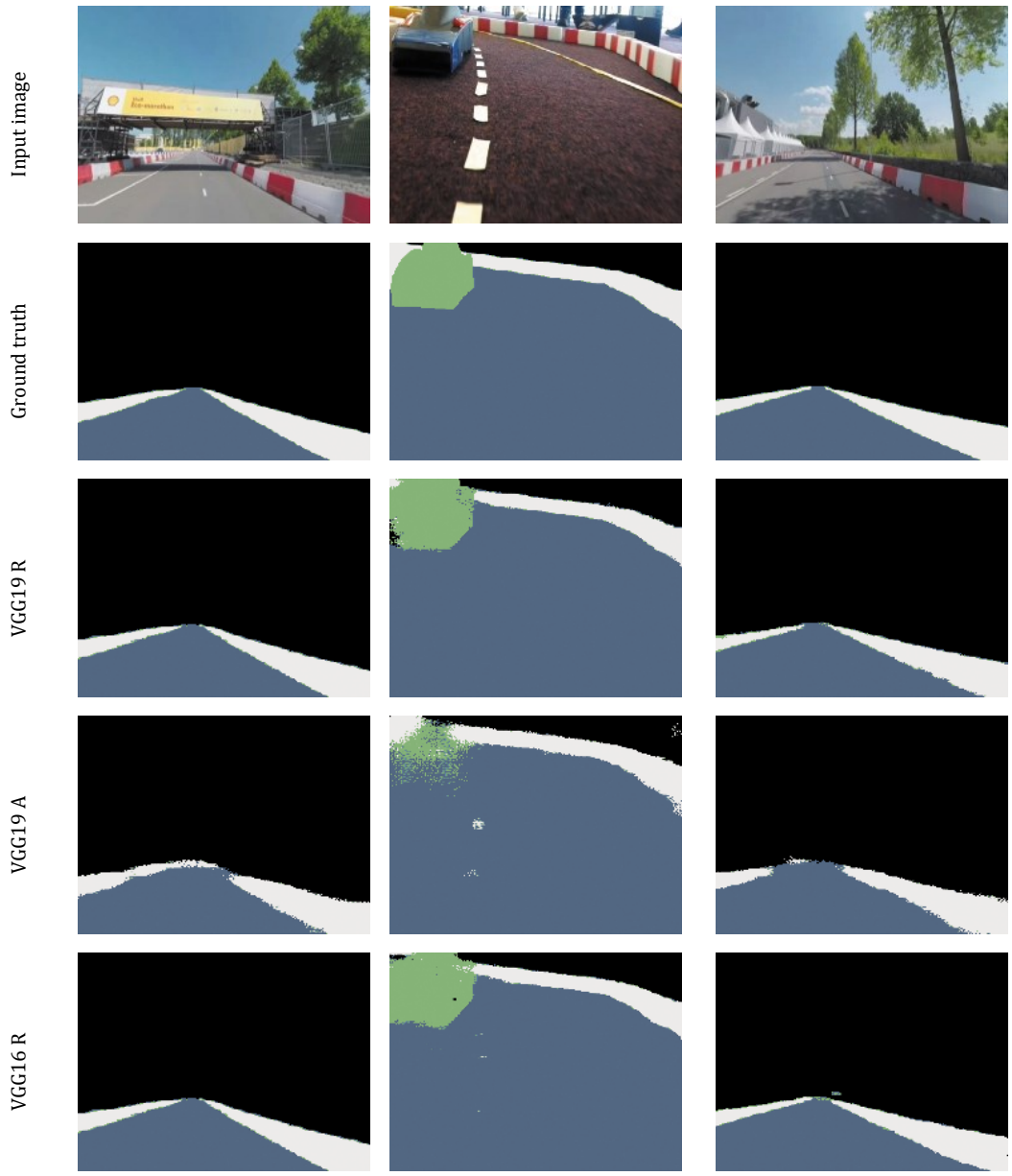


Figure 101 - Training results on different architectures (VGG-16-FCN, VGG-19-FCN) with different optimization techniques (A - Adam and R - RMSprop)

