



University of
St Andrews

FOUNDED
1413

SCHOOL OF COMPUTER SCIENCE

Master of Science in Artificial Intelligence

CREATIVE AUTONOMOUS NAVIGATION BY MOBILE ROBOTS IN UNSTRUCTURED SPACES

Postgraduate Thesis

AUTHOR

Pavel Sobotka

SUPERVISOR

Dr Mike Weir

ST ANDREWS 2021

ABSTRACT

The goal of mobile robotics is to navigate robots in the environment without colliding with obstacles. Numerous methods deal with tackling this task where one of them is the Potential field approach. However, this method is prone to the Local Minima problem; therefore is not able to solve obstacles of high difficulty, such as c-shape obstacles.

This dissertation project developed a novel approach called the Reverse anglerfish method for mobile robot navigation to plan a safe and efficient complete route to the goal. This method also introduced a new concept of the Progress field. It tackles the problem of the local minima without collisions using the line of least resistance. It is able to solve obstacles of different levels of difficulty. The algorithm does not require access to the global map, so it is suitable for the robot moving in an unknown environment where the only known information is the goal coordinates beforehand. Thus, it is fully dependant on the local sensor. Its applicability is proved both in 2-D and 3-D robot's space and has the potential to be used in N-D applications as well. Furthermore, the user-friendly graphical user interface for setting the navigation parameters was developed, and the algorithm was incorporated and tested on the physical robot.

ACKNOWLEDGEMENTS

First, I would like to extend my sincere thanks to my dissertation supervisor Dr Mike Weir. Thank you for your invaluable insights, weekly consultations, willingness, friendly cooperation, approach and enthusiasm that motivated me throughout.

Thank you to my family and friends for the moral support, especially to my sister and her husband for often video calls and late-night Brawl Stars games. Thank you to my friend Honzik who spent this year-long experience with me.

Last but certainly not least, there is a special thanks to my Mom and Dad for their unconditional support throughout my entire studies. I could not have done it without you.

DECLARATION OF AUTHORSHIP

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is 13,640 words long. In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker and to be made available on the World Wide Web. I retain the copyright in this work.

Date: 17th August 2021

Pavel Sobotka

A handwritten signature in black ink, appearing to be 'P. Sobotka', written in a cursive style.

CONTENT

ABSTRACT.....	3
ACKNOWLEDGEMENTS	5
DECLARATION OF AUTHORSHIP	6
CONTENT.....	7
1 BACKGROUND	9
1.1 Motivation	9
1.2 Mobile robots.....	9
1.2.1 Locomotion.....	10
1.2.2 Perception	11
1.2.3 Cognition	13
1.2.4 Navigation	14
1.2.5 TurtleBot2.....	17
1.3 ROS (Robotic Operation System)	18
1.4 Optimisation – Finding minimum of a function	19
1.4.1 Line of Least resistance	19
1.5 Potential field approach	22
1.5.1 Local Minima Problem	22
1.6 FM2 approach.....	23
2 DESIGN AND IMPLEMENTATION	24
2.1 Setup	24
2.2 Motion planning	24
2.2.1 Differential drive	24
2.2.2 Localisation	25
2.2.3 Motion control	26
2.3 Navigation	27
2.3.1 Control contour.....	27
2.3.2 Perception	28
2.3.3 Optimisation of subgoal calculation	30
2.4 Potential field.....	31
2.5 Progress field	33
2.5.1 Key finding	37
2.5.2 Reverse anglerfish with deviating lantern point method	38
2.6 Applicability of Reverse anglerfish algorithm for 3-D space travel.....	49
2.7 Using Navigation GUI.....	51
2.8 Incorporation into a physical robot.....	52
3 EXPERIMENTS AND EVALUATION	53
4 CONCLUSION AND FUTURE WORK	55
4.1 Suggested future work and improvements	55
4.2 Challenges and achievements	56
5 APPENDIX 1 – EXPERIMENTS AND ROBUSTNESS TESTS.....	58
5.1 Experiment 1: No obstacles, starts facing the goal.....	58
5.2 Experiment 2: No obstacles, starts facing away from the goal	59
5.3 Experiment 3: Single cube obstacle.....	60
5.4 Experiment 4: A wall with a hole obstacle.....	61

5.5	Experiment 5: Random cube maze	62
5.6	Experiment 6: Mild c-shape obstacle	64
5.7	Experiment 7: A long wall obstacle	65
5.8	Experiment 8: Medium c-shape obstacle	66
5.9	Robustness of Reverse anglerfish method	67
5.9.1	Robustness test 1: V-shape obstacle	68
5.9.2	Robustness test 2: Severe c-shape obstacle.....	69
5.9.3	Robustness test 3: C-shape snail obstacle	70
5.9.4	Robustness test 4: Deviated c-shape snail obstacle	71
5.9.5	Robustness test 5: Obstacles inside c-shape snail obstacle.....	72
5.9.6	Robustness test 6: Goal closed in the box (inside reversed c-shape).....	73
5.9.7	Robustness test 7: Reversed closed c-shape snail obstacle.....	75
5.9.8	Robustness test 8: Start inside spiral.....	76
5.9.9	Robustness test 9: Start inside c-shape snail obstacle.....	78
5.9.10	Robustness test 10: V-shape inside the snail	79
6	APPENDIX 2 - PHYSICAL TURTLEBOT IMPLEMENTATION	82
	ETHICS.....	84
	REFERENCES.....	85

1 BACKGROUND

1.1 Motivation

Researching the framework introduced in the FM² paper in the unstructured world for autonomous navigation and learning by mobile robots is motivated by the novelty of the application of this approach in the unstructured world and potential use in robot learning. The goal is to modify the contour-generating model for an environment that is only known dynamically through the robot's local sensors and, as an effect, get rid of the robot's need for a global map. Further, to make the novel approach effective in N-D, $N > 2$, to cater for incorporation into air-based or multi-joint robots. Lastly, the N-D system may also be shown to be relevant to a search for navigation to useful learning states in the robot brain.

1.2 Mobile robots

Robotics is a booming interdisciplinary field that incorporates science, engineering and technology. Its goal is to produce programmable machines, called robots, that can assist humans or mimic human actions. A robot is a physical agent that interacts with the physical world through effectors used to develop physical action on the environment. Effectors can be in a form such as legs, wheels, joints, and grippers. The robots are also equipped with sensors such as cameras, lasers, gyroscopes and accelerometers, allowing them to perceive their environment. Robots have countless utilisation and are often used in a dangerous environment (bomb detonations), an environment where humans could not survive (space exploration) or manufacturing (repeated work or handling of heavy loads). [1]

Mobile robots are one of the main subfields of robotics that deals with robots that travel around their environment. Effectors allow them to move; thus, they are not fixed to one location, unlike manipulators such as industrial robots. Other components are sensors, a control system (e.g., embedded microcontroller, a microprocessor, or a personal computer), and power systems. The Control system coordinates all the subsystems that form the robot, and the power system supplies the robot with electric power. [1, 2]

According to [3] mobile robots market is rapidly growing, especially in the home sector, where autonomous robots are used for tasks such as cleaning the house (autonomous vacuum cleaner) or moving the lawn.

Autonomous mobile robots are mobile robots capable of moving and navigating in the environment without any guidance from external human operators. Contrary to human-controlled robots, they demonstrate the ability to react, make decisions and perform actions based on the perception they receive from the environment. Operating in an unpredictable and partially unknown environment, they must navigate without

disruption and be able to avoid abrupt obstacles that appear in their planned path. A control unit of a mobile robot has to be integrated with four main systems to overcome these challenges and fulfil its objectives without human interaction in the real world. These systems are locomotion, perception, cognition, and navigation. [1, 2]

1.2.1 Locomotion

Locomotion expresses the ability to transport or move from one place to another; doing it is the first challenge that an autonomous mobile robot has to overcome. One has to take into account mechanism and kinematics, dynamics and control theory to solve the locomotion problem. The locomotion system may depend on the environment in which the robot is moving. This could vary from an industrial plant, laboratory, museum, different medium (e.g. underwater, in the air), but also extremely hostile environments like during space exploration (e.g. NASA's Mars Perseverance rover on the surface of the Red Planet). The design of a mobile robot also depends on factors such as manoeuvrability, controllability, terrain conditions, efficiency, stability, and others. Depending on the robot's purpose, it can mostly walk, roll, skate, run, jump, slide, swim and fly. [2, 4, 5]

According to [2], mobile robots can be classified into the following categories depending on their locomotion system.

Stationary robots are mainly industrial robots with a fixed base performing tasks such as welding, painting, or assembling.

Land-based robots are further divided into subcategories, including wheeled, walking, tracked, and hybrid mobile robots. Wheels are easy to use, cheap and do not raise stability problems. Therefore, they are widely used in autonomous intelligent vehicles (AVIs). On the other hand, wheels are not suitable for navigating over obstacles, such as rocky terrain. For this scenario, walking robots equipped with legs are a great solution. They are energy efficient with good mobility and the ability to move on uneven terrain. Sometimes the need for a robot's versatility leads to creating a hybrid that, for example, has both wheels and legs.

Air-based robots, nowadays commercially popular unmanned aerial robots (drones), can take off independently, land and return to their starting position.

Water-based robots are submarine-like robots that are operating underwater, mainly used for seabed exploration.

The last group is *other robots* that are hard to classify, for example, nanorobots, snake-like and cooperative robots. [2]

Autonomous mobile robots can be used in countless applications like planetary exploration, emergency rescue operations, industrial automation, and entertainment.

1.2.2 Perception

An autonomous robot needs to acquire knowledge about its environment and its internal condition. To percept objects around itself or its relative position, the robot must be mounted with sensors extracting relevant measurements from the surroundings. Accurate estimation of the object's position in the environment enables autonomously performing robot localisation and positioning tasks. Sensors are also used for object recognition, data collection, mapping and representation. Through the sensors, robot gains access to the information about the world and applications like drones and automated driverless cars are impossible without them. [2, 5]

Proprioceptive and exteroceptive sensors can be distinguished, where proprioceptive sensors, in the form of encoders, potentiometers, gyroscopes, compasses, measure internal values of the robot such as battery level, wheel position, joint angel, motor speed, and others. Exteroceptive sensor, on the other hand, supplies the robot with information about the environment, such as distances, light intensity, and sound amplitude. These sensors can be sonar sensors, infrared (IR) sensitive sensors, ultrasonic distance sensors. [2, 5]

Sensors can be further recognised as passive or active. Passive sensors are sensing the ambient energy of the environment measured by microphones, touch sensors, thermometers, and passive cameras using Complementary Metal Oxide Semiconductor (CMOS) or Charge Coupled Device (CCD). On the contrary, active sensors directly interact with the environment by emitting energy into the environment and then measuring its impact on the surroundings. By this technique, sensors can gain high-quality data from the environment and thus enhance the overall performance. Nevertheless, they may suffer from signal interference from external signals. [2, 5]

The three-dimensional world is compressed into a two-dimensional image when using the conventional 2-D camera. The depth sensors, also called 3-D cameras, are used to overcome this problem. They are a good example of the exteroceptive and active sensor used in object detection, scene reconstruction, 3-D inspection, etc. They provide a robot with depth information that enhances the interpretation of the world. [6]

In 2009 Microsoft announced the Kinect sensor for its Xbox game console that senses the user's movement in front of the screen and with a microphone that understood voice commands. Primarily designated for gaming purposes, the Kinect camera was used to solve the simultaneous localisation and mapping (SLAM) problem in 2011. This lead to the development of a software development toolkit (SDK) for Kinect. [7]

Kinect version one camera is equipped with a depth sensor, a colour camera and an array of microphones, all enclosed in a small plastic black case. The housing is attached to the motor, which allows the device to be tilted in the horizontal direction. All components can be seen in figure 1. [7]

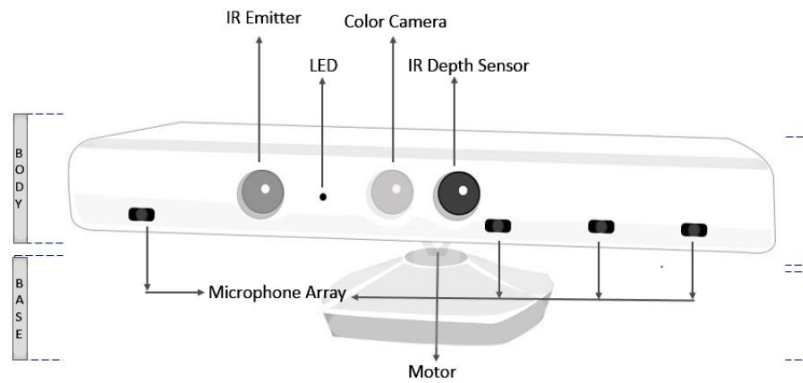


Figure 1: Kinect camera components [7]

Microphone array allows it to apply voice recognition and a colour camera is responsible for capturing and transmitting colour video data in RGB format. The depth sensor consists of an infrared (IR) emitter and an IR camera. The infrared projector continually emits an infrared light pattern, which is read by the depth sensor. By composing two images, emitted pattern and an image read by the IR sensor, it calculates the depth information using the Stereo Triangulation method. [7, 8]

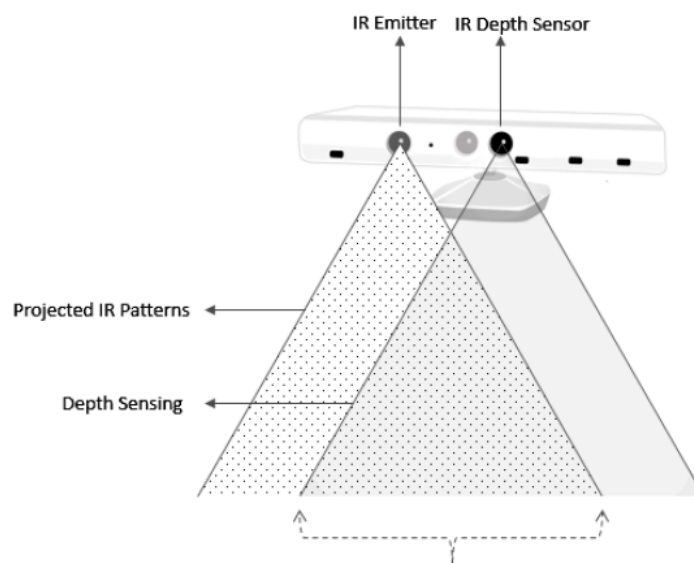


Figure 2: Composing two images (emitted pattern and an image read by the IR sensor) [7]

The result of this scanning is a depth image (figure 3) of which each pixel contains a number that represents the distance of an object in front of the sensor in millimetres. [7, 8]



Figure 3: Depth image [8]

1.2.3 Cognition

Cognition is in charge of the control system of a mobile robot. It is responsible for achieving the robot's goals by processing and analysing the data received from sensors and taking the corresponding actions. [2]

Perception, processing and cognition, and action are the three main tasks of the control system. By the perception, the robot derives the information about the environment itself, as mentioned above. The sensed information is further processed and evaluated, and then the relevant commands are sent to actuators that move the mechanical structure of the robot. When all the information about the environment, robot's direction, destination, or purpose are gathered, the cognitive architecture then focuses on achieving high-level objectives by planning the robot's path. In addition, the control system is responsible for combining all input data and planning the robot's motion to move properly. The robot may need a cognitive model to represent the robot, environment, and how they interact. These models can be created by algorithms of artificial intelligence (AI) that play a significant role in the decision-making and execution part. [2]

Computer vision (CV) is a relatively young interdisciplinary field that is solving a computer's understanding of digital images or videos. In mobile robotics, it is used to recognise patterns and track objects. The most significant breakthrough moment in CV history happened in 2012 when the researchers from the University of Toronto completely changed standards by introducing a convolutional neural network (CNN) called AlexNet. CNN is a deep neural network that can work with dense data and become the standard in image recognition. [6]

Further, constructing and updating a map of an unknown environment that can be crowded and noisy is an essential problem in autonomous mobile robots. Mapping algorithms solve this computational problem. The two most used algorithms are point-based matching and feature-based matching. In the point-based technique, measured

data points are directly used for mapping. Its disadvantage is a requirement for a large memory that diminishes the overall performance, especially in large environments. The second technique transforms the raw measurements data into meaningful geometric features, which can be used for matching and scan corrections. In contrast with the first method, it does not require as much memory; thus, it is more efficient and even provides more information about the environment. Furthermore, the extracted features can be used as building blocks for both mapping and localisation and are therefore used for already mentioned SLAM problems. [9]

Finally, motion planning and other AI algorithms are used to plan how the robot should behave in certain situations and how to achieve its objectives without colliding with obstacles, without falling over, overcoming uneven surfaces, etc. Motion planning controls and optimises the actual movement that the robot has to complete to progress on its path towards the goal. This might be, for example spinning the wheels, braking, or raising the robot's legs. The robot must imply forces on its actuators to achieve the desired trajectory. [2, 10]

Control strategies are based on tracking the error in order to evaluate the performance of the control system. There are many different techniques for calculating and tracking this error; the best-known are, for instance, computed torque control, robust control, adaptive, and neural networks methods. [2]

1.2.4 Navigation

The navigation skill is probably the most important aspect of autonomous mobile robots. Autonomous navigation aims to get the robot from one place to another in both known or unknown environments (depending on whether the robot is provided with the global map – known environment) by considering the perceptions from the sensors. In order to do that robot must know where it is at present, where is the goal location, and how to get there. This implies that the navigation task is dependent on other aspects such as perception, cognition, localisation, and motion control. Successful application of navigation has to perform generating a map of the environment, computing a collision-free path to the goal, and avoiding collisions with obstacles while moving along the calculated path. It brings challenges to overcome where the most recognised is path planning, localisation and obstacle avoidance. [2, 5]

These challenges are subsequently discussed where the relevant environment (in which mobile robot operates) to this project is: partially observable, single agent, stochastic, sequential, unstructured, dynamic, continuous, and unknown. That means that the robot does not possess the global map and is thus completely dependent on the local sensors of the robot. It also has to be able to react to changing environments due to its dynamic nature.

Path Planning

As the robot cannot always take the direct path to the goal, a path planning technique must be implemented. Path planning is solely focused on enabling the robot

to navigate the goal configuration around the obstacles and find the best path while neglecting velocities and accelerations. Here it is good to notice a difference between path planning and obstacle avoidance. The path planning needs an environment model or map, a goal location, and the robot's current location so it can plan a path to reach the target, while obstacle avoidance is fully dependent on the local sensors. [2, 10]

According to [2], path planning algorithms can be assigned to five broad categories, i.e. classical methods, probabilistic methods, heuristic planners, evolutionary algorithms, and sensor-based planners. Two of the foremost classical approaches are cell decomposition and skeletonisation. The real world is a continuous space, and these methods tackle this problem by reducing it to a discrete graph search problem. [1, 2]

The cell decomposition method is based on continuous space decomposition into the cells. The mean value can be then calculated for each cell which transforms the planning problem into a discrete-graph problem. That allows the use of search algorithms such as A*. The second method is based on skeletonisation that reduces the robot's free space into a one-dimensional representation. It then produces a Voronoi graph of the free space. According to this graph, the robot plans the path, gets close as possible to the goal, and then uses a straight path to achieve it. These methods, however, depends on the fact that the robot has access to the global map of the environment. [1]

As stated in [5], path planning methods can also be distinguished by the type of environment it operates in (figure 4). The environments in which the robot navigates are primarily dynamic. Unlike in a static environment, obstacles can change their position at any time. Further, global navigation operates in a known environment, which means that the robot possesses a global map. Lastly, relating to this project, local navigation deals with the unknown or partially known environment. This means it must entirely depend on the information from its local sensors. Into this category also falls sensor-based planners mentioned earlier. [5, 11]



Figure 4: Classification of mobile robot path planning methods [5]

Sensor-based planners are free of a global map and depend solely on local sensors to locate themselves and avoid collisions. Nevertheless, they still guarantee completeness (meaning guaranteed to find a solution if there is one). An example of this kind of path planning can be bug algorithms (bug1, bug2, and tangent bug).

