Technische Universiteit
Eindhoven
University of Technology

Department of Mechanical Engineering
Control Systems Technology (CST) Group
Tech-united @home

Integration of Environment Manipulation within The Navigation Topology

Bachelor End Project

Elmasry, Ismail

Supervisors:
dr. Elena Torta
Peter van Dooren

Final version

Eindhoven, July 2022

Abstract

In this paper the problem of robot navigation among movable objects is addressed. In general robot navigation in dynamic environments is an active research area where a lot of progress have been done over the years. However, the complexity of the path planning problem increases when the robot needs to account for object manipulations in its plan to free up a desired path. Therefore, this concept of navigation among movable objects has a lot of applications in domestic and rescue robots, where the environment is highly unstructured and may require some extra navigation functionalities. Moreover, This report discusses an approach to solving a simplified version of the NAMO problem with one movable object considered in its environment. The solution combines the concepts from the work of Stilman and Kuffner [2004] and Wu et al. [2010] to reach a locally optimal solution that is adaptable to different dynamic obstacles.

Preface

This work is part of my bachelors thesis conducted together with Tech-United@home. The concept of this research is to act as a first implementation step towards more robust navigation among movable objects, such as doors, chairs and small objects which are known to be movable. This is a very interesting and highly active research area within robot navigation as it allows robots to interact with their environment to enhance their navigation capabilities. In this paper a lot of focus is put on the plan generation process and concepts. However, future work still needs to be done on object classification, detection and object manipulation. Therefore, the integration methods and concepts will be discussed in section 6.1. That been said, I would like to thank Peter Van Dooren for the close supervision and assistance thought the project, as well as, dr. Elena Torta and dr.ir.Rene Van De Molengraft. The hard work put in this research paper would not be completed without their assistance and valuable knowledge sharing.

Contents

Contents **iv**

List of Figures **v**

1 Introduction **1**
1.1 Background & Motivation . . . 1
1.1.1 Mobile Robot Navigation . . . 1
1.2 Problem Description: . . . 3
1.2.1 Problem Definition . . . 4
1.2.2 Deliverable and Performance measures . . . 4
1.2.3 Report Outline . . . 4

2 Literature Study **6**
2.1 Planning spaces . . . 6
2.2 Path planning algorithms . . . 7
2.2.1 Planning Components . . . 7
2.2.2 Graph Search Algorithms . . . 7
2.3 Local Planning . . . 10
2.4 Navigation Stacks . . . 10
2.4.1 ROS Navigation stack . . . 10
2.4.2 TU/e Navigation stack for Hero Robot . . . 11
2.5 Navigational planners for movable objects . . . 12
2.5.1 World Modeling . . . 12
2.5.2 Cost Function definition and optimization . . . 13
2.5.3 Object Manipulation Techniques . . . 13
2.5.4 Planning Algorithms . . . 14

3 Conceptual Approaches **15**
3.1 Conceptualization . . . 15
3.1.1 Conceptual approach 1 - Set Point Definition: . . . 15
3.1.2 Conceptual approach 2 - Configuration space Partitioning: . . . 16
3.1.3 Conceptual approach 3 - Local & Global map separation: . . . 16
3.2 Extension & Reformulation . . . 17

4 Implementation **19**

5 Experimental Results **22**
5.1 Experimental setup . . . 22
5.2 Scenarios . . . 22
5.2.1 **Scenario - 1 - No Constraining Object** . . . 22
5.2.2 **Scenario - 2 - Constraining Object not Blocking Path** . . . 23
5.2.3 **Scenario - 3 - Constraining Object Blocking Path** . . . 24
5.2.4 **Scenario - 4 - Constraining Object at different location** . . . 24

5.2.5 **Scenario - 5 - Static Constraining Object** 25
5.3 Discussion 25

6 Future Work and Conclusion **26**
6.1 Future Work 26
6.2 Conclusion 27

Bibliography **28**

Appendix **29**

A Examples of Bio-mimetic technologies **30**

B Planning and Organization **31**

C Comparative Table of NAMO Planning Algorithms **34**

List of Figures

1.1 Occupancy grid map (Santana et al. [2011]) 2
1.2 Model-driven world model with semantic information Pronobis and Jensfelt [2012] 3
1.3 Problem Scenario 3

2.1 Configuration vs work space (Coenen) 7
2.2 BFS node expansion example on a 4-connected grid. (Coenen) 8
2.3 DFS node expansion example on a 4-connected grid. Coenen 8
2.4 Comparative example between A* and Dijkstra's search algorithms. (Coenen) . . . 9
2.5 ROS Navigation Stack (Appeldoorn) 10
2.6 TU/e Navigation Stack (Appeldoorn) 11