For example, MXNet is new and really fast-growing solution which has very good performance, but not the best for fine-tuning existing networks. On the other hand, Caffe is can be bulky for big networks, but it is very good for transfer learning but lack of good

visualization about the training process. TensorFlow is developed by Google and became a very popular technology specialized for deep learning. We have chosen TensorFlow because of TensorBoard for visualization, data and model parallelism, and easy-to-use API. The mentioned pretrained models are usually easily convertible to each other's deep learning framework formats, we could use for example models which were originally trained in Caffe. On the bottom of the network fully connected layers can be transformed into convolution layers, which pixelwise classification or in other words semantic image segmentation. [57] According to the observations choosing a right optimization method can shorten the training process. On figure 8 there are some results visible on different architectures with different optimization techniques. The training was done with the same architecture and with the same training step. It is visible that the 16 layered VGG-FCN with RMSprop has better results as the 19 layered version with Adam. Usually the 19 layered version has better results, but if not the best optimization technique was chosen the even the 16 layered can be better, at least at the same training steps.

3.4.3 Realization

For the sake of the experiment described in the thesis TensorFlow, Keras, PyTorch and Caffe2 was considered as a framework to rely on. TensorFlow is the definitely one of the principal machine learning framework. It has the most GitHub activity, Google searches, online job offers, books and scientific papers. The other possibility, Keras emerges from the other frameworks because of its usability for developers. Keras is rather an API standard for model definition, so it is not tied to a specific implementation. That is why for example TensorFlow Theano, or CNTK implemented it.

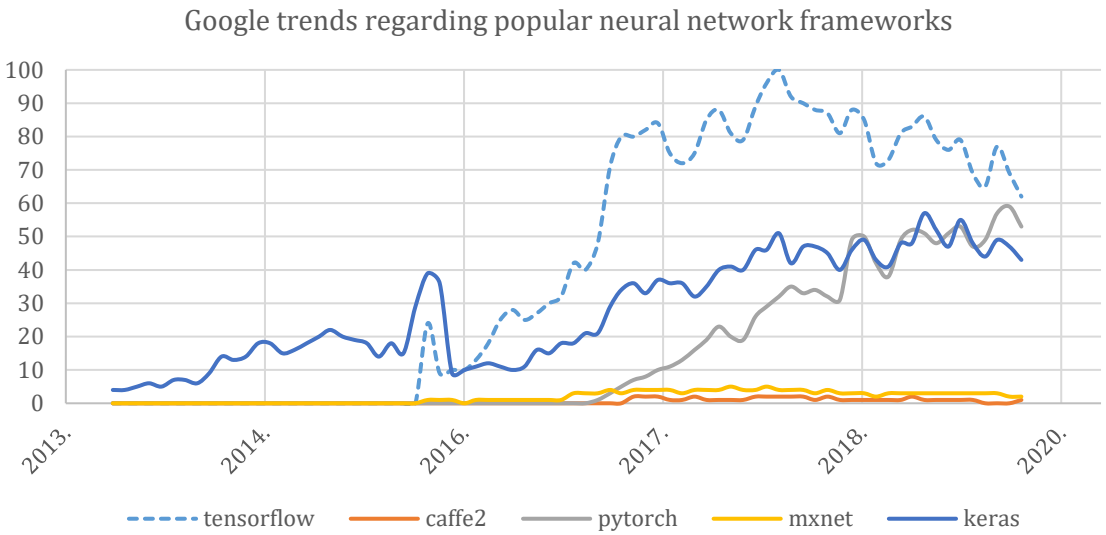


Figure 102 - Search queries of popular neural network frameworks

PyTorch is also a popular overall framework, and as mentioned it is supported by Facebook. PyTorch has a deep integration into Python which allows popular and packages to be used for writing neural network layers. Caffe is one of the oldest framework in this list, but it has recidivist recent usage. Nowadays it is slowly replaced by Caffe2, which builds on Caffe but since new computation patterns have emerged, e.g. in distributed computation some design principles of it were abandoned. All of the potential frameworks are implementing Adam, RMSprop and many other optimizers. Because of the popularity and the support, finally TensorFlow was chosen.