Localisation

Localisation is, together with perception and motion planning, a key aspect of a successful navigation system. Its essence is in determining the robot's position in the workspace where the perfect scenario is that the robot's sensors, together with mapping, should promptly identify the robot's present position and orientation, uniquely and repeatedly. However, the reality is quite far from perfect; therefore, a good localisation technique must deal with errors, downgrading factors, improper measurement and estimations. It is mainly dependent on the robot's sensors, and their noise, distortion, or misclassification must be considered to minimise uncertainty and error. Most of the natural environments are unstructured – noisy with many obstacles making the localisation difficult. [2, 5]

Localisation techniques can be split into two categories relative and absolute. Relative methods are completely dependent on the combination of a robot's local sensors. Localisation starts from the initial position of the robot and is continuously updated in time. On the contrary, absolute methods determine the robot position externally. These methods include satellite-based such as Global Positioning System (GPS), external monitoring solutions (e.g. overhead camera system), navigation beacons, and active or passive landmarks. [5, 12]

In some applications, the robot does not possess the global map of the environment (environment is unknown). Therefore, it cannot determine its location relative to a map it does not have. In this situation, the robot must incrementally build the map by mapping the environment and performing localisation simultaneously. This problem is known as already mentioned simultaneous localisation and mapping (SLAM). [1]

Obstacle avoidance

An obstacle avoidance system is used to first detect and then avoid obstacles during robot motion using its local sensors. It is crucial in real-world applications because the environment is dynamic; thus, the obstacles are moving. Evaluating the motion vector of a moving object is still an ongoing research problem. Efficient localisation methods and reliable sensors are necessary to employ a successful avoidance system since the robot must know its current position and direction. This system can be used in both unknown and known environments to protect the robot from collisions. [2, 5]

The difference between path planning and obstacle avoidance was explained earlier. In a known environment, they can be used together so that the path prepared by path planning can be modulated by obstacle avoiding system according to the data

received from the sensor. This is very useful because global maps cannot capture the real-world essence. [2, 5]

Robots rely on local sensor systems in an unknown environment because such an environment has no explicit representation in any form (no map is provided). Some of the approaches dealing with this problem are vector field histogram, dynamic window, or the Potential field discussed later. [5]

1.2.5 TurtleBot2

The autonomous mobile robot used for this project is TurtleBot2, created at Willow Garage by the original makers of ROS (Robotic Operation System). It is the second generation of a low-cost open-source robot supported by the ROS developer community. TurtleBot's software development kit (SDK) is available to download from ROS wiki. This includes a development environment for desktop, demo applications, and libraries for visualisation, perception, control and error handling. It is equipped with a powerful Kobuki robot base, a standard netbook, gyroscope, and Kinect v1 3-D camera (described earlier). This robot's locomotion control is based on differential control. It consists of two independently driven wheels and two non-driven, freely rotating directional wheels. The control principle is based on different rotational speeds of driven wheels, possibly also in the direction of rotation. [13, 14]



Figure 5: TurtleBot2 [14]

SPECIFICATIONS	
Size and weight	Speed and performance
Dimensions: 354 x354 x420 mm	Max speed: 0.65 m/s
Weight: 6.3 kg	Obstacle clearance: 15 mm
Max payload: 5 kg	Drivers and APIs: ROS

1.3 ROS (Robotic Operation System)

ROS is an open-source, meta-operating system that provides libraries and tools to help software developers create robot applications. This includes hardware abstraction, device drivers, libraries, visualisers, message-passing, package management, and many others. It supports code reuse in robotic research and development that allows obtaining, building, writing, and running code across multiple computers. ROS currently runs on Unix-based platforms. [15]

In the runtime, ROS works as a peer-to-peer network of processes. This creates a graph like a distributed network where the building block is called Node. These nodes can be further grouped into Packages and Stacks, which makes them easy to distribute. As these processes (Nodes) run separately, it makes ROS language independent. This means that one Node written in one language can still communicate with any Node written in different languages. It is also robust, meaning when one Node stops working, others continue, and it does not halt the robot from working. [15]

ROS implements different types of communication such as synchronous communication over services, asynchronous streaming of data over topics, and storage of data on a parameter server. [15]

Services are running on a similar basis as client-server architecture. The client sends a request, and the server sends back a response. This is illustrated in figure 6. [15]



Figure 6: Service in ROS

On the other hand, topics are unidirectional streams where the publisher continuously sends the information, and the subscriber can listen to it. The topic is a type of message that must be the same for the publisher and the subscriber. One topic can have many publishers and many subscribers, just like in figure 7. [15]

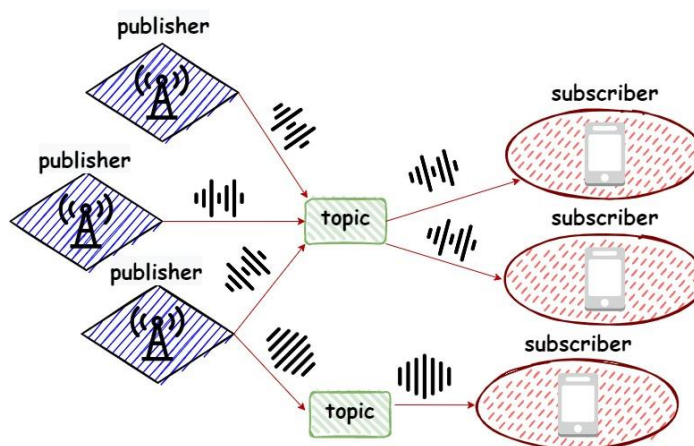


Figure 7: Topics in ROS

Parameter server is then used for automatically assigning parameters such as robot name, sensor read frequency, simulation mode and more. This is used when the program is launched so that the user does not have to set up parameters manually every time the program is initiated. [15]

Gazebo

Gazebo is the robotic simulator that is used to test algorithms, design robots, perform regression testing, and train AI systems using realistic scenarios. It is installed together with the ROS in the form of packages that provide the necessary interfaces to simulate a robot in the Gazebo 3D interface. It integrates with ROS using ROS messages, services and dynamic reconfigure. In this project, it was used as a simulation environment for the TurtleBot2. [16, 17]

RVIZ

It is short for the 3-dimensional ROS visualisation tool. It is used to help visualise what the robot is seeing and doing. This project was used to visualise the perception of the TurtleBot2 in the environment and display the trajectory created by the robot and essential features that help the robot to navigate through the environment. The visualisation is carried out via ROS messages that RVIZ receives from the program. [18]

1.4 Optimisation – Finding minimum of a function

An optimisation problem can be described as maximising or minimising a given function. In this project, optimisation by minimising the function is discussed. The content of this chapter was derived from [19].

1.4.1 Line of Least resistance

Line of least resistance is a line that provided the least resistance while performing motion along this line. In the context of autonomous mobile robotics, this line can be thought of as a path that the robot produces by searching for the least resistance on the way from the initial position to the goal. The least resistance can be interpreted differently in different approaches; for example, in the Potential field (chapters 1.5 and 2.4), it is the lowest potential, and in the Progress field (chapter 2.5), it is the highest progress. In order to find this line, various optimisation techniques can be used. Here are discussed two of them that are relevant to this project.

Triple/Golden search method

This method uses a bracketing technique to find a minimum of a function. Bracketing means that points surround the minimum; this bracketing interval is then getting smaller until the minimum is found. For a minimum to be bracketed we need there a triplet of points, $a < b < c$ (or $c < b < a$), such that $f(b)$ is less than both $f(a)$ and

$f(c)$. Then we know that the function has a minimum in the interval (a, c) . The middle point always represents the best minimum achieved so far.

This function has to be singular, and in regards to this project, it must meet the following:

- f is continuous on $[a, b]$.
- There is a unique value on $[a, b]$ where the derivative $f'(x)$ exists and is zero.
- f is non-constant on $[a, b]$.

Then we have to choose new point x , either between a and b or between b and c . If this point's $f(x)$ results into $f(b) < f(x)$, then it becomes a new neighbour point and a new bracketing triplet of points is (a, b, x) . In the same way, a new triplet (b, x, c) is created when $f(b) > f(x)$. This continues until the distance of two outer points has no sense to split it into intervals. This distance is defined by tolerance, and it is recommended not to set it less than the square root of the used machine's floating-point precision (because smaller values would act just like 0 distance).

The strategy for choosing a new point x between the triplet (a, b, c) is to use golden mean or golden section (called golden section search), which gives us fraction 0.38197 from one and 0.61803 from the other end. The next point is then chosen in the larger of the two intervals (measured from the central point) in the golden section fraction. After starting this method, bracketing triplet segments quickly converge to self-replicating ratios (golden ratios). This method is then guaranteed to bracket the minimum into the interval of size 0.61803 smaller than the preceding interval.

It is important to get the minimum bracketed correctly; therefore, it is necessary to use the initial bracketing method.

This method is used in this project as a baseline for the one-dimensional scenarios.

Downhill Simplex method

The downhill simplex method (also called Nelder–Mead method) considers the multidimensional minimisation problem. It means finding a minimum for the function with more than one independent variable. This method cannot take advantage of one-dimensional minimisation; on the contrary, it is an entirely self-contained strategy. Furthermore, the method does not use derivatives; thus is completely dependent on function evaluations. Due to a high number of function evaluations, it is not very efficient in these terms; however, it might be the best suitable for the problems with low computational complexity (which is the case of this project).

It is represented by a geometrical figure (simplex) with $N + 1$ points in N dimensions. For example, in two-dimensional space, simplex is represented by a triangle and in three-dimensional space by a tetrahedron. The method is only interested in nondegenerate simplexes. Nondegenerate is a bilinear form such that a map from one vector space to another is an isomorphism (it encloses a finite inner N -dimensional volume) that means an inverse mapping can reverse mapping between two structures of

the same type. When one of the simplex points is taken as the origin, then the N -dimensional vector space is spanned by vector directions defined by the N other points.

In multidimensional space, we cannot enclose the minimum into the brackets; hence the downhill simplex algorithm starts with an initial simplex guess that is defined by $N + 1$ points. One of the points is taken as an initial point \mathbf{P}_0 , and the other N points are then calculated by $\mathbf{P}_i = \mathbf{P}_0 + \lambda \mathbf{e}_i$. Or the whole initial simplex can be provided.

- \mathbf{e}_i is a unit vector of each N point
- λ is a constant of an initial guess for the problem's length scale (could be different for each unit vector - λ_i)

The method now takes series of steps in order to find a minimum. It starts with reflection, where it takes the point of the simplex where the function is largest and press it through its opposite face. Another step expands the simplex in some direction in order to take larger steps. After reaching the function's valley, the method contracts itself in a transverse direction and flows towards a flat bottom of the function. If needed, the function contracts itself in all directions in order to get closest to the lowest point. These steps are shown in figure 8. In figure 9, there is shown an exemplary solution of how the simplex method found the minimum of the 2-D problem.

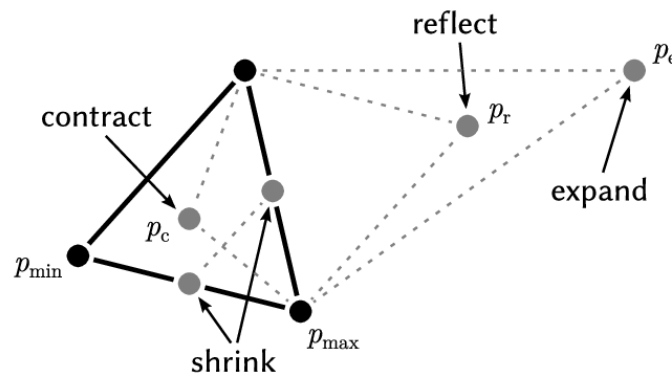


Figure 8: Four kinds of simplex method steps [20]

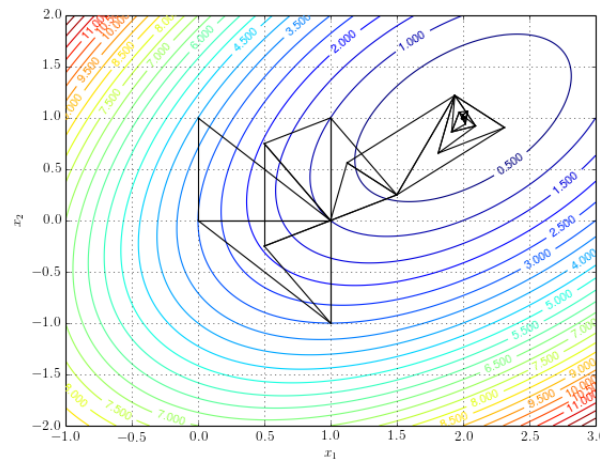


Figure 9: Simplex method finding the minimum in 2-dimensional problem

As a termination criterium, it can be set to terminate when the vector's distance in a step is fractionally smaller than some tolerance. Or decrease in the function's value below some tolerance can be considered.

There is a potential risk that the single anomalous step could get stuck; therefore, it is good to restart the method at a point where the claimed minimum is located. N of the $N + 1$ points should be reinitialised with the P_0 point in the claimed minimum. Since the algorithm already converged to the claimed minimum point, the restart should not be very expensive.

This method goes straightforwardly downhill without making any special assumptions about the function. This method can be extremely robust, although it can be rather slow; however, in the matter of this project, the quick settling of restarts of this method will be leveraged. That is, the next local optimum should always be close to a small extension of the previous one since it is the next nearby point along a continuous line. Hence the search for each succeeding local optimum and point on the line remains feasible in spaces of potentially large dimensionality and thereby find the global optimum of the goal state. This will enable the method to remain effective in this context. Also, the storage requirement is low (N^2) since derivative calculations are not required.

In this project, the method is implemented as the main optimisation method and has the potential to be used to solve N-D optimisation problems.

1.5 Potential field approach

The Potential field is an approach used for the navigation of autonomous mobile robots. Its benefit is that it can be used directly for generating the motion and omitting the path planning part. This is very useful in dynamic unknown environments. The approach is based on repulsive and attractive forces, just like in the magnetic field. The robot can be viewed as a particle that is pulled towards its goal with an attractive potential and repelled from the obstacles with a repulsive potential. By applying this method, the robot has only one global minimum. The Potential field method is regarded as robustly effective in enabling relatively limited local sensor information and computation to efficiently direct behaviour to a local end state by lowering potential or cost. [1, 2]

1.5.1 Local Minima Problem

Although the Potential field approach solved the problem with a need for a global map, they are commonly criticised for not guaranteeing the local end state to always be the goal state because of the Local Minimum Problem, which undermines general completeness claims based on potential or cost. That means that the robot finds the minimum, but it is not a minimum of the goal. This problem arises within a common framework for the approach, which sets a fixed potential or cost for each possible state without taking into account the possibility of multiple different end attractors being created. This causes trapping, no passage between closely spaced obstacles, and limit cycles. [1]

1.6 FM2 approach

The FM2 research paper [21] tackled the problem with local minima, which is also used as inspiration in this project. This research presents a potential-field-based algorithm for 2-D travel that avoids the Local Minimum Problem through only generating a single end attractor and using a global map. This means it avoids the problem caused by the Potential fields.

2 DESIGN AND IMPLEMENTATION

The program, specially developed for this project, controls the robot's movement from turning the wheels to the final novel method that allows the mobile robot to move in an unstructured world. This part describes the whole functioning of this program and the reasoning behind it.

2.1 Setup

ROS Melodic distribution was used on Ubuntu 18.04.5 LTS that was run from the virtual machine (VMware Workstation Player). Python 2.7 was used because that is the version supported by ROS Melodic.

Ubuntu can be downloaded from [21] and ROS Melodic distribution from [22] Desktop-Full installation version to achieve the same setup. The full installation automatically installs the Gazebo simulator and RViz that were used in this project.

After this setup, it is needed to install the TurtleBot2 packages. Officially the TurtleBot2 was lastly supported on the Kinetic version (documentation can be accessed here [23]). In order to make TurtleBot2 work on the Melodic distribution, the GitHub project from the community that solves this problem was used. The project is available here [24], and you can also follow the instructions provided here [25].

2.2 Motion planning

2.2.1 Differential drive

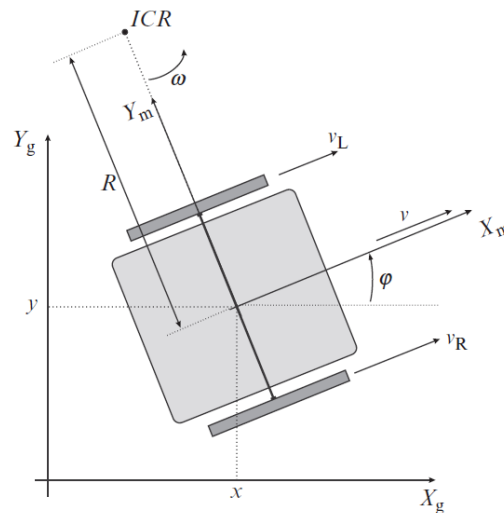


Figure 10: Differential drive kinematics [26]

TurtleBot2 is using the differential drive, one of the simplest drive mechanisms vastly used for smaller mobile robots. It consists of two main wheels, both placed on a

common axis and one or more castor wheels to support the vehicle and prevent tilting. Separate motors control the velocity of each of the main wheels, meaning the velocities of the wheels are independent. The direction of the robot's movement is determined by the point that lies on the common axis of the two drive wheels, and the robot then rotates about this point. By different speeds of each drive wheel, this point can be varied and results in different trajectories. This is depicted in figure 10. [26, 27]
The drive wheels are rotating around the ICR at the same angular speed:

$$\omega = \frac{v_L(t)}{R(t) - \frac{L}{2}}$$

$$\omega = \frac{v_R(t)}{R(t) + \frac{L}{2}}$$

Figure 11: Angular speed around ICR [26, 27]

From where can be further derived, angular velocity, R distance from the ICR and tangential vehicle velocity (all of these are functions of time):

$$\omega(t) = \frac{v_R(t) - v_L(t)}{L}$$

$$R(t) = \frac{L}{2} \frac{v_R(t) + v_L(t)}{v_R(t) - v_L(t)}$$

$$v(t) = \omega(t)R(t) = \frac{v_R(t) + v_L(t)}{2}$$

Figure 12: Calculation of tangential vehicle velocity [26, 27]

TurtleBots's base does not have separate speeds for each wheel, but instead, it has a subscriber to the topic `cmd_vel_mux/input/navi` where it accepts the messages of a `geometry_msgs/Twist.msg` type. Therefore the velocity publisher had to be created to control the robot's wheels. The message type bears the information in the form of linear velocity and angular velocity of the robot. Linear velocity means the forward speed of the robot, and angular is the turning speed. When the angular velocity is zero robot goes straight; when the linear velocity is zero and the angular not, the robot is turning on a spot. Signs of the velocity numbers decide directions.

2.2.2 Localisation

The robot's odometry is used to determine the robot's current location and other important points in space, such as obstacles and goal position. Odometry uses data from applied motion increments commands to determine the change in position over time.

These increments are usually obtained from axis sensors that are attached to the robot's wheels. [26]

The TurtleBot uses odometry to estimate its position and orientation relative to a starting location given in terms of an x and y position and an orientation around the z -axis. TurtleBot's base publishes odometry as `/odom` topic. The program has a subscriber to this topic and accepts the robot's current position the entire time it is running. The orientation of the robot is received in the form of quaternion, which is a more convenient and robust way of representing rotation than in RPY (Roll-Pitch-Yaw) radians. The quaternion is converted to RPY using `tf` library [28].

`tf` is a package that allows a user to keep track of multiple coordinate frames over time and create transformations between them. An example of a frame can be odometry that has its origin in the position where the robot was turned on. Another example might be TurtleBot's base frame that has its origin located in the centre of gravity of the robot with an x -axis facing forward from the robot, y -axis to the left, and z -axis to the ground. The objective of a frame is to have a reference to express the position and orientation of an object. Without a frame, there is nothing the location of an object in space can be related to.

2.2.3 Motion control

The most basic motion of the robot is arc movement. First, the robot had to learn to plan a path to the subgoal with a certain radius (motion planning). Later during navigation, the robot sets these subgoals on the way to the goal and creates a final trajectory consisting of these arc movements.

In the program developed for this project, the user chooses the linear velocity, and angular velocity is then calculated accordingly to reach the subgoal by a path of a certain radius, as shown in figure 13.

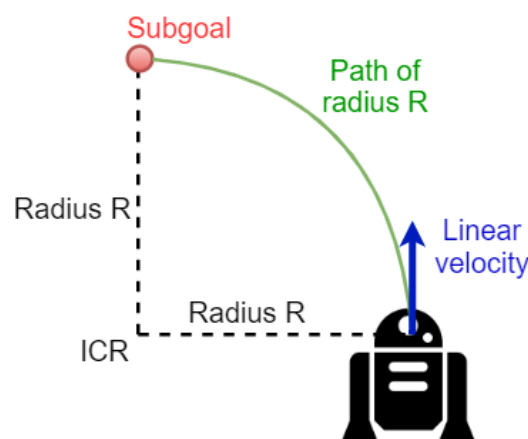


Figure 13: Motion planned to the subgoal

The program is only given chosen linear velocity and the subgoal coordinates. It then calculates the angle α that is between the linear velocity vector and directional

vector to the subgoal. The angle alpha can be found in other places according to the triangle similarity theorems, as shown in figure 14.

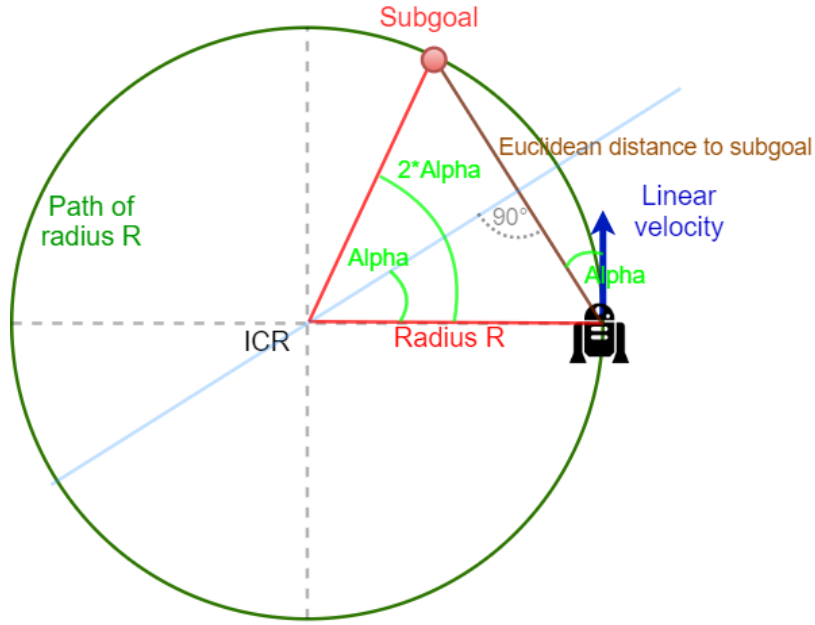


Figure 14: The logic behind the calculation of angular velocity

The angular velocity needed to achieve radius R is calculated from the formula:

$$(1) \omega = \frac{\text{linear velocity}}{\text{Radius } R}, \text{ where}$$

$$(2) \text{Radius } R = \frac{\text{Euclidian distance to subgoal}}{2 \sin(\alpha)}$$

The robot then starts to proceed towards the subgoal. It is important to note that in order to create the smoothest path possible, the robot does not finish the whole path to the subgoal. Instead, it was set that the robot calculates a new subgoal when it gets one-hundredth of *Euclidian distance to subgoal* closer to the subgoal. Meaning the robot calculates a new subgoal fairly often.

2.3 Navigation

2.3.1 Control contour

The basic motion introduced in the previous chapter is the building component of navigation. The aim of navigation is to plan subgoals on the way to the goal to create a safe trajectory for the robot. The subgoals are selected through the control contour located at a fixed distance (selected by the user) from the robot with a selected angle span. The best point selected by the main algorithm (algorithm based on the Potential or the Progress field in case of this project) is a sought subgoal, as shown in figure 15. The

best point is selected according to the goal location and the data perceived from the robot's sensor.