3.1 Visualization of conceptual design1 with pre-defined set points 15
3.2 Visualization of conceptual design1 with pre-defined set points 16
3.3 a) 17
3.4 b) 17
3.5 separation between a) map for global planner and b) map for local planner 17

6.1 Proposed Navigation Stack Architecture 26

A.1 (a)Atlas an intelligent humanoid robot. (b) DelFly micro a flapping wing robot. (c) JESSIKO an under water fin propelled robot. (d) four legged soft turtle robot (Wang et al. [2021]) 30

B.1 implementation planning 33

C.1 Comparative table of NAMO planning algorithms. (Renault et al. [2019]) 35

Chapter 1

Introduction

1.1 Background & Motivation

The need for domestic robots that can assist in everyday tasks around the house is on the rise, which puts more pressure on the development of robots that can achieve higher levels of autonomy. This requirement becomes more of a necessity in order to allow the robots to operate in these highly unstructured environments. As a bio-mimetic technology, robots were always inspired by the surrounding biological organisms (the reader is referred to appendix A for examples). This not only assisted in solving complex operation problems by mimicking how these living organisms adapted their biology to operate in unstructured environments, but also generated a better understanding of the overall requirements that govern the physical world. Similarly, this approach can be taken by analysing how humans operate in the household and consequently develop robots that can, to some extent, function in a similar fashion.

Moreover, the tasks that these domestic service robots need to accomplish, require different functions, such as, navigation, object detection, object manipulation, environment perception, interaction with humans and many more. However, this study will merely focus on formulating and developing an approach for improving navigation in these unstructured environments, by integrating object manipulations in the navigation plan. This is in fact, similar to how humans navigate in the household, where they can manipulate certain objects like doors, chairs, small boxes that may be blocking the desired path. In contrast to humans, the robot cannot comprehend that by manipulating these objects, which act as obstacles, the once blocked desired path can now be followed. As a result, the robot goes in a livelock state; where it can no longer follow the navigation path, since no collision free path in its instantaneous world model can be observed.

Furthermore, this instills the need to develop alterations for the navigation stack so that it comprises of certain object manipulations, allowing the generation of plans towards the end goal. In order to achieve this, firstly, a definition of how domestic robots perceive, manipulate and navigate the world around them needs to be formulated.

1.1.1 Mobile Robot Navigation

Navigation is an important task when it comes to domestic robots, as it allows the robot to move around its environment without the risk of collisions that can lead to permanent damage. In order to do so, the robot needs to have a good understanding of its environment and where its positioned in it. These two requirements are expressed in the notions of *mapping* and *localization*, where the robot creates a map (*world model*) of the current actual world and identifies its position in it w.r.t other geometric features in the environment. As mentioned above, these robots operate in highly unstructured environments, which may not be considered static. This creates the strong

dependency on sensory data to update the world model accordingly. Allowing the robotic system to construct collision free paths to the end goal position.

Def 1.1.1. *World Model:* It is the spatial and temporal representation of an instance of the real environment. This representation is used by the robot to interact with its perception of the real world.

Def 1.1.2. *Mapping:* It is the concept of acquiring and keeping track of the spatial model of the environment using sensory data. More on this mapping problem can be found in the work of (Thrun [2002]).

Def 1.1.3. *Localization:* It is the concept of location awareness of a robotic system w.r.t features in the environment.

The concepts of *localization* and *mapping* usually come together with the implementation of **SLAM** algorithms (Simultaneous Localization and Mapping). These algorithms allow synchronized location awareness,as well as, data recordings of the robot's environment to construct a virtual instance of the world state. In fact, such algorithms are a key to autonomous robot navigation in indoor environments, and the determining of which algorithm to use is dependent on the application (González-Baños and Latombe).

The notion of world model construction is very broad but crucial in robotics, where it involves not only the navigation component, but also, manipulation and many others. Its cruciality lies in the fact that it supplies the required information of the environment to the different components, in order to successfully complete the task. Moreover, *World models* can be categorized into two different sections that predominantly focus on the data acquisition and data visualization methods.

Firstly, **grid based, data-driven models**: It is the generation of a world model using purely sensory data and usually represented in an occupancy map, where each cell can either be empty or occupied. An example of an occupancy grid map generated using Gmapping can be seen in figure 1.1.