3.4.4 Conclusion

The goal of our work was to test different solutions in the narrow subdomain of perception for experimental vehicles. After that by collecting and analysing results we phrased recommendations for this subdomain. I recommend the following workflow. It is advised to use ReLU or leaky ReLU as activation function for FCNs. As a matter of optimization technique RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances, but in this case RMSprop is a slightly better choice despite the fact that Adam generally overperforms RMSprop.

4. Conclusions and outlook

The doctoral work is usually considered as a theoretical-focused act, where the candidate's aim to prove that he has the research skills needed in order to get a PhD. My doctoral work is no different with a slight addition that I always beard in my mind the practical application of these results. Therefore, I have chosen the topic of research and optimization of perception and path planning algorithms for mobile robots and road vehicles. This diverse topic involves robotics, mathematics, optimization and machine learning and the realization of the theorems requires computer scientist expertise too. I emphasized three aspects of my work in three theses.

In my first thesis I proposed a method to solve the Coverage Path Planning problem (CPP) called Iterative Structured Orientation Coverage (ISOC). I showed that it can lead to a shorter trajectory compared to the commonly used Boustrophedon Cellular Decomposition Coverage (BCDC). In the context of this work, I elaborated three different solutions to construct the main lines. The solution "A" and "B" relies on the rotation of parallel lines, whereas the solution "C" uses the concept of inertia. The suggested approaches are validated by simulation and experimental results. In my second thesis I showed that probabilistic occupancy grid map construction is possible to be done via the "crisp signatures" sensory model. The novelty of this approach consists of involving the concept of signatures in map construction and the elimination of Bayesian sensory, so the probabilistic characteristics of the environment model could be involved in the mapping phase. This approach is considered as another mathematical sight of sensory model realization and works well especially with time-of-flight-based sensors such as a LIDAR. The proposed approaches are validated by simulation and experimental results. In my last thesis I showed that in narrow subdomain of perception for experimental vehicles a VGG-VD neural network trained on low annotation categories with transfer learning case RMSprop converges slightly faster than Adam optimizer; despite the known fact, that that Adam generally overperforms RMSprop.

To conclude my three theses, the first one is basically a set of algorithms in other words a methodology, the second one is model and the last one is a recommendation based on a set of measurements.

5. List of figures

6. Bibliography

[1] H. Moravec, “Mind children: The future of robot and human intelligence,” *Harvard University Press,* 1988.

[2] R. Siegwart, “No need to worry!,” 2018.

[3] D. Anguelov, “Taming the Long Tail of Autonomous Driving Challenges,” Massachusetts, New England, USA, 2019.

[4] B. Siciliano and O. Khatib, Handbook of Robotics, New York, Inc. Secaucus, USA: Springer-Verlag, 2007, p. 855.

[5] C. Alvin and J. Katupitiya, “Kalman filters for the identification of uncertainties in robotic contact,” in *Proceedings of the 37th IEEE Conference on Decision and Control*, Tampa, FL, USA, 1988.

[6] J. Leopoldo, S. Longhi and G. Venturini, “Development and experimental validation of an adaptive extended Kalman filter for the localization of mobile robots,” *IEEE Transactions on Robotics and Automation,* vol. 15, no. 2, pp. 219 - 229, 1999.

[7] B. Kuipers and Y.-T. Byun, “A Robust, Qualitative Method for Robot Spatial Learning,” *Prociedings of The AAAI Conference on Artificial Intelligence,* vol. 88, pp. 774-779, 1988.

[8] C. Wang, High Performance Computing for Big Data: Methodologies and Applications, Boca Raton, FL, USA: CRC press, 2017.

[9] S. Thrun, W. Burgard and D. Fox., "A probabilistic approach to concurrent mapping and localization for mobile robots," *Autonomous Robots,* vol. 5, no. 3-4, pp. 253-271., 1998.

[10] E. Ivanjko, I. Petrovic and M. Brezak, “Experimental Comparison of Sonar Based Occupancy Grid,” *Automatika: Journal for Control, Measurement, Electronics, Computing & Communications,* vol. 50, no. 1-2., pp. 65-79, 2009.