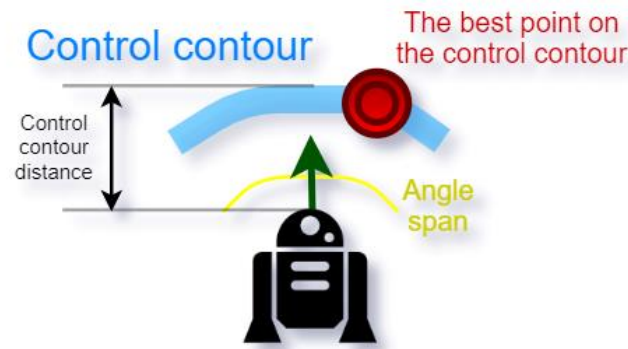


Figure 15: Control contour

2.3.2 Perception

TurtleBot2 uses the Kinect v1 sensor to detect obstacles that appear in front of the robot. Earlier it was explained how the Kinect works and how it gains depth information from the environment. This project utilises the package called *depthimage_to_laserscan* that converts information from the depth camera to the laser scan measurements. These are 640 measurements in the form of a list that TurtleBot publishes on a topic called */scan*. The subscriber to the topic in the main program can access these measurements and also information about their angle span, which in this case is from -30° to 30° . The sensor can detect obstacles from the range of 0.449 meters to 10 meters; every measurement out of this range has a nan value.

The *sensor_filter_function* was implemented to filter out nan values and lower the number of distance measurements to save some computational power. The new values are then stored in two lists. One contains the distance to the detected object, and the other contains the angle under which was given measurement taken from the robot's heading.

These data are with respect to the sensor frame that is practically the same as a robot's base frame. The function *calculation_to_odom* that transfers these data to the odometry frame was implemented. It returns coordinates of the detected obstacle in odometry.

Memory

As mentioned, the used 3-D camera provides only a 60° view. This is problematic, especially when the robot goes around the corner because it stops seeing the obstacle and crashes into it. Therefore, the robot's memory was implemented to tackle this problem.

The memory consists of two queues. One is current obstacle memory that contains coordinates of the currently closest obstacle, and it is added to the list of currently sensed obstacles for evaluation. The second queue is flexible obstacle memory

that keeps searching for a new closest obstacle when the current obstacle memory is occupied. Suppose the obstacle saved in the flexible memory is closer than the one in the current memory. In that case, the one in the current memory is substituted by the new closest obstacle, and the flexible memory keeps searching.

The memory allows the robot to go around corners without hitting the obstacle; however, for a price that the trajectory loses a bit of its smoothness and can be bumpy in places (further discussed later).

Emergency manoeuvre

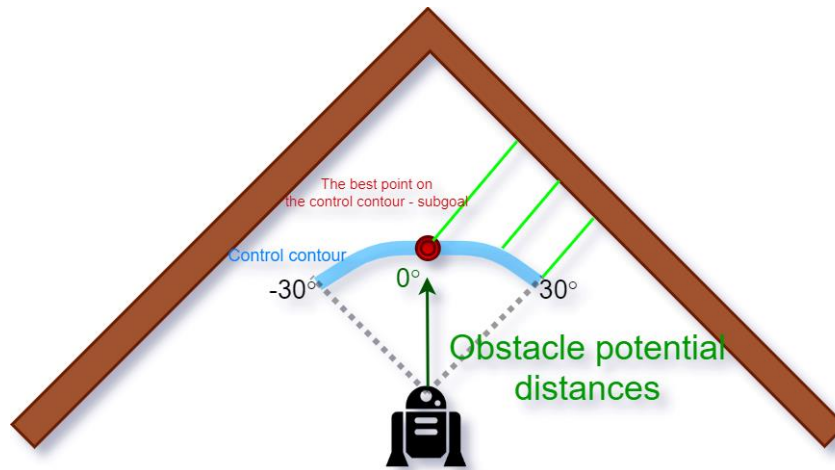


Figure 16: The robot inside the v-shape obstacle

When the robot faces a v-shape obstacle, the robot calculates that the best direction is forward. This is caused by the obstacle distance, where a further distance from the obstacle means lower obstacle potential; thus, higher resulting progress or potential, as shown in figure 16.

The emergency manoeuvre was implemented to avoid the crash. The robot gets so deep into the v-shape that the robot cannot escape from it by setting the subgoal on the control contour. At that moment, the emergency maneuver is activated, and the robot rapidly slows down and plans the path to the subgoal that is just 0.25m away in 90 or -90 degrees from the centre of the robot, as shown in figure 17 (the subgoal designated by the red dot). The robot then turns until it faces out of the obstacle. The full escape can be seen in robustness test 1.

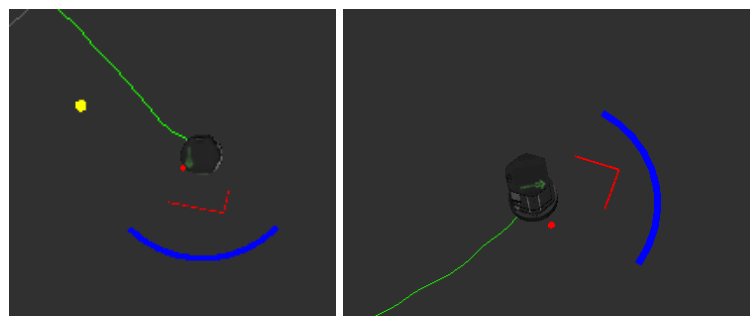


Figure 17: The robot during the manoeuvre

2.3.3 Optimisation of subgoal calculation

The robot driving through the environment continuously evaluates which subgoal on the control contour has better progress or potential. The function that calculates the resulting potential or progress depends on the kind of a used method. This project implements the version of the classical Potential field method and the Reverse anglerfish method specially designed for this project (both discussed in detail later).

Both mentioned methods need calculation of obstacle potential. There are implemented two different functions to calculate it. The first one calculates average obstacle potential from all detected obstacles, and the second one only takes obstacle potential from the nearest obstacle. The second one is slightly more computationally efficient because it has to calculate obstacle potential only from one obstacle; therefore, it is chosen as default.

Further, during this project, there were implemented three methods for calculating the best subgoal. Each of these methods evaluates the resulting progress or potential function mentioned above until it finds the function's minimum on the control contour. These methods are naïve method and then two already described optimisation methods Triple method and Downhill Simplex method. As a result, the methods return an angle to the selected subgoal, as shown in figure 18.

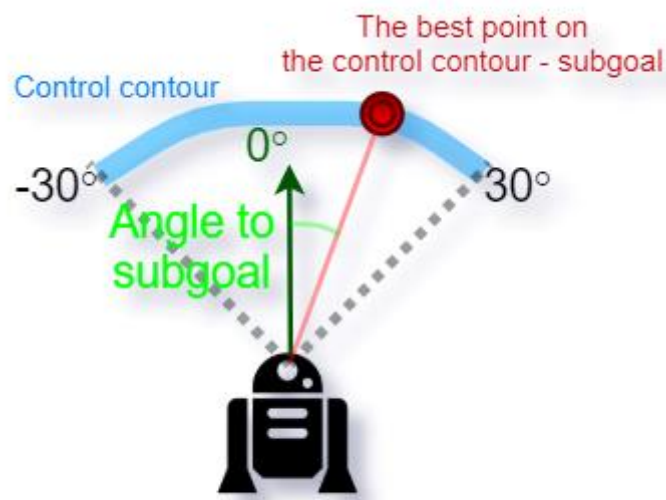


Figure 18: Returned angle to subgoal

At the beginning of the project, the naïve method was implemented where the robot had placed 60 sample points on the control contour spaced by 1° . The resulting potential or progress was calculated for each sample point, and the one with the lowest potential or highest progress was chosen as the next subgoal. Thus, every time the robot was recalculating subgoal coordinates, it carried out 60 calculations.

Optimisation methods rapidly decrease the number of calculations and are also much more precise since they are not restricted by a 1° angle increment.

Note that the robot moving in 2-D is a 1-D optimisation problem because it needs to decide only one independent variable (direction angle) for the subgoal.

The Triple method was used as a baseline and was implemented from scratch. With this method, the number of calculations dropped to an average of 30, which is half more efficient than the naïve method. The drawback of the Triple method is that it can only search in 1-D space. This is enough for the robot that is moving in two-dimensional space and thus needs only one direction angle to subgoal.

The Downhill Simplex method is not restricted to 1-D optimisation space and can be potentially implemented into N-D. The method was implemented by using the SciPy library for the robot moving in 2-D and was proved it can be used for 3-D travel (discussed in detail later). For the robot moving in 2-D, the best results were achieved when the points of the initial simplex were defined on the edges of the control contour in -30 and 30 degrees. The method stops iterating when the change in the resulting direction angle is lower than 0.01 radians. There is also set a hard limit to a maximum of 30 iterations because of occasional outliers. The Downhill Simplex method achieved an average of 14 evaluations, which is half more efficient than the Triple method and four times more efficient than the naïve method.

2.4 Potential field

In addition to the developed novel method, this project implements a version of the classical Potential field method. The principle of the method was explained earlier. The potential functions were defined as follow.

For an attractive goal potential:

$$(1) \text{goal_potential} = \text{distance_to_goal}^2$$

For a repulsive obstacle potential, it was derived from a power-law functional relationship.

$$(2) \text{Obstacle_potential} = \frac{a}{\text{distance_to_obstacle}^n}$$

Coefficients a and n were calculated from the two equations with two unknown variables.

$$(3) y_1 = \frac{a}{x_1^n}, y_2 = \frac{a}{x_2^n}$$

Solving the equations above:

$$(4) n = -\frac{\ln\left(\frac{y_1}{y_2}\right)}{\ln\left(\frac{x_1}{x_2}\right)}, a = y_1 x_1^n$$

The x and y values (x represents the *distance_to_obstacle* and y the obstacle potential) were derived from experiments. It was observed how significant obstacle potential needs to be for the given size of the robot to start turning. It was found that the robot starts reacting when the obstacle potential is around 0.2, and the robot should start reacting 0.9 meters from the obstacle to make a safe turn (also found from experiments). Therefore, $x_1 = 0.9$ and $y_1 = 0.2$; from this point, it is desirable for the obstacle potential to grow rapidly, so for a coefficient calculation, it was used $x_2 = 0.75$ and $y_2 = 500$. From the formula (5) it results in $a = 0.0669612838$ rounded to 0.067 and $n = 31.13$ rounded to 31, which yields the final formula of the obstacle potential:

$$(5) \text{Obstacle_potential} = \frac{0.067}{\text{distance_to_obstacle}^{31}}$$

A graph of this function is shown in the figure below. It grows rapidly with decreasing distance to the obstacle. Negative values can be ignored because the robot never achieves a negative distance from the obstacle. Although robust across a wide range of obstacle courses, it is important to notice that this function was specially designed for the TurtleBot's dimensions and would have to be recalculated for the robot with a different size.

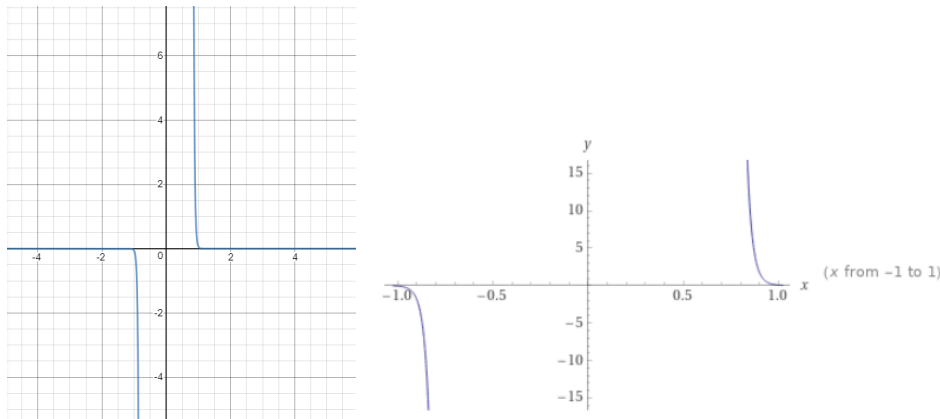


Figure 19: Two graphs of obstacle potential function

$$(6) \text{obstacle_potential} = \frac{0.067}{\text{distance_to_obstacle}^{31}}$$

The distances are calculated Euclidean distances to the goal or to the obstacle. The goal potential gets lower when the robot is getting closer to the goal, and the obstacle potential gets higher when the robot is getting closer to the obstacle. The resulting potential is calculated by adding these potentials together.

$$(7) \text{resulting_potential} = \text{obstacle_potential} + \text{goal_potential}$$

Therefore, the robot always tries to pick the subgoal on the control contour with the lowest resulting potential.

2.5 Progress field

The Progress field is the underlying idea for the method (Reverse anglerfish with deviating lantern point method) developed in this project. It is based on the classical Potential field, and it also took inspiration about generating contours from the FM² research paper [29] to solve the problem with the local minima problem, which occurs in the Potential field method.

The FM² approach uses a contour-generating model in a known environment (i.e. a global obstacle map is known). The contours are generated from a single point, more specifically from the goal. The lava flow analogy can be used to get the notion of the way how these contours are generated—the parts of the contours that are closer to the obstacles flow slower than the parts further away. Lava-like contours flow from the goal to all directions, where each succeeding contour contains all its predecessors. The algorithm then finds the line of least resistance from the goal to the start position, as shown in figure 20. The line of least resistance is a path through the points on the contours where the flow of each contour is fastest (the least resistance). The robot follows this path and thus avoids crashing into obstacles in the environment. The robot's trajectory is found before the motion starts. [29]

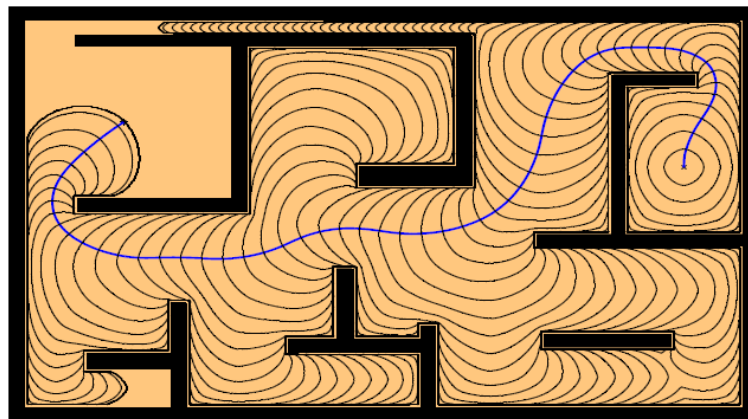


Figure 20: Lava-like contours spreading from the goal position (right side) with a blue line of least resistance from the start (on the left) to the goal [29]

On the contrary, the method developed for this project does not possess the global map of the environment (unknown environment); therefore is entirely dependant on the local sensor. It cannot generate the path to the goal before the motion, so it uses a reverse approach for generating contours. The contours are generated from a generating start point for the motion and are closing around the goal attractor point in the form of a percentage Progress field (with 0% in the start and 100% in the goal), as shown in figure 21.

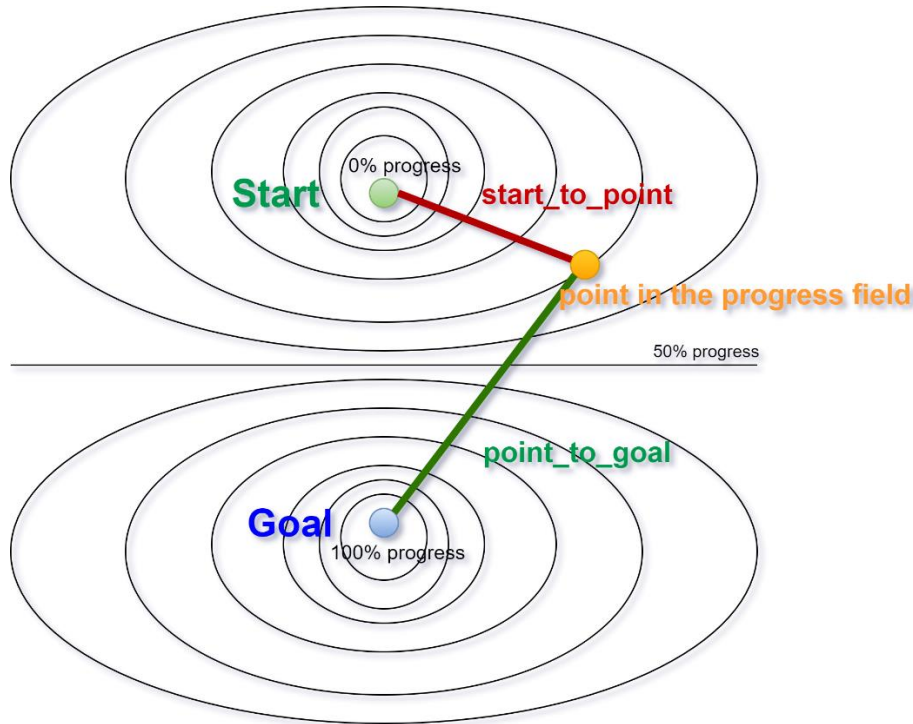


Figure 21: Percentage Progress field with distances used for the fractional progress calculation

Without access to the global map, the robot has to work locally. The progress contours are generated locally and distorted to flow around obstacles by the obstacle potential from currently sensed obstacles. The robot also finds locally the line of highest progress (in other words, the line of least resistance) that creates a safe path for the robot to follow to the goal. More precisely, the robot finds subgoal with the highest progress on the control contour and plans motion to this point by the motion control algorithm described earlier. Even though beforehand the robot knows only the goal's coordinates and everything is done locally, the project's final algorithm is able to solve the local minimum problem using the line of highest progress.

The project's algorithm generates the fractional progress contour field from the single end generating start point to the single end attractor goal point. At the generating point, the fractional progress is 0 and 1 inside the attractor point. Between these points (at 0.5 progress), the generated contour reaches the infinite radius; therefore appears as a straight line (figure 21).

The robot motion algorithm uses these contours to guide itself to the goal. The contours are based on the formula below that calculates the progress fraction for any point in the field. It uses Euclidean distances as follow:

$$(1)Progress = \frac{\text{start_to_point}}{\text{start_to_point} + \text{point_to_goal}}$$

In free space (environment without any obstacles), the robot moves through the contours based on choosing the point of maximum progress at each stage. This allows the robot to achieve the goal from any initial heading, as shown in figure 22.

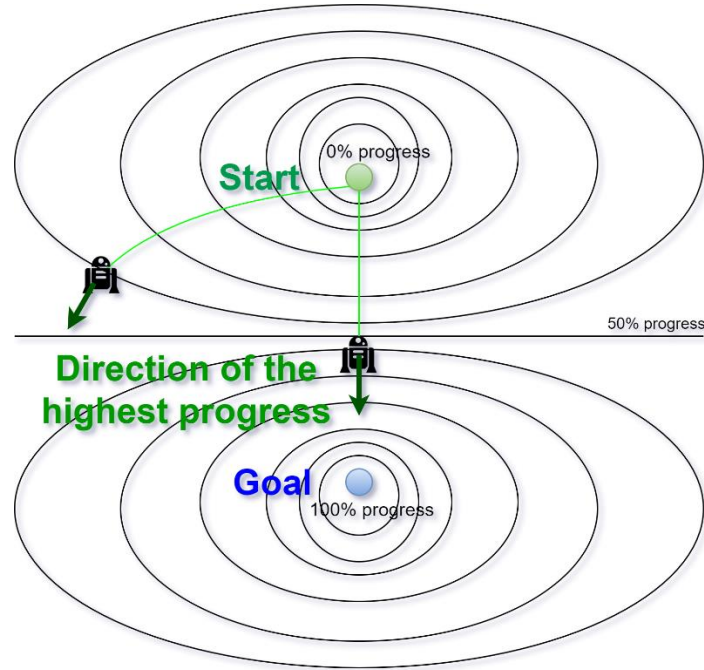


Figure 22: The robot moving through the contours

The contours are also sensitive to obstacles, and the progress is locally distorted according to the obstacle potential, meaning the progress closer to the obstacles is lower. The aim here is to prevent collision with obstacles through increasingly high obstacle potential as the obstacle is approached. When the robot detects the obstacle, the obstacle potential is calculated and is incorporated in progress calculation, as shown in formula 2. That lowers the progress closer to obstacles and causes turning the robot away from obstacles to choose directions with higher progress.

$$(2) \text{Progress} = \frac{\text{start_to_point}}{\text{start_to_point} + \text{point_to_goal} + \text{obstacle_potential}}$$

This potential depends on the robot's distance from the obstacle, and the function for calculating the obstacle potential was determined from a power-law functional relationship, as explained earlier.

$$(3) \text{Obstacle_potential} = \frac{0.067}{\text{distance_to_obstacle}^{31}}$$

Furthermore, *goal_ratio* is introduced to make the robot prefer shorter paths to the goal location. Figure 23 shows the Euclidean distances used for *goal_ratio*

computation, where *current* means a current position of the robot and *sample point* is the point for which the robot calculates the progress at the moment.