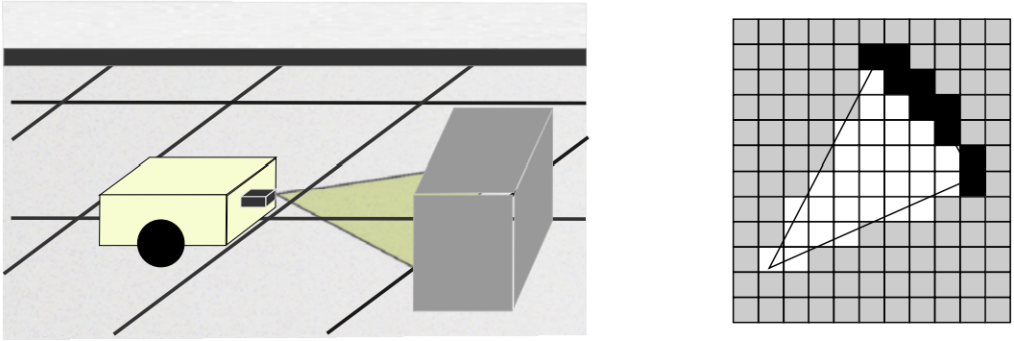


Figure 1.1: Occupancy grid map (Santana et al. [2011])

Secondly, **object-oriented, model-driven models**: It is the generation of a world model by utilizing specific models of the objects available in the environment. Here, the sensory data is combined with the information on preset objects available in the world model. This has a benefit of adding symbolic and/or semantic information to the world model.

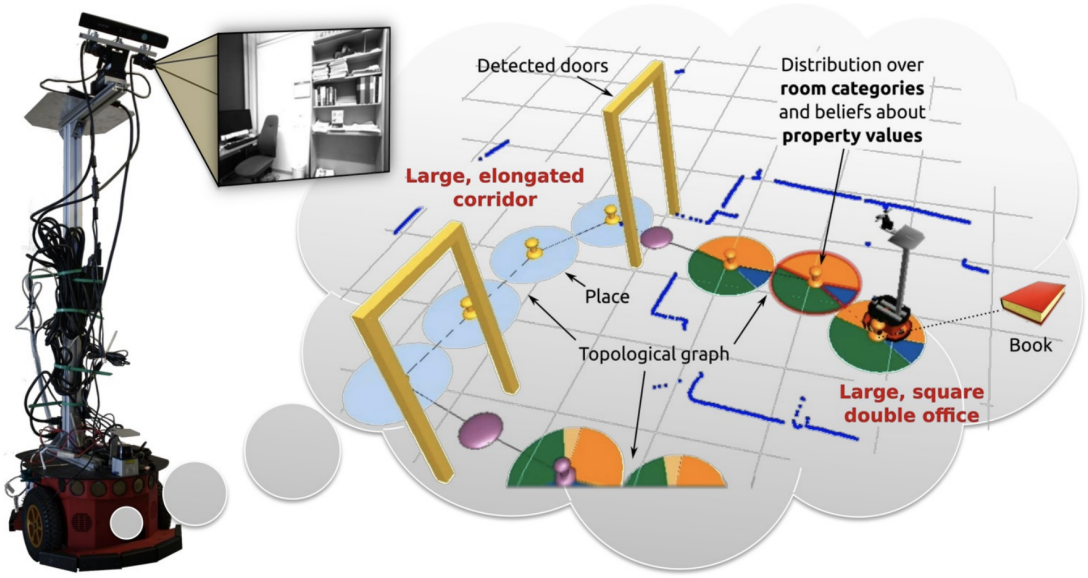


Figure 1.2: Model-driven world model with semantic information Pronobis and Jensfelt [2012]

1.2 Problem Description:

In this section, the problem scenario will be described. In an attempt to illustrate this, figure 1.3 shows a simplified scenario, where the problem definition is limited to one manipulable object, which is the door. The robot here is situated in the predefined world model, where a closed door stands in the way between the robot pose R_1 and the end goal R_2. Therefore, the robot cannot see a possible path between its current pose and its desired end pose and consequently will not navigate.