[11] C. Pozna, N. Minculetec, R.-E. Precup, L. T.Kóczy and Á. Ballagi, "Signatures: Definitions, operators and applications to fuzzy modelling," *Fuzzy Sets and Systems,* vol. 201, pp. 86-104, 2012.

[12] M. Waanders, “Coverage path planning for mobile cleaning robots,” in *15th Twente Student Conference on IT*, The Netherlands, 2011.

[13] H. Choset, “Coverage for robotics – A survey of recent results,” *Annals of Mathematics and Artificial Intelligence,* vol. 31, no. 1, pp. 113-126, 2001.

[14] C. Zengyu, L. Shuxia, Z. Ran and Z. Qikun, “Research on Complete Coverage Path Planning Algorithms based on A* Algorithms,” *The Open Cybernetics & Systemics Journal,* vol. 8, pp. 418-426, 2014.

[15] S. X. Yang and C. Luo, “A Neural Network Approach to Complete Coverage Path Planning,” *IEEE Transactions on Cybernetics,* vol. 34, no. 1, pp. 718 - 724, 2004.

[16] G. Dudek and M. Jenkin, Computational Principles of Mobile Robotics, Cambridge: Cambridge University Press, 2010.

[17] L. Krammer, “Motion Planning for Car-like Robots,” Technische Universität Wien, 2010.

[18] E. Galceran and M. Carreras, “A Survey on Coverage Path Planning for Robotics,” *Journal Robotics and Autonomous Systems,* vol. 61, no. 12, pp. 1258-1276, 2013.

[19] H. Choset and P. Pignon, “Coverage Path Planning: The Boustrophedon Decomposition,” in *International Conference on Field and Service Robotics*, Canberra, 1997.

[20] B. Siciliano and O. Khatib, Springer Handbook of Robotics, Berlin: Springer Berlin Heidelberg, 2008.

[21] J.-C. Latombe, “Robot Motion Planning,” *Kluwer international series in engineering and computer science,* 1990.

[22] S. Thrun, W. Burgard and D. Fox, Probabilistic Robotics (Intelligent Robotics and Autonomous Agents), The MIT Press, 2005.

[23] W. Hess, D. Kohler, H. Rapp and D. Andor, "Real-Time Loop Closure in 2D LIDAR SLAM," *IEEE International Conference on Robotics and Automation (ICRA),* pp. 1271-1278, 2016.

[24] F. Rosique, P. J. Navarro, C. Fernández and A. Padilla, “A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research,” *MDPI Sensors,* vol. 3, no. 19, p. 648, 2019.

[25] L. Fridman, “Deep learing for Self-Driving Cars lecture,” Massachusetts, New England, USA, 2019.

[26] B. Paden, M. Cap, S. Z. Yong, D. Yershov and E. Frazzoli, “A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles,” *IEEE Transactions on Intelligent Vehicles,* vol. 1, pp. 33-35, 2016.

[27] A. Ollero, A. García-Cerezo and J. Martinez, “Fuzzy supervisory path tracking of mobile reports,” *Control Engineering Practice,* vol. 2, no. 2, pp. 313-319, 1994.

[28] M. Samuel, M. Hussein and M. B. Mohamad, “A review of some pure-pursuit based path tracking techniques for control of autonomous vehicle,” *International Journal of Computer Applications,* vol. 135, no. 1, pp. 35-38, 2016.

[29] A. Rodríguez-Castaño, G. Heredia and A. Ollero, “Analysis of a GPS-based fuzzy supervised path tracking system for large unmanned vehicles,” *IFAC Proceedings Volumes,* vol. 33, no. 25, pp. 125-130, 2000.

[30] S. Kato, S. Tokunaga, Y. Maruyama, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii and T. Azumi, “Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems,” in *ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018.

[31] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin and A. Paques, “PythonRobotics: a Python code collection of robotics algorithms,” *Computing Research Repository (CoRR),* vol. 1808.10703, 2018.

[32] J. Giesbrecht, D. Mackay, J. Collier and S. Verret, "Path Tracking for Unmanned Ground Vehicle Navigation: Implementation and Adaptation of the Pure Pursuit Algorithm," *Defence Research and Development Suffield (Alberta),* 2005.