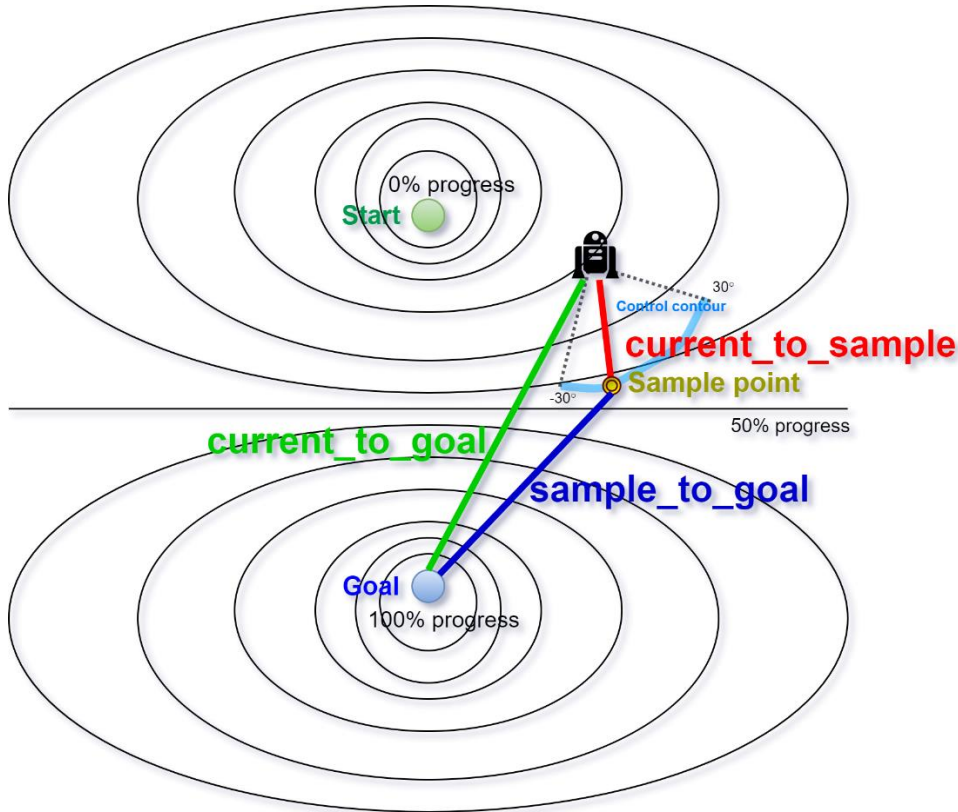


Figure 23: *goal_ratio* Euclidean distances

The *goal_ratio* is then calculated according to these Euclidean distances, as can be seen in formula 4.

$$(4) \text{goal_ratio} = \frac{\text{current_to_sample} + \text{sample_to_goal}}{\text{current_to_goal}}$$

This ratio always has a value of one or above because *current_to_goal* represents the shortest possible path to the goal from the current position in the denominator. In the numerator, there is a rough estimate of the distance the robot plans to take to the goal (*current_to_sample* + *sample_to_goal*) if the sample point is chosen as a subgoal. The *goal_ratio* is incorporated in the final progress calculation, as shown in formula 5.

$$(5) \text{Final Progress} = \frac{\text{Progress}}{\text{goal_ratio}}$$

As a result, it punishes sample points that are further away from the goal by lowering the progress percentage. If the *goal_ratio* had not been used, the robot might have attempted travelling to the goal across the fractional progress contours using excessively long paths to the goal. Figure 24 shows the robot moving across contours when the

initial heading is 90 degrees from the goal direction. This is causing unnecessarily long paths. This problem is solved by incorporating *goal_ratio* into the final progress formula.

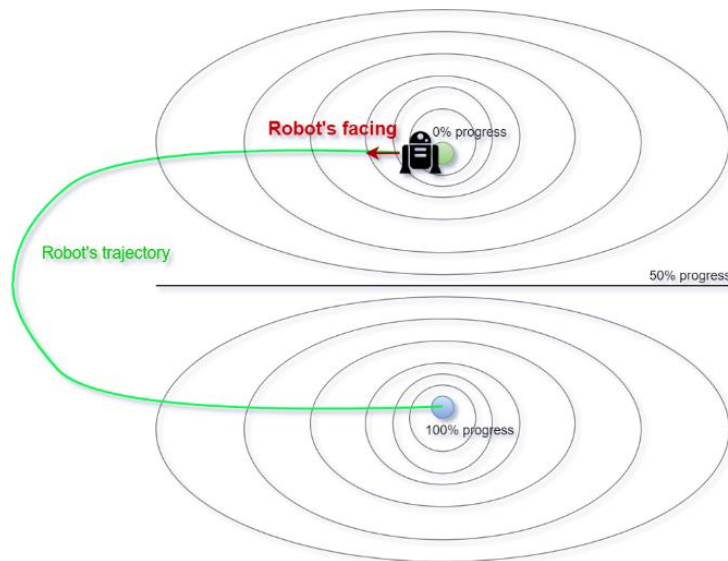


Figure 24: The robot moving across the contours without *goal_ratio*

The contours are now locally distorted due to the obstacle potential and *goal_ratio*. The robot then finds the highest progress on the control contour and sets it as the next subgoal to move to.

2.5.1 Key finding

During the numerous experiments with the Progress field, it was noticed that the robot's behaviour is relative to its location in the Progress field. Because of the nature of the field, it can be noticed that it is shaped in a convex manner in the upper part and a concave manner in the bottom part, as shown in figure 25.

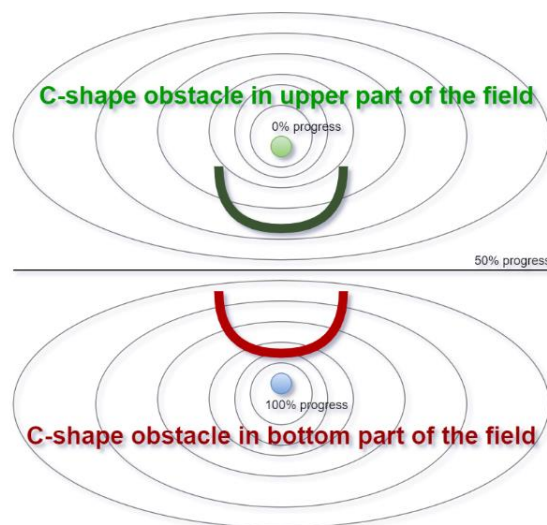


Figure 25: Progress shaped as convex and concave

During the experiments, it was found that the robot is able to escape from a medium c-shape obstacle (as drawn in figure 25) in the convex-shaped Progress field, but not from the concave-shaped as demonstrated by the green paths in figure 26, where the figure of eight on the left shows looping preventing goal completion and the half-heart shape on the right shows goal completion.

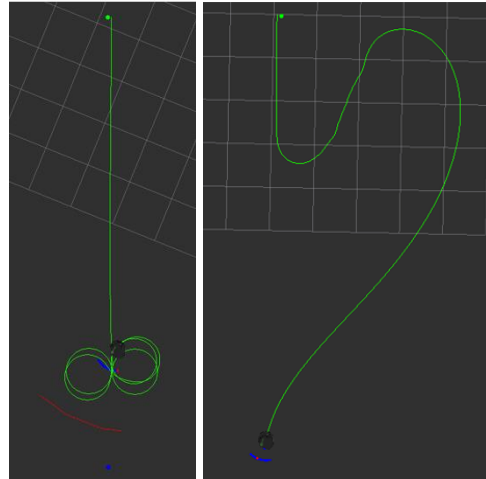


Figure 26: On the left, the c-shape obstacle is placed in the bottom part and on the right in the upper part of the field

2.5.2 Reverse anglerfish with deviating lantern point method

Reverse anglerfish with a deviating lantern point is a biologically inspired solution method for the mobile robot developed for this project in order to solve problems such as in figure 26. It uses the Progress field and is based on the various key findings. It allows the robot to escape from challenging obstacles like the c-shape obstacle of different severities.



Figure 27: Anglerfish

The method was inspired by an anglerfish (figure 27) that uses its lantern to see the environment. On the contrary to the anglerfish, the progress source point that is the analogue of the lantern point is placed *behind* the robot at a fixed distance in order to have convex-shaped progress contours in front of the robot. Moreover, the robot is able to deviate its lantern point between 90 and -90 degrees from the robot's heading.

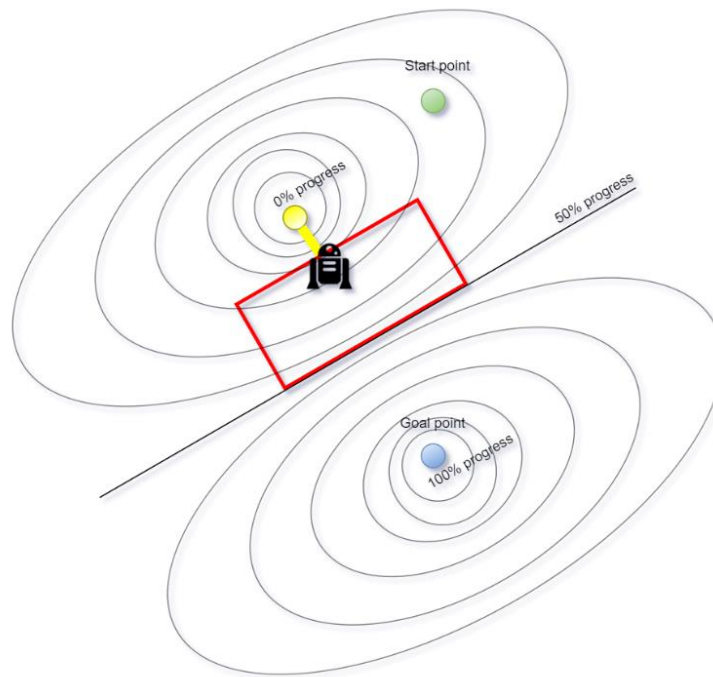


Figure 28: Lantern point (progress source point) in a fixed position behind the robot

This led to the idea to attach the lantern point (progress source point) at a fixed small distance initially behind the robot to keep the desired convex shape of the Progress field always in front of the robot, as seen in the red rectangle in figure 28. This means that when the robot is moving, the lantern point is also moving. When the robot is escaping from the c-shape obstacle, it is then led out of there by the progress contours that tell the robot to keep moving away from where it has been in relation to the obstacle.

Different distances of the progress source point behind the robot were tested. If this point is too close to the sample point, it causes oscillations of the robot because the progress is too little to make significant differences between close samples on the robot's control contour described in section 2.3.1. On the other hand, when the progress source point is too far from the robot, it can cause the robot to appear in an undesirable concave-shaped field while navigating an obstacle too early before it has a chance to reach the goal (attractor point). A range of robustness was found where the performance of the robot was not negatively impacted, which was from 0.6 - 1.5m. Placing the point one meter behind the robot has proved a good compromise between these two limits.

However, this setup still cannot escape from more severe c-shape obstacles. If the c-shape is too severe, it causes the robot's rotation also to be severe, which directs progress to point back again inside the obstacle to form a loop, as in figure 29.

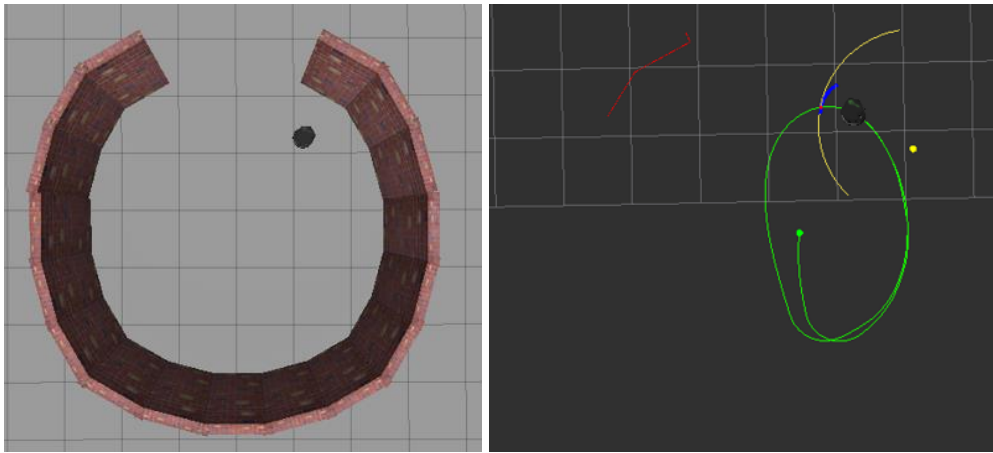


Figure 29: Severe c-shape obstacle - The robot falling back to the obstacle

Another important finding was made about the usefulness of deviating the lantern point from the position exactly behind the robot to the robot's side, as shown in the figure below.

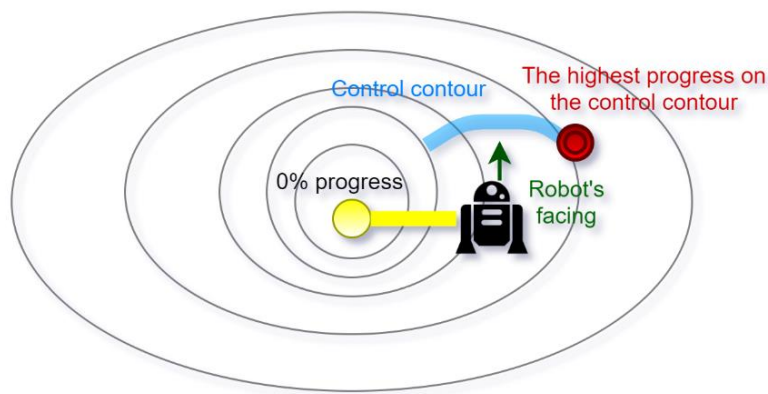


Figure 30: 90 degrees deviation of the progress source point

When the lantern point is directly behind the robot, it encourages the robot to go straight ahead. By placing the lantern point to the side of the robot when desirable, it gives the robot a direction of push more towards the opposite side; so that when it is, for example, on the left side, the progress contours are pushing the robot more towards the right (shown in figure 30) and vice-versa.

When there are no obstacles and the lantern point is deviated, it makes the robot spin in a circle around one point, as shown in figure 31, where the green line is the robot's trajectory. Note that when the lantern point is deviated to the left side, the robot circles around clockwise and vice-versa.

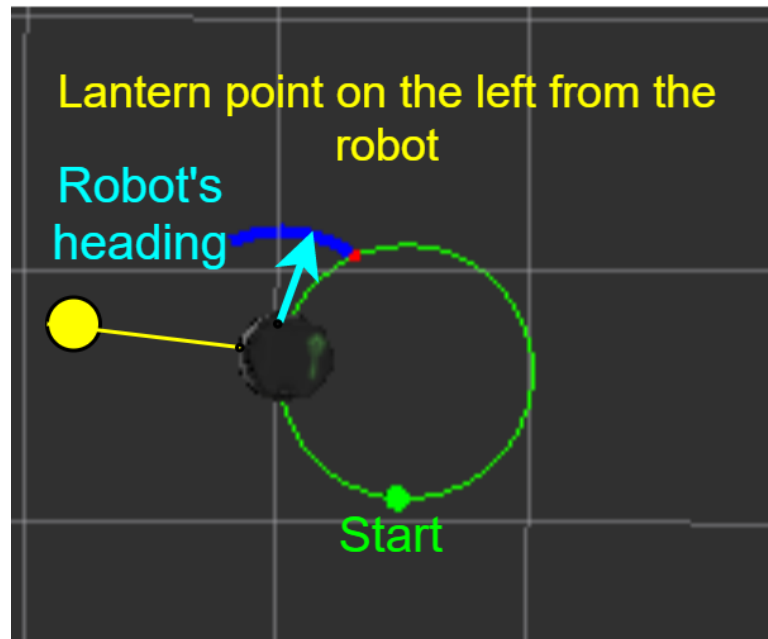


Figure 31: The lantern point deviated in a fixed position to the left from the robot, causing driving in a circle

However, when the robot is next to the obstacle, and the lantern point is deviated to the side opposite the obstacle boundary, this helps the robot start circling the obstacle, as can be seen in figure 32. This is because the robot is pushed to the side (to the obstacle) by the progress generated from the lantern point but at the same time pushed away from the obstacle by the obstacle potential.

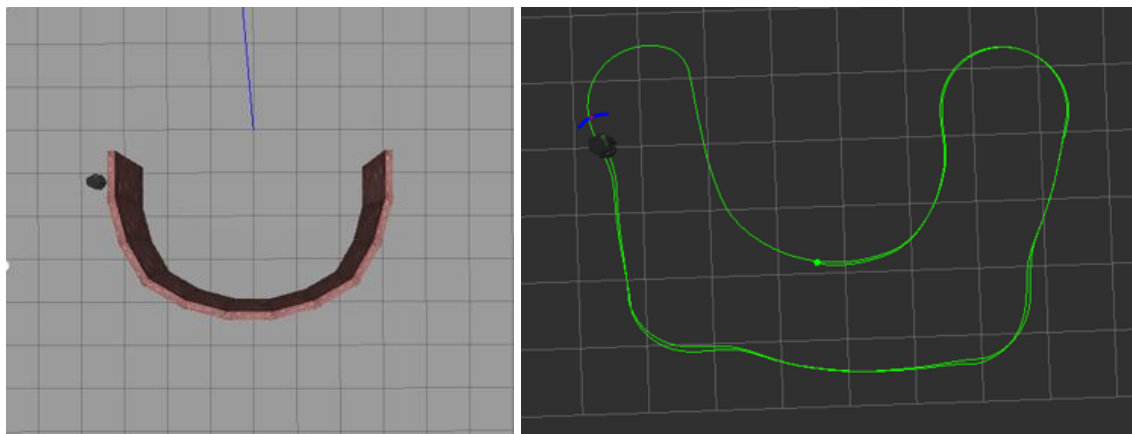


Figure 32: The robot circling the obstacle clockwise with deviated lantern point at 90 degrees on the left side of the robot

All the positions of the lantern (progress source) point in the Reverse anglerfish method are presented below.

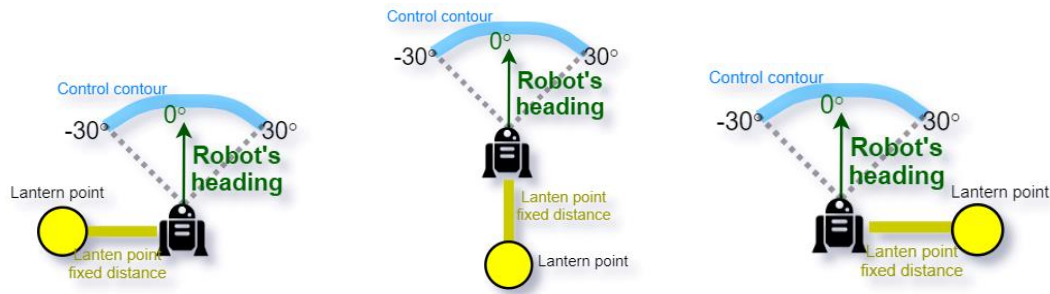


Figure 33: All possible positions of the lantern point

The robot then operates in 2 different modes:

- 1) Free mode – the lantern point is directly behind the robot, exactly deviated 180 degrees from the heading direction of the robot.
- 2) Deviation mode – the lantern point is deviated in 90 or -90 degrees from the facing direction of the robot. This places the lantern point to the left or right side of the robot. When the lantern point is on the left, it forces the robot to turn more to the right and vice-versa.

The robot utilises the directional push created by the lantern point to escape from even the most severe obstacles, such as the c-shape double snail-like obstacle shown in figure 35. During the escape, the robot is in the deviation mode. The lantern point is on the side, pushing the robot to the obstacle, and the obstacle potential is then pushing the robot away from the obstacle. This causes the robot to continue forwards along with the obstacle, as was shown in figure 32.

Switching between free mode and deviation mode is controlled by the control angles. These angles introduce a new angular coordinate system (control angles coordinate system) in which the robot's current heading moves (figure 34).

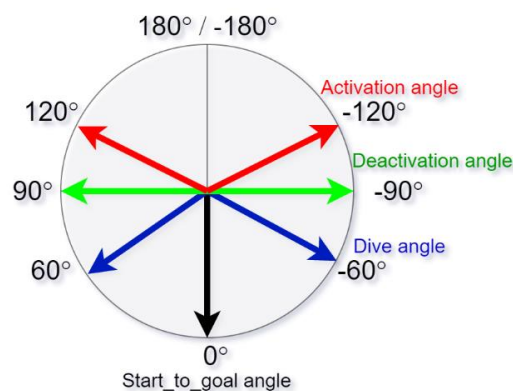


Figure 34: Control angles coordinate system

Before the robot starts moving, the control angles coordinate system is defined by the first of these angles called *start_to_goal* angle. This angle is defined independently of the robot's heading by direction from start to goal. Therefore the system's *start_to_goal* angle always points to the goal from the start and has a value of 0 degrees in the control angles coordinate system, as shown in figure 35.



Figure 35: Initial setting of the control angles coordinate system

The algorithm then also remembers the value of *start_to_goal* angle in the robot's coordinate system (odometry), which is most probably different than 0° . Then according to the value of *start_to_goal* angle in odometry, it converts the robot's current heading during the motion to the control angles coordinate system where the value of *start_to_goal* angle is always 0° . An example of such conversion is shown in figure 36.

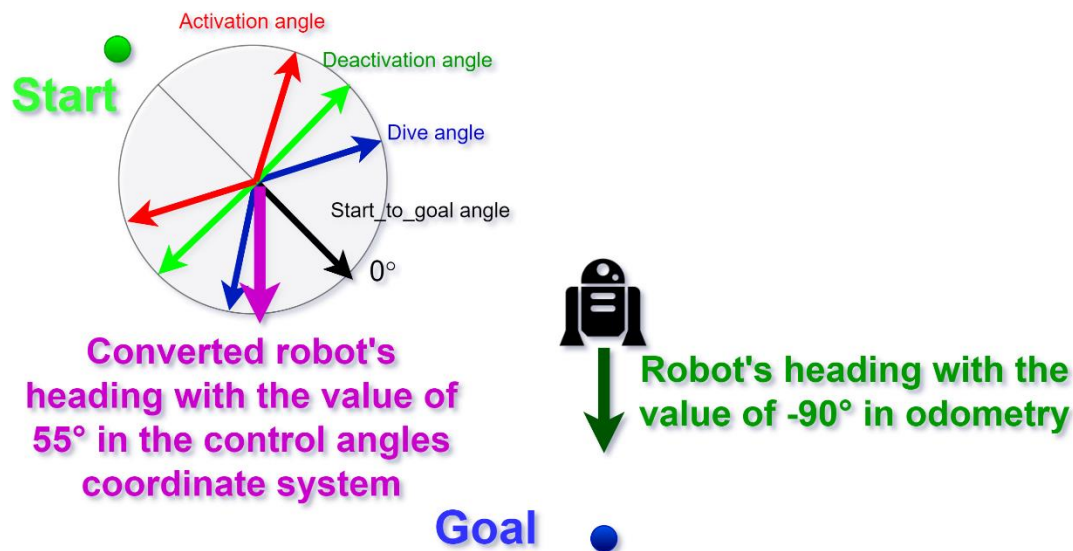


Figure 36: Example conversion of the robot's current heading from odometry to the control angles coordinate system

The other control angles are defined by vectors pointing under certain angles specially designed to turn on and off the deviation mode. There are two *activation* angles (120° and -120°), two *deactivation* angles (90° and -90°) and two *dive* angles (60° and -60°).