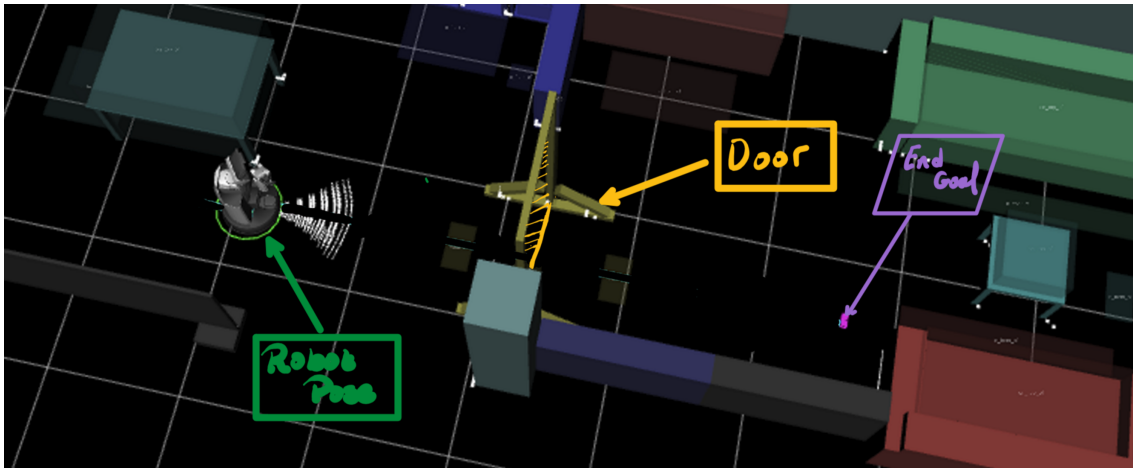


Figure 1.3: Problem Scenario

This problem description can then be extended to also apply to more than one kind of dynamic manipulable objects such as chairs and small boxes.

1.2.1 Problem Definition

In order to formulate the problem definition, some assumptions and requirements need to be taken into account:

- Robot can manipulate Known objects by pushing at set contact points.[1]
- Robot is the only autonomous entity in the environment.
- Robot can detect dynamic obstacles.
- The dynamic obstacles are placed in the environment in such a way to ensure linearity of the problem. [2].

This ensures the validity of the model and allows a better formulation of the problem definition, which is defined as follows:

Design and implement a planner which combines motion planning and object manipulation, using geometric and symbolic information from the world model to allow navigation among movable objects.

1.2.2 Deliverable and Performance measures

The function of the planner is to output a conceptualized navigation plan that iterates motion and the objects to be manipulated until the end goal is reached. This is done by testing on different maps that contain a variable number of dynamic and static objects with different configurations. Therefore, ensuring that the algorithm is robust to variations in the configuration space and is adaptable. The planner's performance measures will be defined based on the following parameters of a given plan:

1. generates the path with the shortest distance.
2. Uses the least amount of work.[3]
3. Can adapt to different blocking locations along the path.
4. Can adapt to different object dimensions and orientations blocking the path.
5. Can differentiate between static and dynamic obstacles.

1.2.3 Report Outline

The problem addressed in this paper was approached in several steps as can be seen in the planning presented in B. Each step of the process is addressed in a separate chapter creating the outline of this report. Therefore, the global outline of the report is as follows:

Chapter 2 - Preliminaries This chapter contains a literature study to get an insight about the current state of the art and evaluate different motion planning components. Additionally, different navigation stacks are addressed and evaluated to be later compared with the designed stack.

Chapter 3 - Conceptual approaches and Requirements This chapter discusses different conceptual designs to solve the problem defined in section 1.2. These designs were created based

[1] other manipulation techniques will be discussed in 2.5.

[2] A problem is Linear if there exists a sequence of independent objects that could be moved to connect two disjoint components of the free configuration space. This will be explained further in section 2.5

[3] Work is expressed as the $Force * Distance$ needed to move the obstacle.

Note: that other performance measures can be added in the future like, fragility of the object or the distance of the object from the end goal, etc.

on an analysis of the state of the art and the different planners mentioned in chapter 2. Moreover, the chosen design will be discussed based on the performance measures.

Chapter 4 - Implementation This chapter discusses the Implementation process of the chosen algorithmic approach and points out specific details with regards to different functions generated in the algorithm.

Chapter 5 - Experimental Results This chapter contains the experimental setup created to test the implementation discussed in chapter 4 and presents the results obtained which are tested for their validity w.r.t the defined requirements.

Chapter 6 - Future work and Conclusion This chapter points out functions that could be integrated in the algorithm to allow smooth inclusion to the Tech-United software. Finally, the work will be wrapped up and some final remarks will be addressed.