[33] E. Horváth, C. Hajdu and P. Kőrös, “Novel Pure-Pursuit Trajectory Following Approaches and their Practical,” in *10th IEEE International Conference on Infocommunications*, Naples, Italy, 2019.

[34] B. Paden, M. Cap, S. Z. Yong, D. Yershov and E. Frazzoli, “A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles,” *IEEE Transactions on intelligent vehicles,* vol. 1, no. 1, pp. 33-55, 2016.

[35] D. S. Lal, A. Vivek and G. Selvaraj, “Lateral control of an autonomous vehicle based on Pure Pursuit algorithm,” in *International Conference on Technological Advancements in Power and Energy*, Kollam, India , 2017.

[36] M. Samuel, M. Hussein and M. Binti, “A Review of some Pure-Pursuit based Path Tracking Techniques for Control of Autonomous Vehicle,” *International Journal of Computer Applications,* vol. 135, no. 1, pp. 35-38, 2016.

[37] J. Kong, M. Pfeiffer, G. Schildbach and F. Borrelli, “Kinematic and dynamic vehicle models for autonomous driving control design,” in *IEEE Intelligent Vehicles Symposium (IV)*, Seoul, South Korea, 2015.

[38] D. Kriesel, A Brief Introduction to Neural Network, 2007.

[39] D. A. Medler, “A Brief History of Connectionism,” *Neural Computing Surveys,* vol. 1, p. 61–101, 1998.

[40] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[41] A. Karpathy, *Convolutional Neural Networks for Visual Recognition,* 2017.

[42] Kaggle, “Convolutional Nets and CIFAR-10: An Interview with Yann LeCun.,” 2014.

[43] Y. LeCun, L. Bottou, Y. Bengio and P. H. , “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE,* vol. 86, no. 11, p. 2278–2324., 1998.

[44] Y. LeCun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard and W. Hubbard, “Handwritten Digit Recognition: Applications of Neural Net Chips and Automatic Learning,” *IEEE Communication,* pp. 41-46, 1989.

[45] A. Mehdi, G. Liu, M. P. Wittie and C. Izurieta, “An experimental evaluation of apple siri and google speech recognition,” in *Proccedings of the 2015 ISCA SEDE*, 2015.

[46] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition (VGG-VD),” *The Computing Research Repository (CoRR),* 2014.

[47] A. Krizhevsky, I. Sutskever and G. E. Hinton, “Imagenet classification with deep convolutional neural networks (AlexNet),” in *Neural Information Processing Systems Conference (NIPS)*, 2012.

[48] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, “Going deeper with convolutions (GoogLeNet),” *The Computing Research Repository (CoRR),* 2014.

[49] F. Chollet, Deep Learning with Python, Shelter Island, New York, USA: Manning Publications Co. , 2018.

[50] M. A. Nielsen, Neural networks and deep learning, USA: Determination press, 2015.

[51] I. N. d. Silva, D. H. Spatti, R. A. Flauzino, L. H. B. Liboni and S. F. d. R. Alves, Artificial Neural Networks - A Practical Course, Switzerland: Springer International Publishing, 2017.

[52] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological review,* vol. 65, no. 6, p. 386, 1958.

[53] J. Hopfield., "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences of the United States of America,* vol. 79, p. 2554–2558, 1982.

[54] A. Karpathy and e. al., Writers, *Convolutional Neural Networks for Visual Recognition notes accompany the Stanford class CS231.* [Performance]. 2017.

[55] O. Russakovsky, H. S. Jia Deng, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV),* vol. 115, no. 3, pp. 211-252, 2015.

[56] C. Szegedy, A. Toshev and D. Erhan, "Deep neural networks for object detection," *Advances in Neural Information Processing Systems,* pp. 2553-2561, 2013.

[57] J. Long, E. Shelhamer and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, UC Berkeley, 2015.

[58] V. Badrinarayanan, A. Kendall and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 2017.

[59] J. Long, E. Shelhamer and T. Darrell, "Fully convolutional networks for semantic segmentation," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* pp. 3431-3440, 2015..