The current heading angle of the robot is converted into the control angles coordinate system during the robot's motion as *converted_heading* angle. The deviation mode is then activated when the *converted_heading* angle exceeds the activation angle. When the positive 120° is exceeded, the right deviation is activated and vice versa. When the robot's heading then comes back below 90° (which is the deactivation angle for the right deviation), the deviation is deactivated. The intuition here is that when the robot's heading goes above 120° or -120° , it means that some obstacle is forcing the robot to go away from the goal. As a response to that, the robot activates deviation mode that causes following the border of the obstacle and attempting to escape from the obstacle until it passes the appropriate deactivation angle.

The rationale behind *dive* angle and other additional features will be explained subsequently on the example of the robot completing the exit from the severe c-shape snail-like obstacle in figure 37. The c-shape obstacles can have different severities starting with a very mildly curved c-shape, then it starts to add on the difficulty for the robot to escape with higher and higher curvature of the c-shape. This goes on until the c-shape starts to form spirals forming the already mentioned c-shape snail-like obstacle (figure 37), this is the most severe case, and the robot needs some mechanism that signals it that it is moving in a spiral in order to escape.

It is also important to mention that the values of angles were chosen intuitively according to experiments that were done in simulation. The *activation* angle of 120° was selected to be above 90° , so it does not cause unnecessary turning off and on while

travelling along the wall that is in 90° from start to goal line but is only activated when the robot starts to climb a wall directed away from the goal. The *dive* angle was picked with similar intuition at 60° , and the *deactivation* angle worked effectively when placed between these two angles.

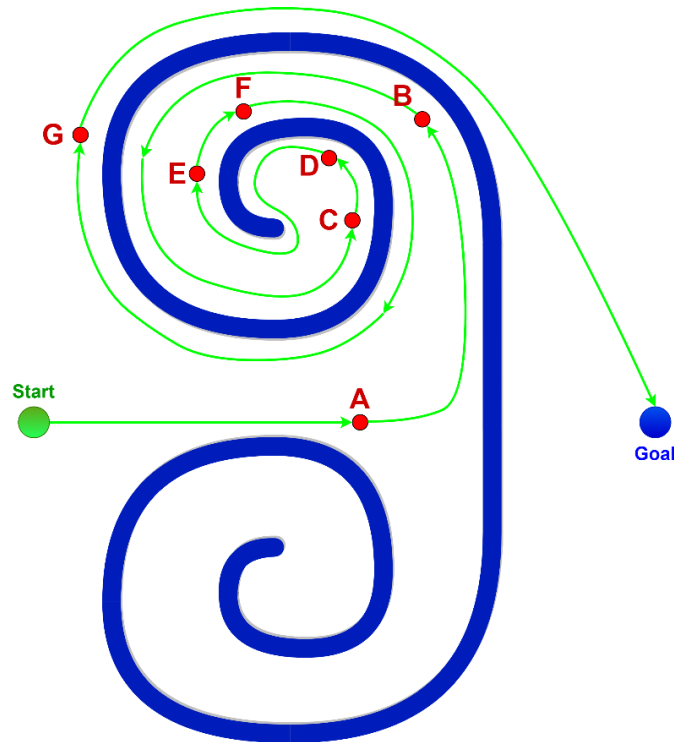


Figure 37: Reverse anglerfish with deviating lantern point method solving a c-shape snail obstacle

The control angles for the situation in figure 37 are as follow:

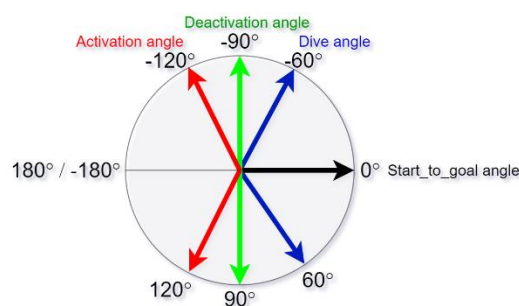


Figure 38: Control angles for the situation in figure 37

The obstacle presented in figure 37 contains mentioned spirals or snail-like curvature of the c-shape obstacle. Therefore, the robot keeps track of the loop count, meaning it remembers how many full loops it has done into the spiral in order to be able to escape from it. This is where the *dive* angle comes into play. When the loop count is above zero, the robot wants to get out of the spiral; therefore, it waits for its heading

angle to dive below the *dive* angle and activates appropriate deviation, thus escaping from the spiral.

- A. The robot starts sensing the obstacle in front of it and starts avoiding it by moving along the bottom of the c-shape obstacle.
- B. The robot achieves -120° and thus activates the left deviation mode.
- C. The robot reached the *deactivation* angle and deactivated the left deviation mode. The robot also knows that the *lastly passed angle* was a -60° dive angle; therefore, it knows it completed the entire loop. For this reason *loop count* is +1.
- D. The robot achieves -120° again and activates the left deviation mode.
- E. The robot reached the *deactivation* angle and deactivated the left deviation mode. However, contrary to the C this time, the *lastly passed angle* is -120° activation angle. For this reason, that is not counted to the *loop count*.
- F. At this moment, the robot's loop count is above 0. That means that the robot is trying to get out of the spiral. When the *loop count* is above 0, and the robot gets below dive angle (in this case -60°), the robot activates the left deviation mode again and tries to escape from the loop.
- G. The *last passed angle* is -120° activation angle, and thus the robot knows it completed the full loop and makes the *loop count* -1. The *loop count* is now equal to 0, and the robot knows it is now safe to leave the obstacle. Now the robot does not need to escape from the spiral anymore, so it does not activate deviation mode when going below *dive* angle. The rest of the path to the goal is carried out by the free mode.

By checking the last passed angle, the robot makes sure that the full loop was made. This serves as the robot's safe to leave condition (meaning the robot successfully escaped the obstacle) and allows the robot to avoid the obstacles inside the snail obstacle without mistakenly changing the *loop count*, as shown in figure 39.

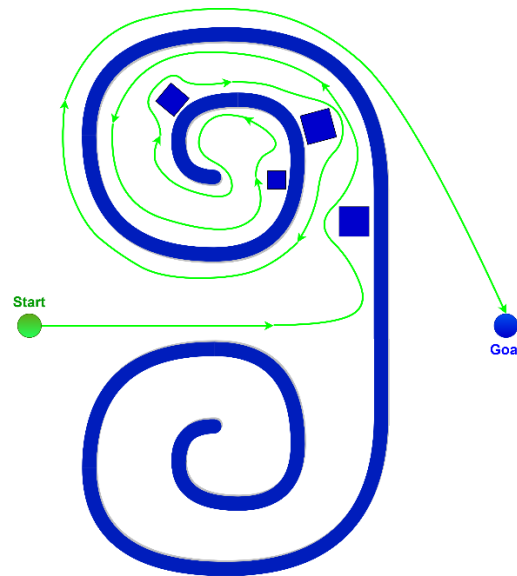


Figure 39: C-shape snail obstacle with inner obstacles

An additional feature in the algorithm is a restart of the control angles coordinate system. This is implemented to prevent undesired trajectories when the robot is forced to approach the goal from behind (figure 40).

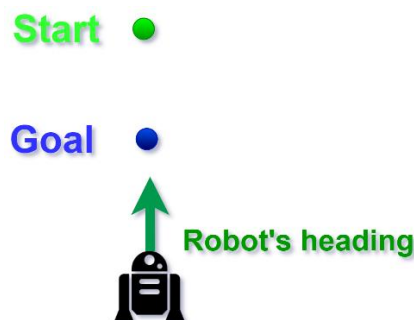


Figure 40: The goal being approached from behind

When the robot had approached the goal from behind, it would have activated the deviation mode that would have dragged the robot away because the deviation mode would repeatedly get activated and deactivated. For this reason, the robot is checking the *current_position_to_goal* angle (independent of the robot's heading). If the current to goal angle exceeds 90 or -90 degrees in the control angles coordinates, it restarts them with the *current_position_to_goal* angle as a *new_start_to_goal* angle, making it a new centre (0°) of the new control angles coordinate system. The restart is shown in figure 41, and its use can be seen in the robustness test 6 and 7.

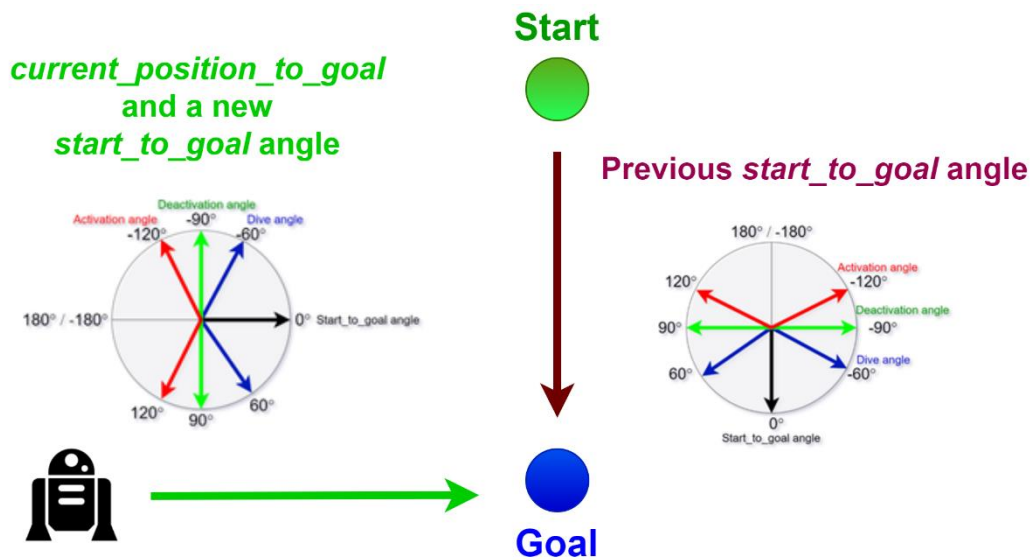


Figure 41: A restart of the control angles coordinate system

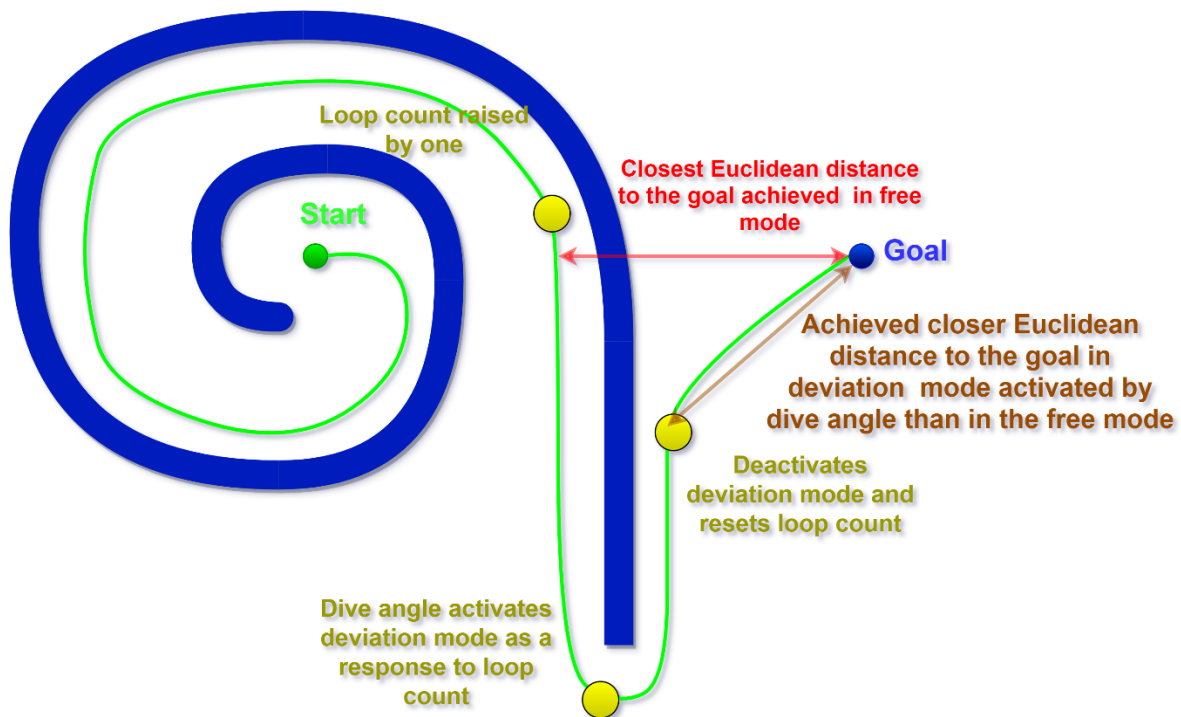


Figure 42: Closest Euclidean distance to the goal feature

Another added feature is remembering the closest Euclidean distance to the goal achieved in the free mode. This is useful when the robot starts from inside of the obstacle and picks up some undesired *loop count* that would cause activation of the deviation by the dive angle, and the robot would drive past the goal and circulate around the obstacle instead of going to the goal. Therefore, the robot always remembers the closest Euclidean distance to the goal achieved in the free mode, and in the deviation mode activated by the dive angle, it checks whether it gets closer than that. If it gets

closer, it indicates that it escaped the obstacle, restarts *loop count*, and deactivates the dive, as shown in figure 42. Results from simulation can be seen in robustness tests 8 and 9.

Reverse anglerfish with deviating lantern point algorithm

The algorithm of this method contains already mentioned control angles. It also stores some true/false values that are *left_deviation_activated*, *right_deviation_activated*, *dived* and *dive_last_passed*. The first two are self-explanatory; *dived* keeps information on whether the robot went under the *dive* angle when the *loop count* is above 0, and *dive_last_passed* lets the robot know from which direction the deactivation angle was reached. Lastly, it also stores the *loop_count* that designates the robot, whether it is moving in a spiral such as a snail obstacle.

- 1) Firstly the algorithm calculates the *start_to_goal* angle that serves as 0° in control angles space.
- 2) It checks whether the restart is needed by calculating the *current_position_to_goal* angle.
- 3) It converts the robot's current heading into the control angles space.
- 4) It checks which mode is activated. If none of the deviation modes is active, then it is in free mode.
- 5) In free mode, it checks whether the robot's current heading exceeds one of the activation angles; if that is true, the algorithm activates the appropriate deviation mode. The free mode also checks whether the robot went under the dive angle when the *loop_count* is higher than 0. If this is true, the algorithm again activates the appropriate deviation mode.
- 6) In deviation mode, it is first checking what was the last passed angle. Then it checks whether the deactivation angle was achieved. Before deactivating the deviation mode, it decides whether to subtract or add one to the *loop_count* or do nothing to the *loop_count*. When the robot did not dive (*dived*=False) and completed the entire loop meaning the last passed angle was a dive angle, it adds one to the *loop_count*. In case the robot did dive (*dived*=True) and the last passed angle is activation angle, it subtracts one from the *loop_count*. If neither of these situations happens, the loop count is not affected.
- 7) Additionally, it checks whether the closest Euclidean distance to the goal achieved in the free mode is longer than the distance to the goal in the currently activated deviation mode by *dive* angle. If that is true, *loop count* is restarted, and deviation mode deactivated.

2.6 Applicability of Reverse anglerfish algorithm for 3-D space travel

A drone flying in space is an example of 3-D travel. The algorithm designed for this project could be used in this scenario to navigate the drone through space without colliding. The difference between the Turtlebot moving in 2-D coordinates is that the

drone has a two-dimensional control surface instead of the one-dimensional control contour, as can be seen in figure 43.

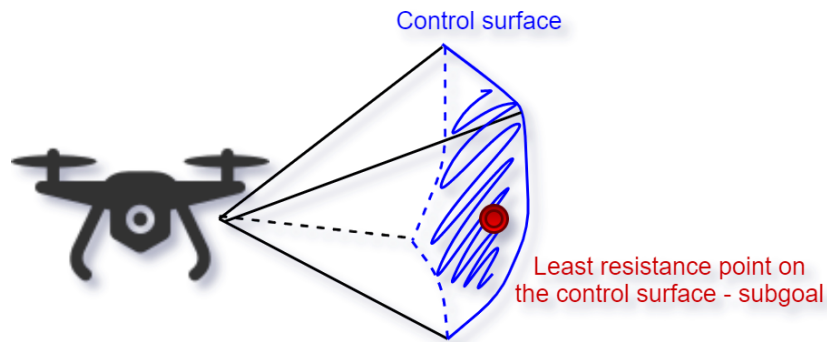


Figure 43: Drone moving in 3D

The position of the least resistance point is defined by the known Euclidean distance of the control surface from the drone and by the two angles. This is equivalent to the polar coordinate system shown in figure 44.

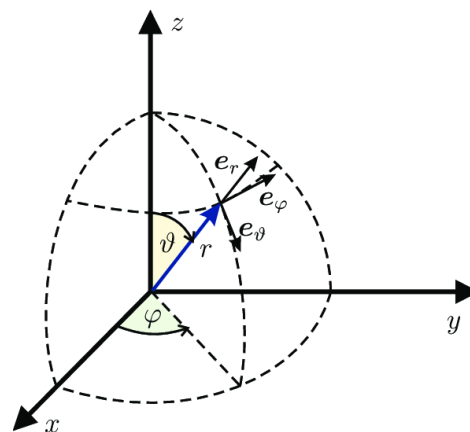


Figure 44: Polar coordinate system – position defined by distance and two angles

The simplex optimisation method implemented in the navigation algorithm then searches for these two angles. Therefore, in this case, it is a 2-D optimisation problem.

The 2-D travel used for TurtleBot was altered to demonstrate the usability of the Reverse anglerfish algorithm for 3-D travel. As mentioned above, there are two angles that need to be found in 3-D space. Therefore, there was defined another angle that specifies the velocity of the robot, in addition to the subgoal angle. This alteration recreates the conditions that are found in 3-D space.

The velocity was implemented into a potential function that now has two independent variables instead of one. Subgoal angle is defined between -30 to 30 degrees and speed angle between 0.1 and 1 m/s. In particular, the velocity was added to the obstacle potential calculation, as shown in formula 1.

(1) *Obstacle potential with velocity* = $obstacle_potential + \frac{obstacle_potential}{1.1 - velocity}$,
 where 1.1 is selected so that the denominator is not 0 when the speed is one m/s

Such implementation causes that when the obstacle potential by itself is high, the simplex method finds minimum speed to be 0.1 m/s making the resulting obstacle potential smaller; therefore, the lower speed is more desirable.

However, when the obstacle potential is very low (almost 0), the simplex method is searching on a flat surface where it is extremely hard to find the minimum and ends up selecting velocity from its whole interval (0.1 to 1 m/s).

This results that the robot is moving with a speed of 0.1 m/s around obstacles and with speed in the interval from 0.1 to 1 m/s in free space. This implementation might not have a practical use but serves well as proof that the method is applicable to higher-dimensional space.

2.7 Using Navigation GUI

The graphical user interface using python Tkinter was created for easier manipulation with parameters, switching between different modes, starting and stopping the robot. When the user turns on the GUI, it loads with default parameters that the user can alter. The goal coordinates are set in the robot's odometry space. The GUI's parameters are self-explanatory, as can be seen in figure 45.

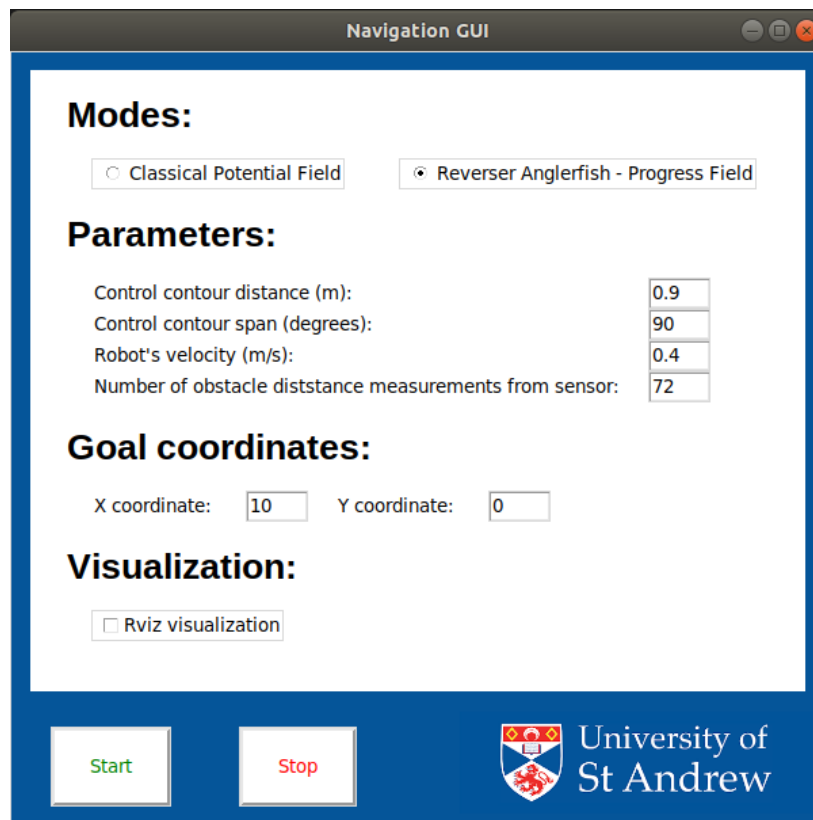


Figure 45: Navigation GUI designed for the project

In order to use the GUI, the user must open one of the created worlds in the Gazebo simulator for the TurtleBot or create their own. The simulation world can be open from the Ubuntu terminal with the command below.

```
@ubuntu:~$ roslaunch turtlebot_gazebo turtlebot_world.launch world_file:=/file_location/world_name.world
```

Figure 46: A command that opens selected world in Gazebo simulation

The Rviz also has to be open when the visualisation button is ticked off to visualise what the robot sees. The following command can start Rviz.

```
ubuntu:~$ roslaunch turtlebot_rviz_launchers view_robot.launch
```

Figure 47: A command that opens Rviz

Then the user navigates to the folder with `navigation_GUI.py` and accesses it by following.

```
/navigation_file$ python navigation_GUI.py
```

Figure 48: A command that opens Navigation GUI

After the user sets desired parameters, he can start navigation to the goal by clicking the start button and stop the robot with the stop button

2.8 Incorporation into a physical robot

The program created for this project was also tested on the physical robot TurtleBot2, shown in figure 5. It successfully avoided simple obstacles. The incorporation is pretty straight forward and running the program is not different from simulation. The manual for setting up the physical robot can be found in appendix 2.