[60] H. Noh, S. Hong and B. Han, "Learning Deconvolution Network for Semantic Segmentation," *Computing Research Repository ,* vol. 1505, no. 04366, pp. 11520-11528, 2015.

[61] B. Rohrer, “Getting closer to human intelligence through robotics,” 2018.

[62] J. Ito, “The Limits of Explainability,” Condé Nast Publications, San Francisco, California, USA, 2018.

[63] G. Elsayed, S. Shankar, B. Cheung, N. Papernot, A. Kurakin, I. J. Goodfellow and J. Sohl-Dickstein, “Adversarial Examples that Fool both Human and Computer Vision,” *CoRR Computing Research Repository, arXiv,* no. 1802.08195 , 2018.

[64] A. Lee, “Comparing Deep Neural Networks and Traditional Vision Algorithms in Mobile Robotics,” in *Swarthmore College*, 2016.

[65] J. Condliffe, “MIT Technology Review,” 2019. [Online].

[66] D. Bau, J.-Y. Zhu, H. Strobelt, B. Zhou, J. B. Tenenbaum, W. T. Freeman and A. Torralba, “GAN Dissection: Visualizing and Understanding Generative Adversarial,” *Computing Research Repository (CoRR),* vol. abs/1811.10597, 2018.

[67] X. Zhang, Y. Wang and W. Shi, “pCAMP: Performance Comparison of Machine Learning Packages on the Edges,” in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*, Boston, MA, 2018.

[68] D. L. Applegate, R. E. Bixby, V. Chvátal and W. J. Cook, The Traveling Salesman Problem:, Princeton, USA: Princeton Univerity Press, 2006.

[69] P. E. Missiuro, Map represents 3rd floor common area of the MIT Stata Center, http://dspace.mit.edu/handle/1721.1/62269: 2010.

[70] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A. Y. Ng, “ROS: an open-source Robot Operating System cite,” *ICRA workshop on open source software,* vol. 3, no. 2, 2009.

[71] S. Thrun, “Learning Occupancy Grid Maps with Forward Sensor Models,” *Autonomous Robots,* vol. 15, pp. 111-127, 2003.

[72] L. Tai and M. Liu, "Deep-learning in Mobile Robotics - from Perception to Control Systems: A Survey on Why and Why not," *CoRR, Computing Research Repository,* 2016.

[73] Ł. Kaiser, A. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones and J. Uszkoreit, “One Model To Learn Them All,” *CoRR, Computing Research Repository,* 2017.

[74] D. G. Lowe, “Object recognition from local scale-invariant features,” *The proceedings of the seventh IEEE international conference on Computer vision,* vol. 2, pp. 1150-1157, 1999.

[75] H. Bay, T. Tuytelaars and L. V. Gool, “Surf: Speeded up robust features,” *European conference on computer vision,* pp. 404-417, 2006.

[76] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, “ORB: An efficient alternative to SIFT or SURF,” *IEEE international conference on Computer Vision (ICCV),* pp. 2564-2571, 2011.

[77] C. C. Hau, Handbook Of Pattern Recognition And Computer Vision, Singapore: World Scientific, 2015.

[78] Y. Bengio, “Learning deep architectures for AI,” *Foundations and trends in Machine Learning,* vol. 2, no. 15, pp. 1-27, 2009.

[79] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[80] S. Ruder, “Ruder, Sebastian. "An overview of gradient descent optimization algorithms,” *arXiv preprint,* 2016.

[81] J. Duchi, E. Hazan and Y. S. ". 1. (2011):, “Adaptive subgradient methods for online learning and stochastic optimization.,” *Journal of Machine Learning Research,* vol. 12, no. Jul, pp. 2121-2159., 2011.

[82] J. Duchi and Y. Singer, “Efficient online and batch learning using forward backward splitting,” *Journal of Machine Learning Research,* vol. 10, no. Dec, pp. 2899-2934., 2009.

[83] D. P. Kingma and J. L. Ba, “Adam: a Method for Stochastic Optimization,” *International Conference on Learning Representations,* p. 1–13, 2015.

[84] T. Tieleman, G. Hinton, N. Srivastava and K. Swersky, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *Coursera: Neural networks for machine learning,* pp. 26-31, 2012.

[85] V. Badrinarayanan, A. Kendall and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 2017.

[86] J. L. Elman, “Finding structure in time,” *Cognitive science,* vol. 14, no. 2, pp. 179-211., 1990.

[87] H.P.Moravec and A.E.Elfes, “Highresolution mapsfrom wide angle sonar,” *Proceedings IEEE International Conference Robotics Automation (ICRA),* 1985.

[88] A. L. Maas, A. Y. Hannun and A. Y. Ng., “Rectifier nonlinearities improve neural network acoustic models,” *Proceedings International Conference on Machine Learning,* vol. 30, no. 1, 2013.

[89] “Google Cartographer Github repository,” Cartographer Authors, [Online]. Available: https://github.com/googlecartographer/cartographer.

[90] K. M. o. Sameer Agarwal, “Ceres Solver,” [Online]. Available: http://ceres-solver.org.

[91] OSRF, “Robot Operating System,” Open Source Robotics Foundation, [Online]. Available: http://wiki.ros.org/.

[92] F. M. Mathieu Labbé, “Long-term online multi-session graph-based SPLAM with memory management,” *Autonomous Robots,* pp. 1133-1150, 2018.

[93] F. M. Mathieu Labbé, “Online Global Loop Closure Detection for Large-Scale Multi-Session,” *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems,* pp. 2661-2666, 2014.

[94] F. M. Mathieu Labbe, “Appearance-Based Loop Closure Detection for Online Large-Scale and,” *IEEE Transactions on Robotics,* vol. 29, no. 3, pp. 734-745, 2013.

[95] M. Labbe, “RTAB-Map ROS wiki page,” [Online]. Available: http://wiki.ros.org/rtabmap.

[96] M. Hosseinzadeh, K. Li, Y. Latif and I. Reid, “Real-Time Monocular Object-Model Aware Sparse SLAM,” 2018.

[97] S. Prakash and G. Gu, "Simultaneous Localization And Mapping with depth Prediction using Capsule Networks for UAVs," 2018.

7. Own publications

[H1] C Pozna, E Horváth, J Kovács: *Developing rapid prototype-capable applications for industrial mobile robot platforms* IEEE, 18th International Conference on Intelligent Engineering (INES) Tihany, Magyarország, 2014

[H2] E Horváth, J Hollósi: *Kutatási célú gyorsprototípus alkalmazások fejlesztése Neobotix MP500 mobilrobot rendszerre* EMT XIII. OGÉT, Nemzetközi Gépészeti Találkozó Şumuleu Ciuc, Csíksomlyó, Románia, 2015

[H3] C Pozna, E Horváth, J Kovács: *An abstraction of the Lidar measurements* 10th Jubilee IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI) Timişoara, Temesvár, Románia, 2015

[H4] E Horváth, P Kőrös, I Lakatos, P Dely: *Development of individual information technology systems of experimental vehicles* 10th Jubilee IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI) Timişoara, Temesvár, Románia, 2015

[H8] C Pozna, E Horváth, J Hollósi: *The inverse kinematics problem, a heuristical approach* IEEE 14th International Symposium on Applied Machine Intelligence and Informatics (SAMI) Herl'any, Szlovákia, 2016

[H9] E Horváth, C Pozna, C Hajdu, J Hollósi: *A use case of the simulation-based approach to mobile robot algorithm development* IEEE 14th International Symposium on Applied Machine Intelligence and Informatics (SAMI) Herl'any, Slovakia, 2016

[H10] E Horváth, P Kőrös, I Lakatos: *A Case Study of Software Developments in Electric Experimental Vehicles* Scientific Bulletin of the "Politehnica" University of Timisoara, Transactions on Automatic Control and Computer Science, ISSN 1224-600X Timişoara, Temesvár, Románia, 2015

[H11] C Pozna, E Horváth: *Predictive Distributed Formation Control for Swarm Robots Using Mobile Agents* Scientific Bulletin of the "Politehnica" University of Timisoara, Transactions on Automatic Control and Computer Science, ISSN 1224-600X Timişoara, Temesvár, Románia, 2016