3 EXPERIMENTS AND EVALUATION

Multiple worlds with various obstacle complexity were created in the Gazebo simulator in order to test and compare the performance of the developed method.

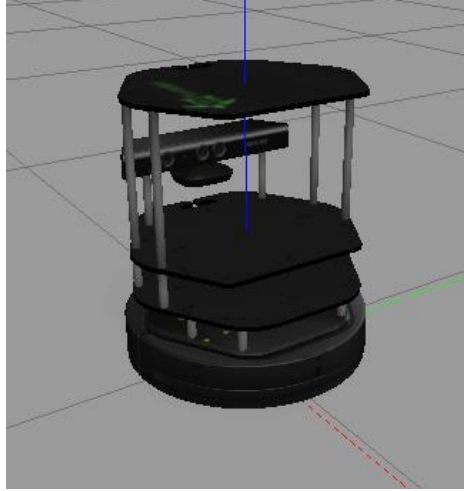


Figure 49: TurtleBot2 in Gazebo

Visualisation publishers were created to display what the robot sees in RVIZ. Red lines represent the obstacles that the robot perceives at the moment. A green point is a starting point, and the blue one is a goal. A green path is a trajectory produced by the robot. A blue curved line in front of the robot is a control contour where the red point represents the best progress or potential. For the Reverse anglerfish method, the lantern point is displayed as a yellow point, and the progress contour showing the best percentage progress is a yellow contour (figure 50). The percentage progress contour displayed in RVIZ can sometimes be slightly off because the calculation of the contour was simplified due to its otherwise complex computation.

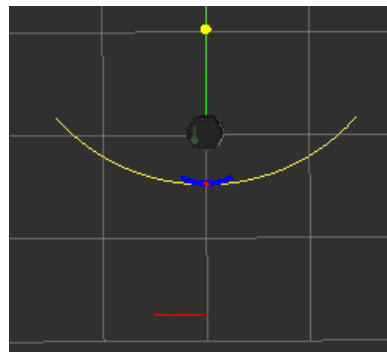


Figure 50: Visualisation in RVIZ

In Appendix 1, there are presented experiments in different world configurations. The motion tasks were steadily made more complicated with new levels of difficulty added one at a time where possible so the causes and effects could be most easily ascertained and dealt with, together with a broad systematic range of testing. The

Reverse anglerfish method developed for this project is compared with the version of the classical Potential field approach. The velocity, number of distance measurements, control contour distance and angle, are constant throughout all experiments for both tested methods. The trials found that the best performance and smoothest path is achieved with the control contour distance of 0.9 m with an angle span of 90 degrees. It takes 72 distance measures, and the velocity was chosen arbitrarily to be 0.4 m/s, where the maximum TurtleBot's speed is 0.65 m/s. These values are set as default in the navigation GUI. Both tested methods were implemented with the obstacle memory.

Experiments and Robustness tests summary

As mentioned, all the experiments, robustness tests and inferences with more details are presented in Appendix 1. Each experiment and robustness test in the appendix presents a new challenge for the robot, which is then analyzed and explained.

At first, a version of the classical Potential field method was compared with the Reverse anglerfish method. Both methods performed well with a slight difference in trajectory curvature in the worlds with simple obstacles or when initial facing was in a different direction than towards the goal. The Potential field method started failing when it came to medium c-shape obstacles (experiment 8), whereas the Reverse anglerfish method produced a smooth and complete path to the goal.

Further, the robustness tests of the Reverse anglerfish method were performed. Firstly, it was tested on the different severity levels of c-shape obstacles where it was able to solve extremely severe (snail-like) obstacles using a *loop count*. It also copes with obstacles placed inside of the c-shape obstacle (robustness test 5).

In robustness test 6, the robot successfully changed its control angles coordinate system twice in order to approach the goal from the opposite direction than from the start to the goal direction. In another test, the robot was able to solve a c-shape snail-like obstacle when coming to the goal from behind.

Furthermore, the robot can produce a complete path when starting from inside of the spiral thanks to remembering the closest Euclidean distance to the goal achieved in the free mode. This can be observed in robustness tests 8 and 9.

Additionally, the robot successfully employs the emergency manoeuvre, as can be seen in robustness tests 1 and 10. It successfully avoids v-shape obstacles (the v-shape problem described in chapter 2.3.2). However, a situation might occur where the emergency manoeuvre accidentally decreases the loop count, which is later suggested for future development.

4 CONCLUSION AND FUTURE WORK

4.1 Suggested future work and improvements

Incorporation into N-D systems

In the 2.6 chapter, it was demonstrated on the 3-D problem that the developed algorithm is capable of operating in higher dimensional spaces, and it can be further extended into N-D systems. As an example, a multi-joint robot with 25 degrees of freedom, meaning a 25-D problem, can be imagined. When there are placed proper sensors, the algorithm can use the Downhill Simplex method to find and follow the line of least resistance (the highest progress). The principal is still the same; the robot does not have to search through the whole space, but it only searches part of it, relative to the robot's current position (in the 2-D environment, it is control contour, and in the 3-D, it is a control surface). Because the search for the highest progress always starts at the end of the last state of the robot

goal_ratio below 1

Some experiments were done where the *goal_ratio* (presented in chapter 2.5 in formula 4) was allowed to go below 1 when close to obstacles. This resulted that the robot started preferring longer paths to the goal around obstacles. This is an interesting finding that is worth further investigation.

Control contour search restriction

When the robot goes around the obstacle, part of the control contour can appear beyond the detected obstacle boundary. The robot then searches the potential for this part of the control contour as well, which is usually not a problem (figure 51). However, there can occur a situation where the bigger part of the control contour appears beyond the obstacle boundary (usually in the tight places), and the robot might find better progress beyond the outline of the obstacle, thus causing a collision. In future development, it could be implemented that the robot does not consider the part of the control contour that exceeds the obstacle boundary (the part of the control contour indicated by an orange circle in figure 51).

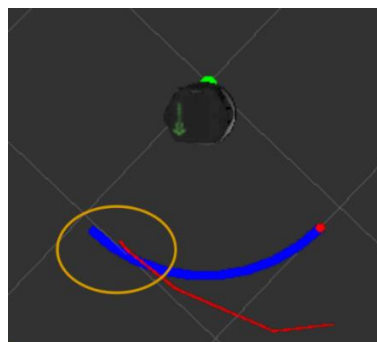


Figure 51: Control contour (blue line) exceeding wall boundary (red line) indicated by an orange circle

Memory or sensor with wider view span

The robot has only a 60-degree view, so it struggles to see the rest of the obstacles when going around corners; therefore, the memory was implemented. The smoothness of the created trajectory is slightly distorted by the use of the memory when it keeps remembering the closest point. The memory remembers only one point that makes the robot deviate from the obstacle, and then the robot comes back to it, thus distorts the trajectory. This could be in future work solved by more complex memory or by using a sensor with a larger view span to produce a smoother path and avoid obstacles with higher precision.

Upgrade emergency manoeuvre

The possible improvement can be made on the emergency manoeuvre to mitigate undesired *loop count*, as can be seen in the robustness test 10 figure 90.

4.2 Challenges and achievements

Primary Objectives

- 1) A novel navigation system (Reverse anglerfish method) was developed that is capable of complete navigation in N-D ($N > 1$). It introduces a new concept of the Progress field where the robot searches for the highest progress distorted by the obstacle potential close to the obstacles in each motion step.
- 2) Completeness capability for various 2-D problems and a 3-D problem has been demonstrated.
- 3) The robustness of the system was tested on different courses containing obstacles of a range of difficulties, and a qualitative evaluation was performed.

Secondary Objectives

- 4) A user-friendly interface was developed for comfortable control of the navigation. The user can manipulate parameters, set the goal coordinates, switch between different modes, and start and stop the robot.
- 5) Incorporation of the system into drone moving in the 3-D space and multi-joint robotic arm representing the N-D system were suggested.
- 6) The performance of the developed system was compared with the existing navigation method, more specifically, with the version of the classical Potential Field.
- 7) Finally, the system was incorporated and tested on the physical robot.

To meet these objectives, the writer was on the learning curve with the ROS, and other additional challenges arose during the development. The restricted vision of the Kinect sensor used for TurtleBot2 caused crashing in certain situations; therefore, to prevent this robot's memory was implemented. The robot was unable to avoid v-shape

obstacles, as mentioned in the 2.3.2 chapter. The emergency manoeuvre was developed to tackle this problem.

The strength of the developed method is that it can be used directly to generate the motion and plan the path using only a local sensor with no need for a global map, and still be able to deal with a wide variety of obstacle difficulties. The developed Reverse anglerfish method is capable of leading the robot out of the c-shape obstacle from mild to extreme (snail-like) severity. It copes with obstacles placed inside of the c-shape obstacle. It allows the robot to approach the goal from behind with the same effectiveness. The robot can also produce a complete path when the start position is located inside the obstacle, like a spiral shape obstacle. With additional improvements, it drives around the corners without crashing and can escape from the v-shape obstacles.

5 APPENDIX 1 – EXPERIMENTS AND ROBUSTNESS TESTS

5.1 Experiment 1: No obstacles, starts facing the goal

Setup

In a free space with no obstacles, the robot is facing the goal.

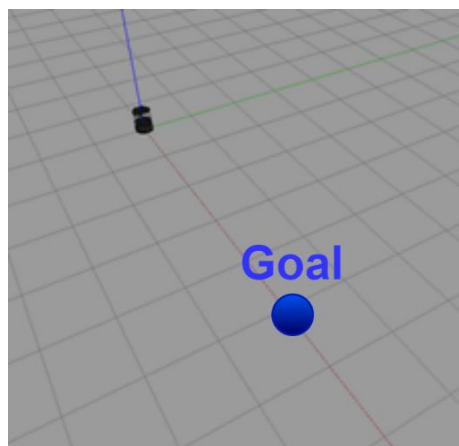


Figure 52: Setup - Experiment 1

Observation

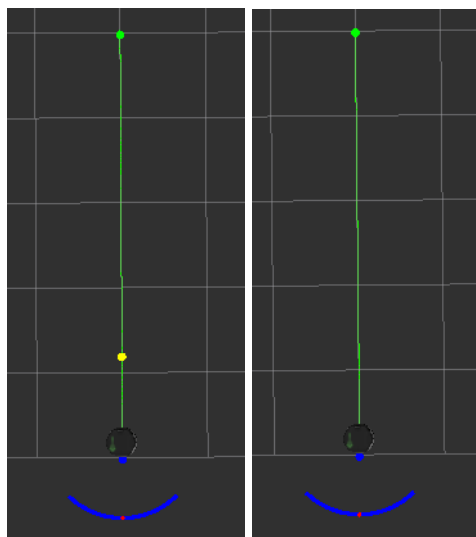


Figure 53: Experiment 1 trajectories – Reverse anglerfish on the left, classical Potential field on the right

<i>Method</i>	Reverse anglerfish	classical Potential field
<i>Success</i>	Yes	Yes
<i>Smooth</i>	Yes	Yes

Inference

Both methods found the same straight, smooth path to the goal.

5.2 Experiment 2: No obstacles, starts facing away from the goal

Setup

Free space with no obstacles, the robot is facing from the goal.

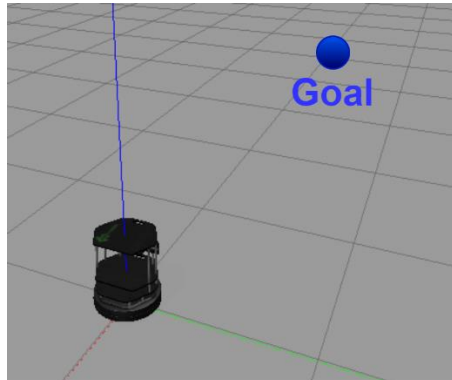


Figure 54: Setup - Experiment 2

Observation

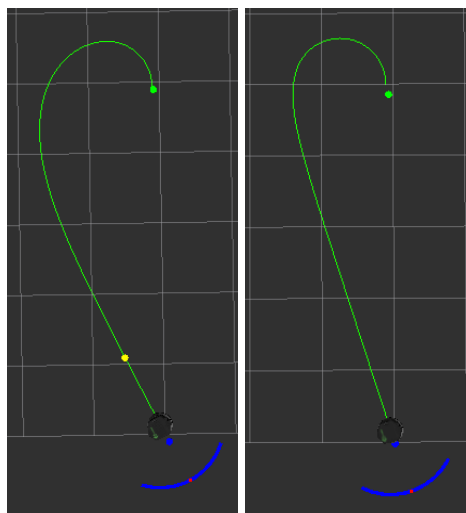


Figure 55: Experiment 2 trajectories – Reverse anglerfish on the left, classical Potential field on the right

<i>Method</i>	Reverse anglerfish	classical Potential field
<i>Success</i>	Yes	Yes
<i>Smooth</i>	Yes	Yes

Inference

At the beginning of the movement, the trajectory of the classical Potential field bends more sharply and then go straight to the goal. On the other hand, the Reverse anglerfish's trajectory is slightly less curved at the beginning but then approaches the goal with higher curvature. This is a result of the Progress field that is not directly directed to the goal as the Potential field. This feature is later exploited when escaping from the more complex obstacles such as the c-shape.

5.3 Experiment 3: Single cube obstacle

Setup

A space with one cube obstacle located halfway between the robot and the target.

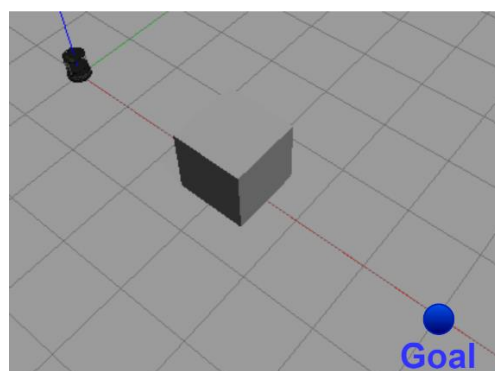


Figure 56: Setup - Experiment 3

Observation

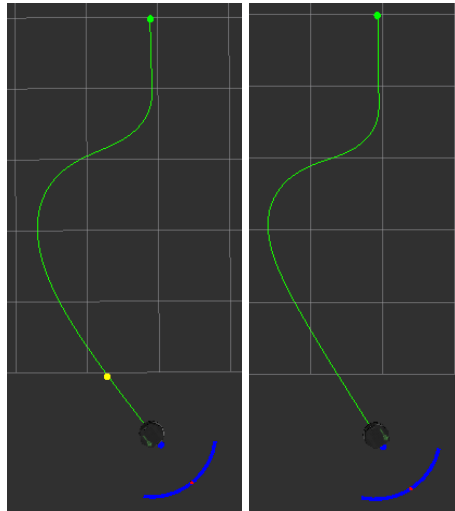


Figure 57: Experiment 3 trajectories – Reverse anglerfish on the left, classical Potential field on the right

<i>Method</i>	Reverse anglerfish	classical Potential field
<i>Success</i>	Yes	Yes
<i>Smooth</i>	Yes	Yes

Inference

Both algorithms effectively avoided the obstacle in a very similar manner. Again, a subtle difference in curvature can be noticed.

5.4 Experiment 4: A wall with a hole obstacle

Setup

A wall with a hole placed between the robot and the goal location.

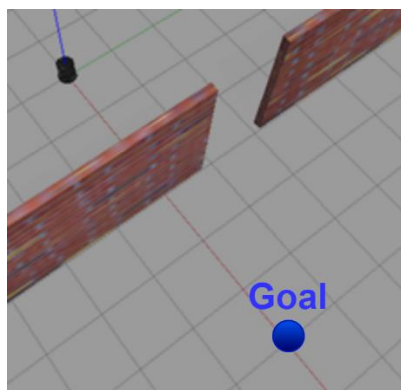


Figure 58: Setup - Experiment 4

Observation

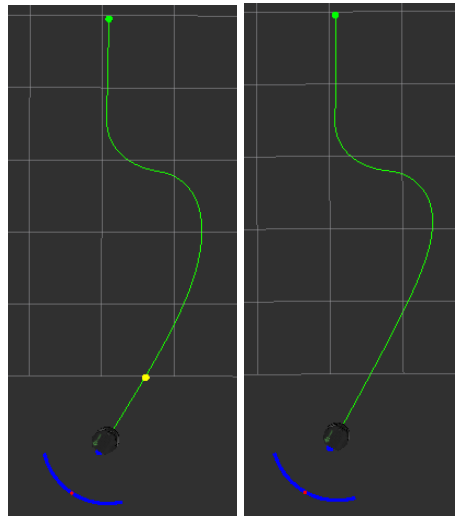


Figure 59: Experiment 4 trajectories – Reverse anglerfish on the left, classical Potential field on the right

<i>Method</i>	Reverse anglerfish	classical Potential field
<i>Success</i>	Yes	Yes
<i>Smooth</i>	Yes	Yes

Inference

Both algorithms passed through the hole in the wall with the same effectivity.

5.5 Experiment 5: Random cube maze

Setup

Ten meters long maze made of cubes with a 0.6 wide gap at the beginning.

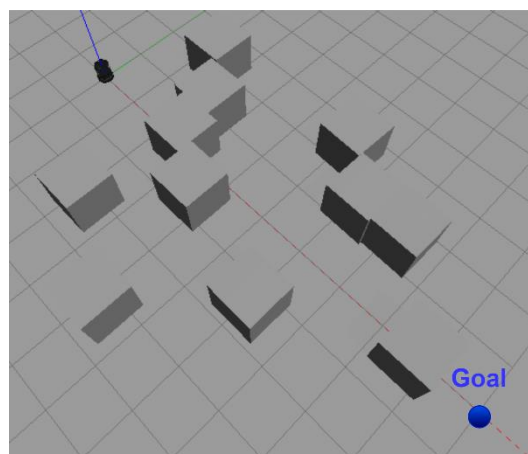


Figure 60: Setup - Experiment 5

Observation

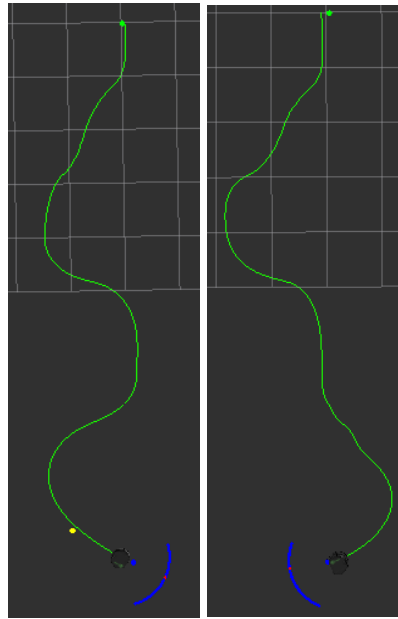


Figure 61: Experiment 4 trajectories – Reverse anglerfish on the left, classical Potential field on the right

<i>Method</i>	Reverse anglerfish	classical Potential field
<i>Success</i>	Yes	Yes
<i>Smooth</i>	Yes	Yes

Inference

Both algorithms passed through the maze without crashing. They managed to go through the gap between the cubes in the upper part of the maze that is only 0.6 m wide, shown in the figure below. The only difference is in the lower part of the trajectory, where each method picked a different side to go around the final cube.

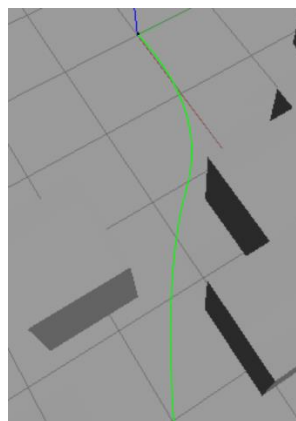


Figure 62: Passing through the 0.6 m wide gap

5.6 Experiment 6: Mild c-shape obstacle

Setup

A mild C-shape obstacle that is 12 meters wide is placed between the robot and the target destination.

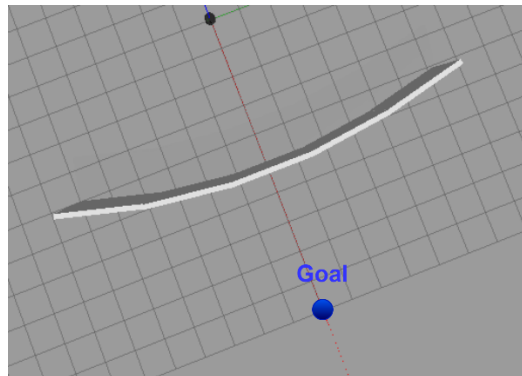


Figure 63: Setup - Experiment 6

Observation

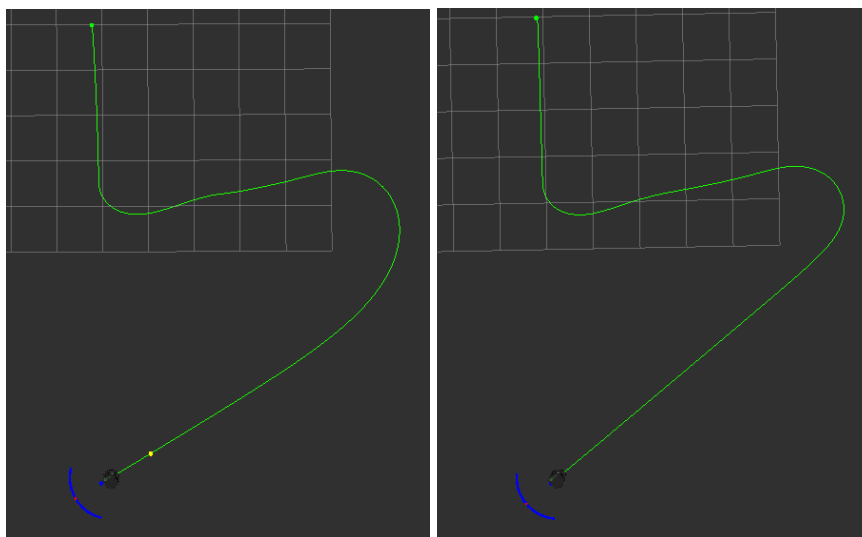


Figure 64: Experiment 6 trajectories – Reverse anglerfish on the left, classical Potential field on the right

<i>Method</i>	Reverse anglerfish	classical Potential field
<i>Success</i>	Yes	Yes
<i>Smooth</i>	Yes	Yes

Inference

Both algorithms were able to exit this obstacle with a smooth path. Again a lower curvature can be noticed with Reverse anglerfish.

5.7 Experiment 7: A long wall obstacle

Setup

Thirty-two meters long wall is placed between the robot and the target destination.

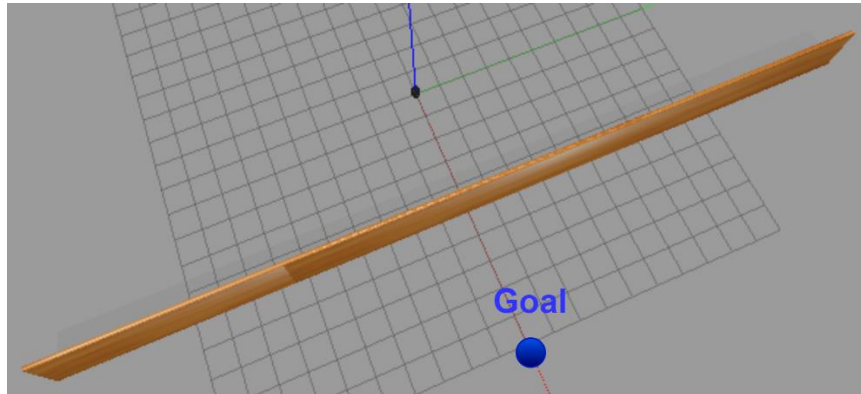


Figure 65: Setup - Experiment 7

Observation

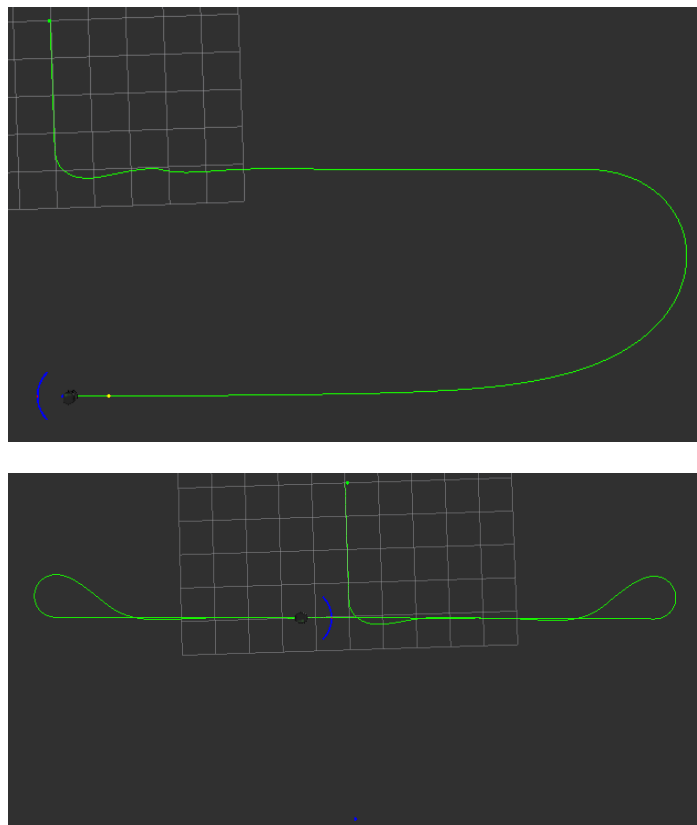


Figure 66: Experiment 7 trajectories – Reverse anglerfish on the top, classical Potential field on the bottom

<i>Method</i>	Reverse anglerfish	classical Potential field
<i>Success</i>	Yes	No
<i>Smooth</i>	Yes	NaN

Inference

The classical algorithm failed due to the nature of the Potential field. The robot follows the wall for some time, but then the goal potential gets too big, making the robot favour to turn for a lower resulting potential. On the contrary, the Reverse anglerfish uses the Progress field that allows a robot to progress towards the goal even if it means going away from it.

5.8 Experiment 8: Medium c-shape obstacle

Setup

A medium c-shape obstacle is placed between the robot and the goal.



Figure 67: Setup - Experiment 8

Observation

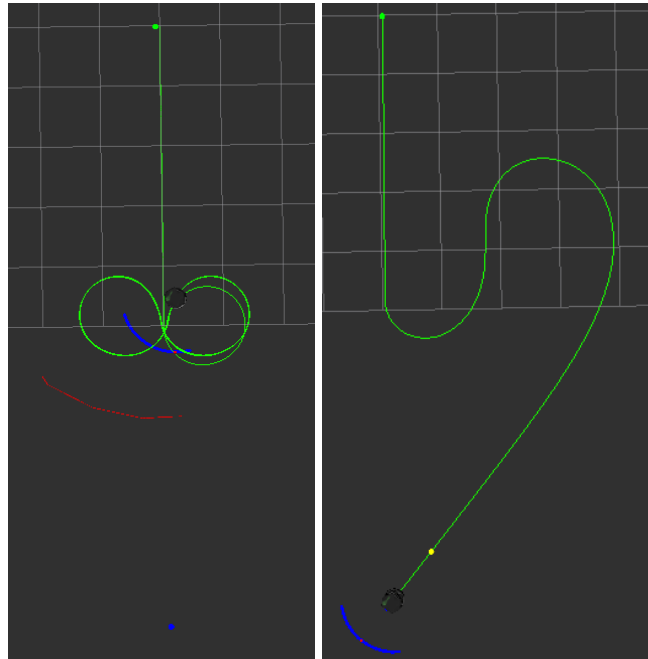


Figure 68: Experiment 8 trajectories – Reverse anglerfish on the left, classical Potential field on the right

<i>Method</i>	Reverse anglerfish	classical Potential field
<i>Success</i>	Yes	No
<i>Smooth</i>	Yes	NaN

Inference

The classical algorithm failed to escape the c-shape obstacle. The goal potential pulled the robot down to the obstacle and subsequently was repelled by the obstacle potential but then pulled back by the goal potential. Reverse anglerfish deviated the lantern point to the left and escaped the obstacle.

5.9 Robustness of Reverse anglerfish method

In the last experiments, it could be seen that the Reverse anglerfish method outperforms the classical Potential field when it comes to escaping from more complex obstacles such as c-shape. The classical Potential field is never able to escape from the c-shape with higher severity than mild. Therefore, in this chapter, the robustness of the Reverse anglerfish is tested.

5.9.1 Robustness test 1: V-shape obstacle

Setup

A medium v-shape obstacle is placed between the robot and the goal.

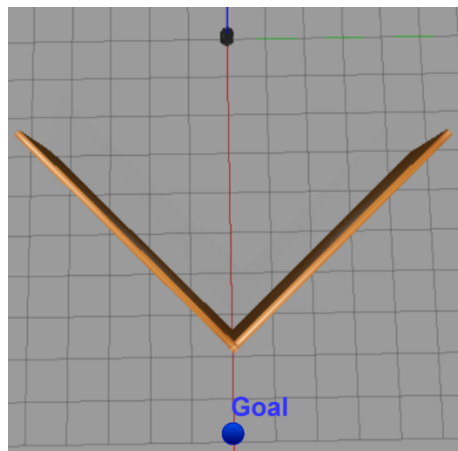


Figure 69: Setup – Robustness test 1

Observation

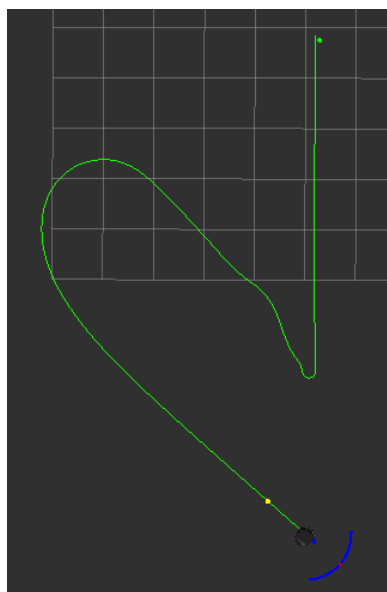


Figure 70: Robustness test 1 trajectory – Reverse anglerfish method

<i>Method</i>	Reverse anglerfish
<i>Success</i>	Yes
<i>Smooth</i>	Yes

Inference

The robot successfully activated an emergency manoeuvre that enabled him to escape from the v-shape obstacle. (the v-shape problem described in 2.3.2)

5.9.2 Robustness test 2: Severe c-shape obstacle

Setup

A severe c-shape obstacle is placed between the robot and the goal.

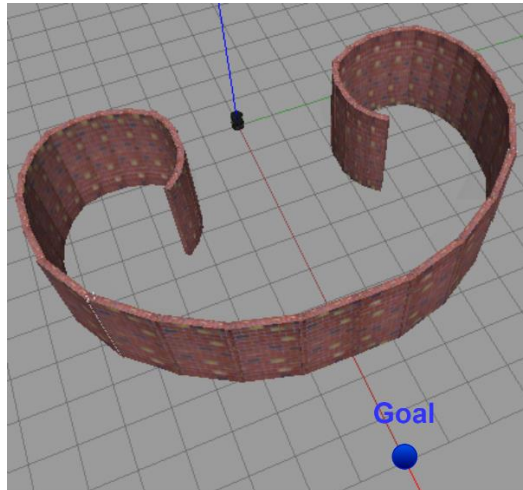


Figure 71: Setup – Robustness test 2

Observation

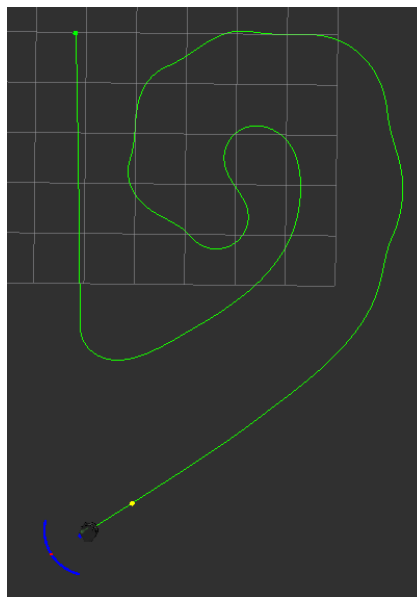


Figure 72: Robustness test 2 trajectory – Reverse anglerfish method

<i>Method</i>	Reverse anglerfish
<i>Success</i>	Yes
<i>Smooth</i>	Yes

Inference

The robot activated deviation mode and was able to escape from the severe c-shape obstacle. Slight bumps can be noticed on the trajectory; they are caused by the obstacle memory. For a reason and suggested improvement, look into the conclusion chapter.

5.9.3 Robustness test 3: C-shape snail obstacle

Setup

Extremely severe c-shape obstacle, called the c-shape snail obstacle, is placed between the robot and the goal.

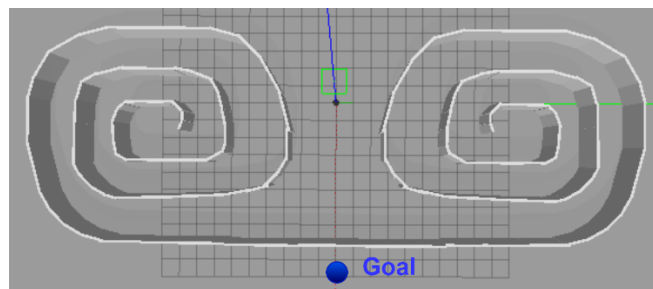


Figure 73: Setup – Robustness test 3

Observation

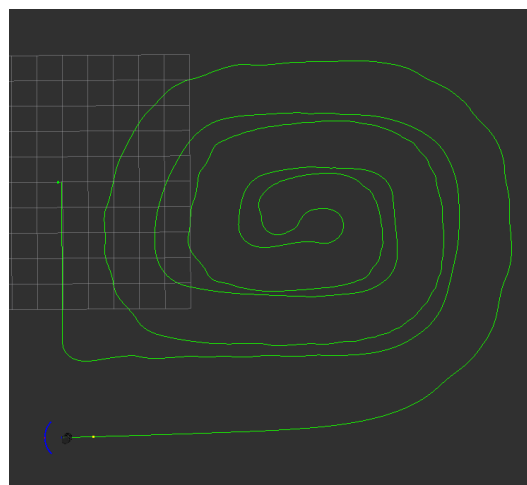


Figure 74: Robustness test 3 trajectory – Reverse anglerfish method

<i>Method</i>	Reverse anglerfish
<i>Success</i>	Yes
<i>Smooth</i>	Yes

Inference

The robot activated deviation mode and correctly counted how many times it got inside (loop count explained earlier) in order to get outside of this extremely severe c-shape obstacle.

5.9.4 Robustness test 4: Deviated c-shape snail obstacle

Setup

The c-shape snail obstacle was deviated and placed between the robot and the goal.

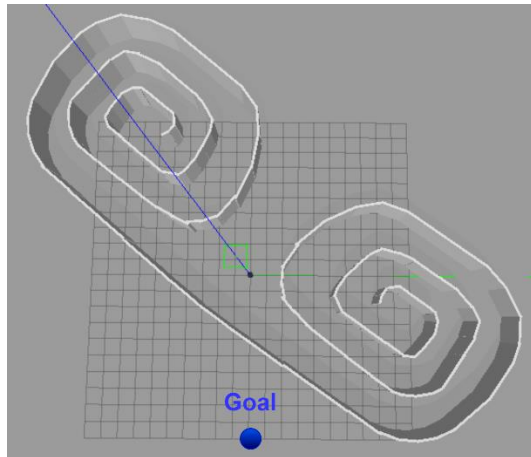


Figure 75: Setup – Robustness test 4

Observation

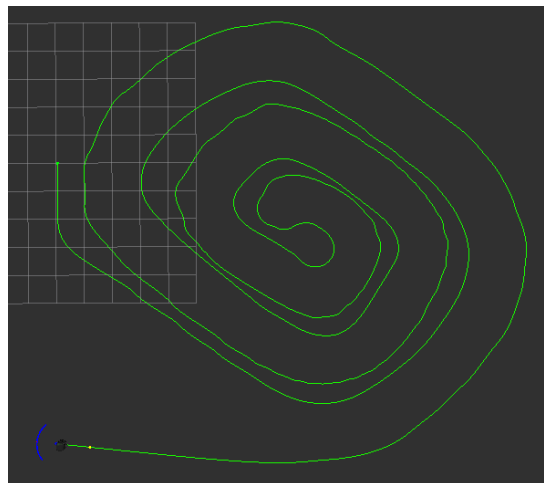


Figure 76: Robustness test 4 trajectory – Reverse anglerfish method

<i>Method</i>	Reverse anglerfish
<i>Success</i>	Yes
<i>Smooth</i>	Yes

Inference

The robot activated deviation mode and managed to escape from the deviated snail obstacle.

5.9.5 Robustness test 5: Obstacles inside c-shape snail obstacle

Setup

The c-shape snail obstacle with the additional obstacles inside was placed between the robot and the goal.



Figure 77: Setup – Robustness test 5

Observation

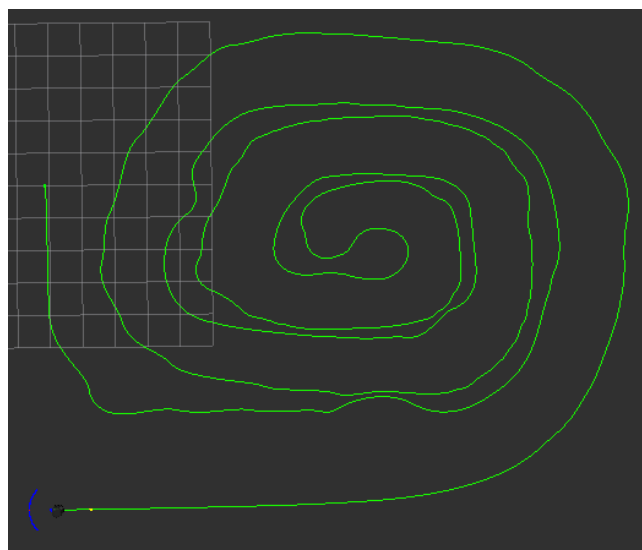


Figure 78: Robustness test 5 trajectory – Reverse anglerfish method

<i>Method</i>	Reverse anglerfish
<i>Success</i>	Yes
<i>Smooth</i>	Yes

Inference

The robot was able to avoid all the obstacles inside the snail and escape the c-shape snail obstacle.

5.9.6 Robustness test 6: Goal closed in the box (inside reversed c-shape)

When the robot is forced to approach the goal from behind, it activates the deviation mode, and the robot is then dragged by the lantern point away. In order to avoid this problem, the robot is able to change its control angles coordinate system.

Setup

The goal is closed with the reversed severe c-shape obstacle.

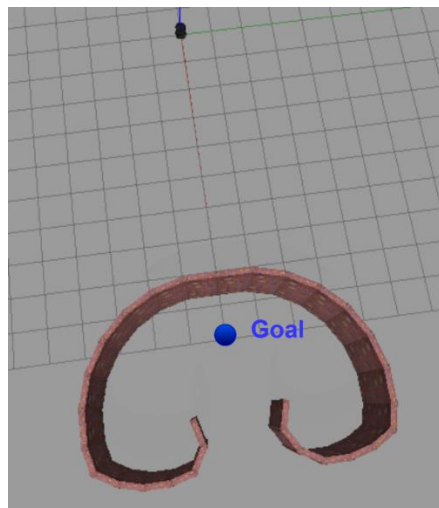


Figure 79: Setup – Robustness test 6

Observation

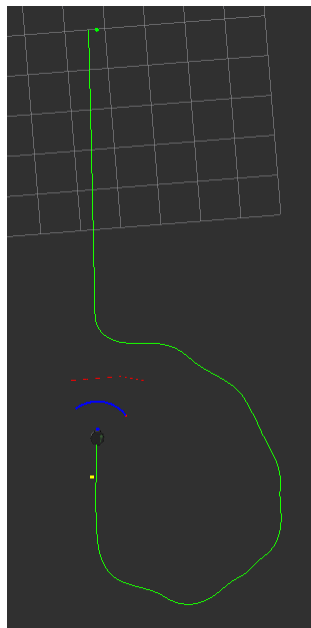


Figure 80: Robustness test 6 trajectory – Reverse anglerfish method

<i>Method</i>	Reverse anglerfish
<i>Success</i>	Yes
<i>Smooth</i>	Yes

Inference

The robot changed its control angle coordinates twice. That allowed him to go straight to the goal without sticking to the wall or being dragged away. The trajectory produced without changing the control angle coordinates can be seen in figure 81.

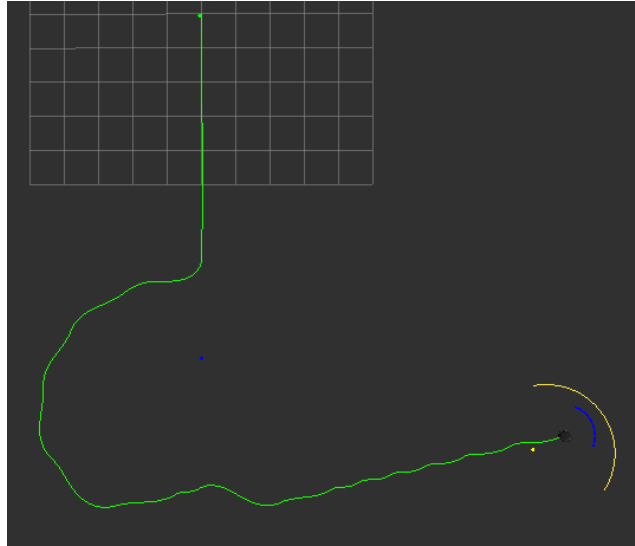


Figure 81: The robot being dragged away from the goal by constant activation of the lantern point deviation

5.9.7 Robustness test 7: Reversed closed c-shape snail obstacle

Setup

The goal is closed with the reversed severe c-shape obstacle.

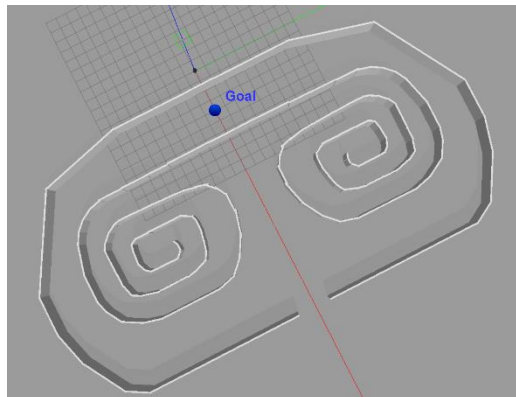


Figure 82: Setup – Robustness test 7

Observation

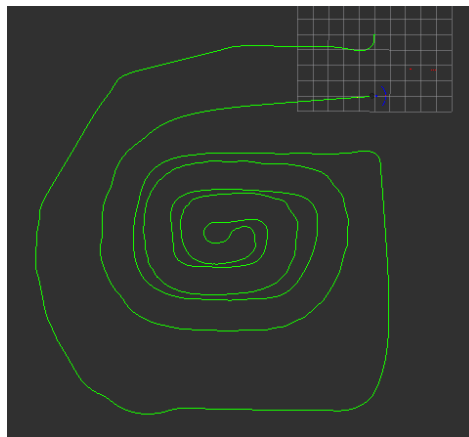


Figure 83: Robustness test 7 trajectory – Reverse anglerfish method

<i>Method</i>	Reverse anglerfish
<i>Success</i>	Yes
<i>Smooth</i>	Yes

Inference

The robot changed its control angle coordinates twice. That allowed him to go straight to the goal without sticking to the wall. The robot activated deviation mode and managed to escape from the deviated snail obstacle.