[H12] E Horváth, C Pozna, H Somogyi: *Járműveken és mobil robotokon alkalmazható "foglaltsági rács" alapú térképépítési eljárás* EMT XIV. OGÉT, Nemzetközi Gépészeti Találkozó Deva, Déva, Romania, 2016

[H13] H Somogyi, E Horváth: *Autonóm járművek navigációs rendszerének minimális költségű szenzorokkal való megvalósíthatóságának elemzése* EMT XIV. OGÉT, Nemzetközi Gépészeti Találkozó Deva, Déva, Romania, 2016

[H14] E Horváth, C Pozna: *Probabilistic Occupancy Grid Map Building for Neobotix MP500 Robot* IEEE WPNC'16 IEEE 13th Workshop on Positioning, Navigation and Communications Bréma, Bremen, Germany, 2016

[H15] E Horváth, P Kőrös: *Alacsony energiafelhasználású villamos hajtású prototípus jármű identifikációs mérései és fejlesztési kérdései* EMT XXV. OGÉT, Nemzetközi Gépészeti Találkozó , 243-246. Kolozsvár, Románia, 2017

[H16] E Horváth, C Pozna, Á Ballagi: *Road Recognition using Fully Convolutional Neural Networks* 3rd International Conference for Doctoral Students - IPC, Braşov, Romania, 2017

[H17] E Horváth, P Kőrös, I Lakatos, F Szauter: *Two Operating States-Based Low Energy Consumption Vehicle Control* 13th ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications Cleveland, Ohio, USA, 2017

[H18] E Horváth, P Kőrös, I Lakatos: *Két üzemen alapuló energiahatékony járművezérlés megvalósítása kísérleti járművön* Innováció és fenntartható felszíni közlekedés konferencia (IFFK) Budapest, Magyarország, 2017

[H19] E Horváth, C Pozna, R E Precup: Robot coverage path planning based on iterative structured orientation Acta Polytechnica Hungarica, Volume 15, Issue 2, DOI: 10.12700/APH.15.1.2018.2.12, Impact factor: 0.745, 2018

[H20] J Hollósi, E Horváth, C Pozna: *Two-stage Racetrack Segmentation Method using Color Feature Filtering and Superpixel-based Convolutional Neural Network* 12th IEEE International Symposium on Applied Computational Intelligence and Informatics Temesvár / Timişoara, Románia, 2018

[H21] E Horváth, C Pozna, Á Ballagi: *Evaluation of Neural Network-based Sensing and Perception in Experimental Vehicle*s 22nd IEEE International Conference on Intelligent Engineering Systems (INES2018) Las Palmas de Gran Canaria, Spain, 2018

[H23] E Horváth, Cs Hajdu, C Pozna, Á Ballagi: *Range Sensor-based Occupancy Grid Mapping with Signatures* 20th International Carpathian Control Conference ICCC Poland , 2019

[H24] E Horváth, C Pozna, Á Ballagi: *A Mobile Robot and Vehicle Occupancy Map Construction Model* - 23nd IEEE International Conference on Intelligent Engineering Systems (INES2019)- INES Hungary, 2019

[H25] E Horváth, Cs Hajdu, C Pozna, Á Ballagi: Range Sensor-based Occupancy Grid Mapping with Signatures 2019 20th IEEE International Carpathian Control Conference (ICCC), Kraków-Wieliczka, Poland, 2019

[H26] E Horváth, C Pozna, Á Ballagi: *Novel Pure-Pursuit Trajectory Following Approaches and their Practical Applications* 10th IEEE International Conference on Infocommunications, Naples, Italy, 2019

[H27] E Horváth, Cs Hajdu, C Pozna: *Enhancement of pure-pursuit path-tracking algorithm with multi-goal selection* IEEE International Conference on Gridding and Polytope Based Modeling and Control, Győr, Hungary, 2019

[H28] E Horváth, C Pozna, P Kőrös, Cs Hajdu, Á Ballagi: *Theoretical background and application of multiple goal pursuit trajectory follower*, Hungarian Journal of Industry and Chemistry, 2020