5.9.8 Robustness test 8: Start inside spiral

Setup

The robot starts inside of the spiral obstacle.



Figure 84: Setup – Robustness test 8

Observation

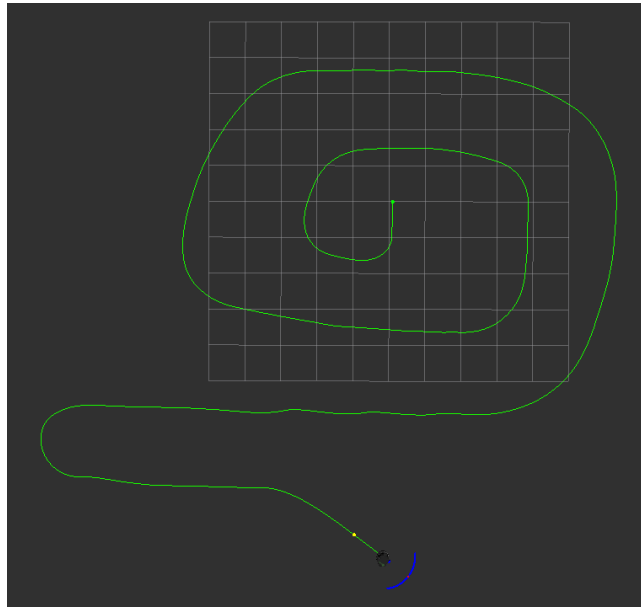


Figure 85: Robustness test 8 trajectory – Reverse anglerfish method

<i>Method</i>	Reverse anglerfish
<i>Success</i>	Yes
<i>Smooth</i>	Yes

Inference

The robot remembers the closest Euclidean distance to the goal achieved in the free mode. When the robot activates deviation mode by the dive angle and gets closer to the goal than in the free mode, it indicates that it escaped the obstacle, and it restarts *loop_count* and deactivates the deviation.

Without this feature, the algorithm would have driven the robot past the goal back into the obstacle because the *loop_count* would have caused the deviation by the dive angle, as shown below.

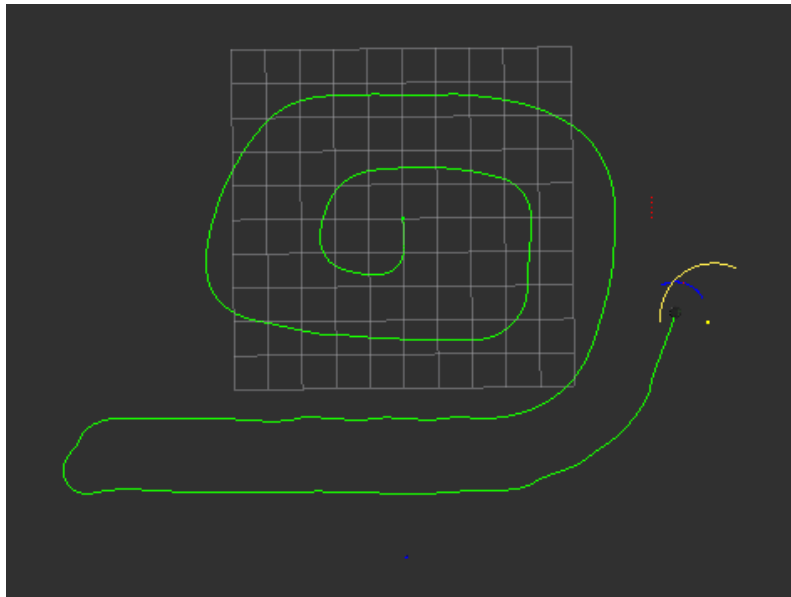


Figure 86: The robot, without remembering the closest Euclidean distance to the goal achieved in free mode

5.9.9 Robustness test 9: Start inside c-shape snail obstacle

Setup

The robot starts inside of the c-shape obstacle.

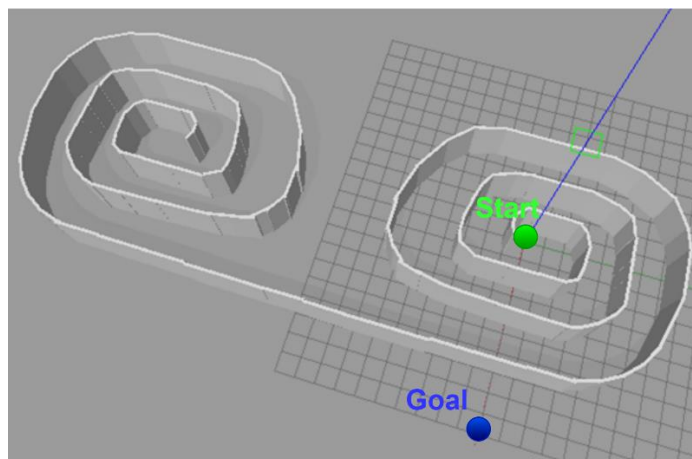


Figure 87: Setup – Robustness test 9

Observation

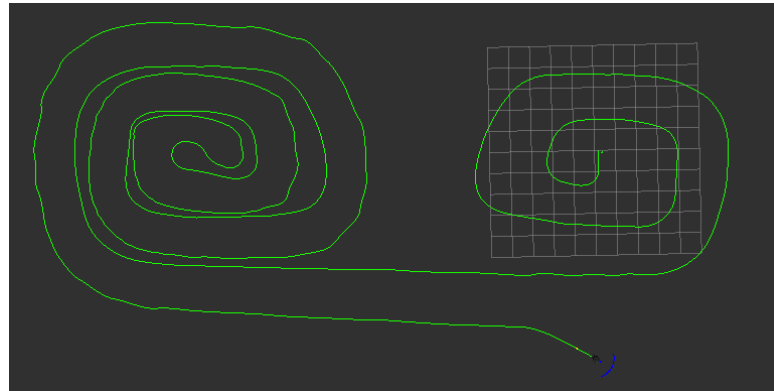


Figure 88: Robustness test 9 trajectory – Reverse anglerfish method

<i>Method</i>	Reverse anglerfish
<i>Success</i>	Yes
<i>Smooth</i>	Yes

Inference

The robot again used the feature of but also was able to escape from the c-shape snail obstacle when the starting position was inside of the obstacle.

5.9.10 Robustness test 10: V-shape inside the snail

Setup

The c-shape snail obstacle is placed between the robot and the goal with the v-shape obstacle inside.

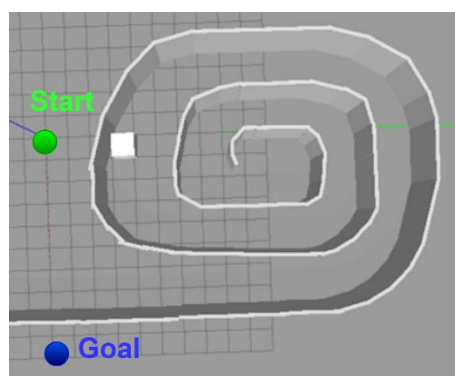


Figure 89: Setup – Robustness test 10

Observation

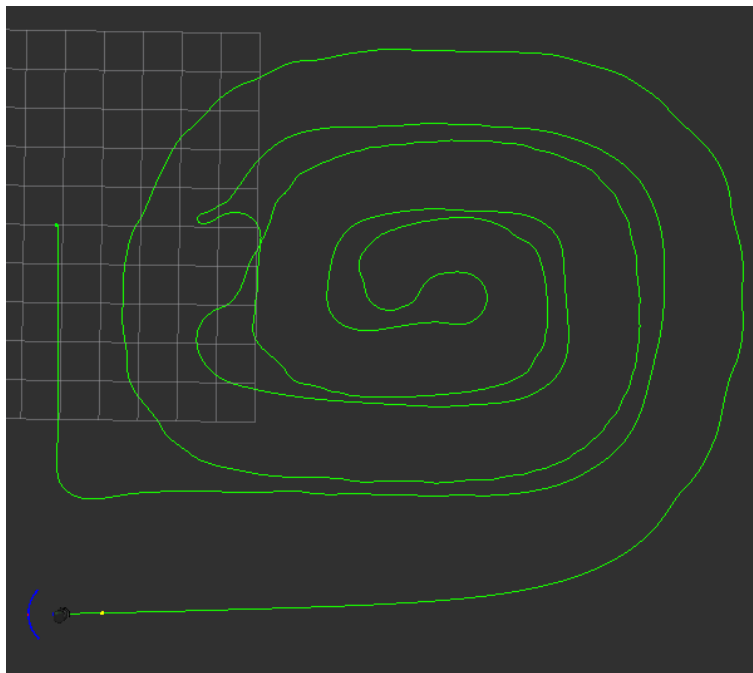


Figure 90: Robustness test 10 trajectory – Reverse anglerfish method

<i>Method</i>	Reverse anglerfish
<i>Success</i>	Yes
<i>Smooth</i>	Yes

Inference

The robot successfully escapes from the c-shape snail obstacle with the v-shape inside because it manages to keep track of control angles during the emergency manoeuvre.

However, there is a risk that the robot is forced to pick the escape turn that creates a loop. It disrupts the *loop count*, as shown in figure 91. The robot then does not activate the dive as desired; therefore, there is a possibility for future development.

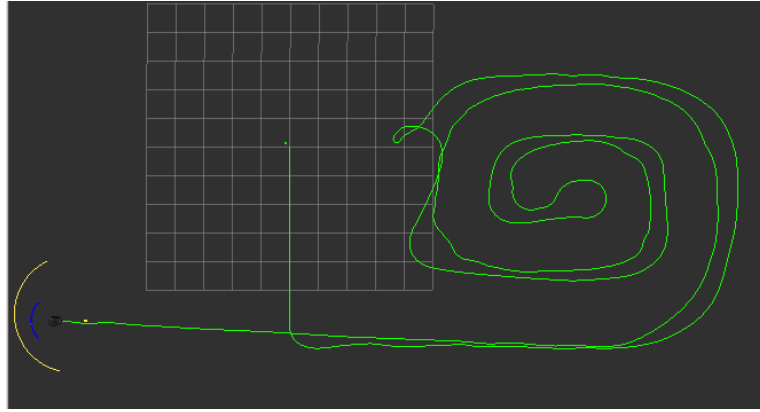


Figure 91: Emergency manoeuvre causing disruption of the *loop count*

6 APPENDIX 2 - PHYSICAL TURTLEBOT IMPLEMENTATION

The TurtleBot is controlled via two computers. One is called Turtlebot NB (notebook) and is attached to TurtleBot all the time. The other one is called master NB and is used to ssh to the Turtlebot NB and is used to control the robot remotely.

In this project, both computers were using Ubuntu 14 and ROS indigo distribution. Ubuntu 14 can be downloaded from here [30], and ROS indigo installation instructions are here [31]. Then it is important to configure the network for both computers by following the tutorial here [32]. The TurtleBot packages also have to be installed (sufficient to do it only on Turtlebot NB); the tutorial is available from here [33]. Additionally, install the drivers and OpenNI on Turtlebot NB to enable it to receive information from the Kinect camera; a full tutorial is available here [34]. Also, make sure the program developed for this project is placed on the Turtlebot NB.

After all of this is set, turn on Turtlebot NB, connect both USBs from TurtleBot to the NB (one for Kinect and one for Kuboki base) and switch on the Turtlebot on the side. Then you will need to open two terminals on the master NB and ssh to the Turtlebot NB from both by typing: `ssh TurtleNBname@TurtleNB_IP`. In the first one, run the command from the figure below to start the Kuboki base.

```
roslaunch turtlebot_bringup minimal.launch
```

Figure 92: Starts TurtleBot's Kuboki base

In the second terminal, run the command from the figure below to start the 3D sensor (Kinect).

```
roslaunch turtlebot_bringup 3dsensor.launch
```

Figure 93: Starts 3D sensor

Then you need to open two more terminals and again ssh to the Turtlebot NB, this time by typing: `ssh -X TurtleNBname@TurtleNB_IP`. Additional `-X` specification allows displaying GUI on the master NB. In the first one, open Rviz to visualise what the robot sees by typing the command below.

```
roslaunch turtlebot_rviz_launchers view_robot.launch
```

Figure 94: Opens Rviz

In the second terminal, navigate to the folder with the script written for this project called `navigation_GUI.py` created for this project and run the following. After this step, the setup is complete.

```
python navigation_GUI.py
```

Figure 95: Runs GUI created for this project

ETHICS

UNIVERSITY OF ST ANDREWS
TEACHING AND RESEARCH ETHICS COMMITTEE (UTREC)
SCHOOL OF COMPUTER SCIENCE
PRELIMINARY ETHICS SELF-ASSESSMENT FORM

This Preliminary Ethics Self-Assessment Form is to be conducted by the researcher, and completed in conjunction with the Guidelines for Ethical Research Practice. All staff and students of the School of Computer Science must complete it prior to commencing research.

This Form will act as a formal record of your ethical considerations.

Tick one box

- ☐ **Staff Project**
☒ **Postgraduate Project**
☐ **Undergraduate Project**

Title of project

Creative Autonomous Navigation by Mobile Robots in Unstructured Spaces

Name of researcher(s)

Pavel Sobotka

Name of supervisor (for student research)

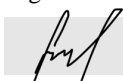
Dr Mike Weir

OVERALL ASSESSMENT (to be signed after questions, overleaf, have been completed)

Self audit has been conducted **YES** ☒ **NO** ☐

There are no ethical issues raised by this project

Signature Student or Researcher



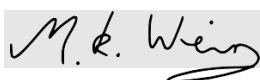
Print Name

PAVEL SOBOTKA

Date

03/06/2021

Signature Lead Researcher or Supervisor



Print Name

M.K. WEIR

Date

04/06/2021

This form must be date stamped and held in the files of the Lead Researcher or Supervisor. If fieldwork is required, a copy must also be lodged with appropriate Risk Assessment forms. The School Ethics Committee will be responsible for monitoring assessments.

REFERENCES

- [1] RUSSELL, Stuart J., Peter NORVIG a Ernest DAVIS. *Artificial intelligence: a modern approach*. 3rd ed. Upper Saddle River: Prentice Hall, 2010. Prentice Hall series in artificial intelligence. ISBN 978-0-13-604259-4.
- [2] RUBIO, Francisco, Francisco VALERO a Carlos LLOPIS-ALBERT. A review of mobile robots: Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems* [online]. 2019, **16**(2), 172988141983959. ISSN 1729-8814, 1729-8814. Available from: doi:10.1177/1729881419839596
- [3] MESTER, Gyula. Applications of Mobile Robots. nedatováno, 7.
- [4] SIEGWART, Roland, Illah Reza NOURBAKHS a Davide SCARAMUZZA. *Introduction to autonomous mobile robots*. 2nd ed. Cambridge, Mass: MIT Press, 2011. Intelligent robotics and autonomous agents. ISBN 978-0-262-01535-6.
- [5] ALATISE, Mary B. a Gerhard P. HANCKE. A Review on Challenges of Autonomous Mobile Robot and Sensor Fusion Methods. *IEEE Access* [online]. 2020, **8**, 39830–39846. ISSN 2169-3536. Available from: doi:10.1109/ACCESS.2020.2975643
- [6] DAVIES, E. R. *Computer Vision: Principles, Algorithms, Applications, Learning*. B.m.: Academic Press, 2017. ISBN 978-0-12-809575-1.
- [7] JANA, Abhijit. *Kinect for Windows SDK programming guide: build motion-sensing applications with Microsoft's Kinect for Windows SDK quickly and easily*. Birmingham: Packt Publ, 2012. Community experience distilled. ISBN 978-1-84969-238-0.
- [8] SOBOTKA, Pavel. *HUMAN MACHINE COLLABORATION - Using Computer Vision*. Brno, 2019. Brno University of Technology.
- [9] RAVANKAR, Ankit A., Yohei HOSHINO, Abhijeet RAVANKAR, Lv JIXIN, Takanori EMARU a Yukinori KOBAYASHI. Algorithms and a Framework for Indoor Robot Mapping in a Noisy Environment Using Clustering in Spatial and Hough Domains. *International Journal of Advanced Robotic Systems* [online]. 2015, **12**(3), 27. ISSN 1729-8814, 1729-8814. Available from: doi:10.5772/59992
- [10] GONZALEZ, David, Joshue PEREZ, Vicente MILANES a Fawzi NASHASHIBI. A Review of Motion Planning Techniques for Automated Vehicles. *IEEE Transactions on Intelligent Transportation Systems* [online]. 2016, **17**(4), 1135–1145. ISSN 1524-9050, 1558-0016. Available from: doi:10.1109/TITS.2015.2498841
- [11] PANDEY, Anish. Mobile Robot Navigation and Obstacle Avoidance Techniques: A Review. *International Robotics & Automation Journal* [online].

- 2017, 2(3) [vid. 2021-06-13]. ISSN 25748092. Available from: doi:10.15406/iratj.2017.02.00023
- [12] PUGH, J. a A. MARTINOLI. Relative localization and communication module for small-scale multi-robot systems. In: *2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.: Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* [online]. Orlando, FL, USA: IEEE, 2006, s. 188–193 [vid. 2021-06-14]. ISBN 978-0-7803-9505-3. Dostupné z: doi:10.1109/ROBOT.2006.1641182
- [13] TurtleBot 2 - Open source personal research robot. *Clearpath Robotics* [online]. [vid. 2021-06-14]. Available from: <https://clearpathrobotics.com/turtlebot-2-open-source-robot/>
- [14] *TurtleBot2* [online]. [vid. 2021-06-14]. Available from: <https://www.turtlebot.com/turtlebot2/>
- [15] *Documentation - ROS Wiki* [online]. [vid. 2021-06-14]. Available from: <http://wiki.ros.org/>
- [16] *gazebo_ros_pkgs - ROS Wiki* [online]. [vid. 2021-07-31]. Available from: http://wiki.ros.org/gazebo_ros_pkgs
- [17] *Gazebo* [online]. [vid. 2021-07-31]. Available from: <http://gazebosim.org/>
- [18] *rviz - ROS Wiki* [online]. [vid. 2021-07-31]. Available from: <http://wiki.ros.org/rviz>
- [19] WINKLER, Joab R. Numerical recipes in C: The art of scientific computing, second edition. *Endeavour* [online]. 1993, 17(4), 201. ISSN 01609327. Available from: doi:10.1016/0160-9327(93)90069-F
- [20] Figure 2: An iteration of the Nelder-Mead method over two-dimensional... *ResearchGate* [online]. [vid. 2021-07-31]. Available from: https://www.researchgate.net/figure/An-iteration-of-the-Nelder-Mead-method-over-two-dimensional-space-showing-point-p-min_fig13_273390709
- [21] Alternative downloads. *Ubuntu* [online]. [vid. 2021-08-02]. Available from: <https://ubuntu.com/download/alternative-downloads>
- [22] *melodic/Installation/Ubuntu - ROS Wiki* [online]. [vid. 2021-08-02]. Available from: <http://wiki.ros.org/melodic/Installation/Ubuntu>
- [23] *Robots/TurtleBot - ROS Wiki* [online]. [vid. 2021-08-02]. Available from: <http://wiki.ros.org/Robots/TurtleBot>
- [24] HUANG, Guanheng. *Turtlebot2-On-Melodic* [online]. Shell. 2021 [vid. 2021-08-02]. Available from: <https://github.com/gaunthan/Turtlebot2-On-Melodic>
- [25] KRONTON ROBOTICS&AI. *Tip4/ Installing the Turtlebot2 package on ROS melodic.* [online]. [vid. 2021-08-02]. Available from: <https://www.youtube.com/watch?v=rniyH8dY5t4&t=86s>

- [26] G. KLANCAR, A. ZDEŠAR, S. BLAŽIĆ, I. ŠKRJANC. *Wheeled Mobile Robotics* [online]. B.m.: Elsevier, 2017 [vid. 2021-08-02]. ISBN 978-0-12-804204-5. Dostupné z: doi:10.1016/B978-0-12-804204-5.09999-6
- [27] DUDEK, Gregory a Michael JENKIN. *Computational Principles of Mobile Robotics*. B.m.: Cambridge University Press, 2010. ISBN 978-0-521-69212-0.
- [28] *tf - ROS Wiki* [online]. [vid. 2021-08-02]. Available from: <http://wiki.ros.org/tf>
- [29] GARRIDO, Santiago, Luis MORENO, Dolores BLANCO a Fernando MARTIN. FM2: A real-time fast marching sensor-based motion planner. In: *2007 IEEE/ASME international conference on advanced intelligent mechatronics: 2007 IEEE/ASME international conference on advanced intelligent mechatronics* [online]. Zurich, Switzerland: IEEE, 2007, s. 1–6 [vid. 2021-06-12]. ISBN 978-1-4244-1263-1. Available from: doi:10.1109/AIM.2007.4412505
- [30] *Ubuntu 14.04.6 LTS (Trusty Tahr)* [online]. [vid. 2021-08-08]. Available from: <https://releases.ubuntu.com/14.04/>
- [31] *indigo/Installation/Ubuntu - ROS Wiki* [online]. [vid. 2021-08-08]. Available from: <http://wiki.ros.org/indigo/Installation/Ubuntu>
- [32] *turtlebot/Tutorials/indigo/Network Configuration - ROS Wiki* [online]. [vid. 2021-08-08]. Available from: <http://wiki.ros.org/turtlebot/Tutorials/indigo/Network%20Configuration>
- [33] *turtlebot/Tutorials/indigo/Turtlebot Installation - ROS Wiki* [online]. [vid. 2021-08-08]. Available from: <http://wiki.ros.org/turtlebot/Tutorials/indigo/Turtlebot%20Installation>
- [34] *Learn TurtleBot and ROS* [online]. [vid. 2021-08-08]. Available from: <https://learn.turtlebot.com/>