Chapter 2

Literature Study

2.1 Planning spaces

In order to ensure safe motion planning, an important notion needs to be defined, which is the concept of *configuration spaces* and *action spaces*. However, before that, a motion planning strategy needs to be outlined. As presented in the work by (Coenen), a very famous concept adopted by the navigation stacks discussed in subsection 2.4, is the hierarchical motion planning strategy. This strategy subdivides the mobile robot motion planning problem into 3 components:

- **Topological Planning** : A planner that works on the high-level representation of the world which expresses spatial relations (semantic and or geometric information) between world model entities.
- **Global Planning**: A planner that generates a high-level path to the end goal based on a search algorithm, which iterates over a low-level planning representation of the world that expresses the geometric properties of the world.
- **Local planning**: A low-level planner which executes the path generated by the global planner in order to reach the global goal without collision.

Note that the focus of this research falls on the high-level planning aspect of navigation, so the first two components of the strategy will be investigated. However, for completeness, the local planning parameters will also be discussed to give an overall view of the desired navigation stack to be developed in the future.

Now that the planning strategies and representations have been defined, it is possible to interpret the concept of *configuration & work spaces*. Based on the work of (Coenen), a configuration space of a robot describes the set of all possible robot poses in the world representation and that it is always explicitly the case that the *configuration space* of the robot is smaller than the *work space*, since it focuses on the possible center point positions of the base, which allows the avoidance of collision checking when planning. In other words, the *configuration space* of a robot is a collision free area that is used for motion planning representation. This is illustrated in figure 2.1, that represents the difference between both.

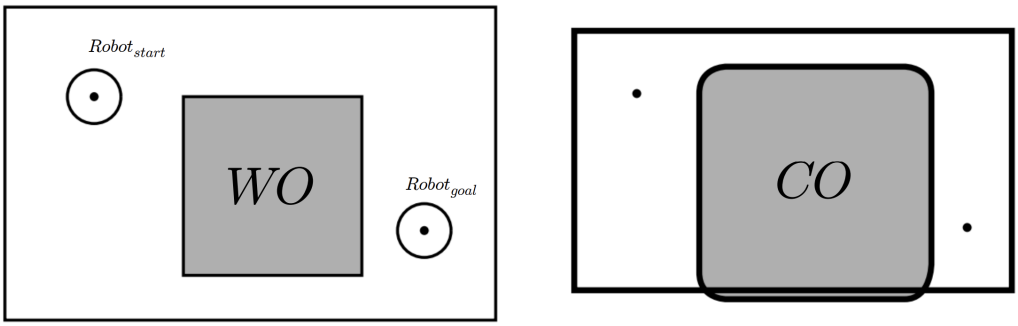


Figure 2.1: Configuration vs work space (Coenen)

2.2 Path planning algorithms

This section illustrates a more in depth description of the path planning problem and its two main components: Topological and Global planning. This will be followed by an explanation of the graph search algorithms used in the implementation.

Note: There are more graph search algorithms, but only 3 will be discussed in depth as they are used in the implementation or mentioned in the report.

2.2.1 Planning Components

It is important to understand that a topological planner and the global planner are high-level planners that search and investigate different planning representations. As previously mentioned(2.1), the topological planner examines the high-level planning representation, while the global planner examines the low-level geometric connectivity graphs of the world representation. With that being said, this shows that, given a goal, the planning performed on the geometric layer of the world representation is done by the global planner. This generated geometric path is then followed by the third component (the local planner) to reach the goal position without collision.

Def 2.2.1. *Connectivity graphs:* a planning representation that describes visible connections between nodes which are positioned on an obstacle edge in the world representation.

2.2.2 Graph Search Algorithms

Coupled with what has been stated above, these high-level planners are search algorithms that iterate through the connectivity graphs generated for the planning representation. There are two main categories of search algorithms: deterministic and randomized[1]. However, only examples from the deterministic category will be addressed. The main difference between both categories is that a deterministic graph search always yields the same result, given that the starting conditions are maintained.

Breadth First Search (BFS)

This graph search algorithm focuses on expanding all the nodes of a search tree at a given depth level before moving on to expanding the nodes in the next depth level. This explanation was presented in the work of (Coenen), and an example can be seen in figure 2.2. Consequently, from a robotics point of view, the search algorithm will start from the initial node and search per depth

[1]As will be seen in section 2.5 some planners use randomized graph searches and specifically RRT (Rapidly exploring Random Trees), but this increases the complexity of the solution. Therefore, it will not be addressed further in this section. However the reader is advised to refer to the work of (Karaman et al. [2011]) for more information

level until the goal is reached. It can be argued that the algorithm is complete and optimal for equal edge costs.

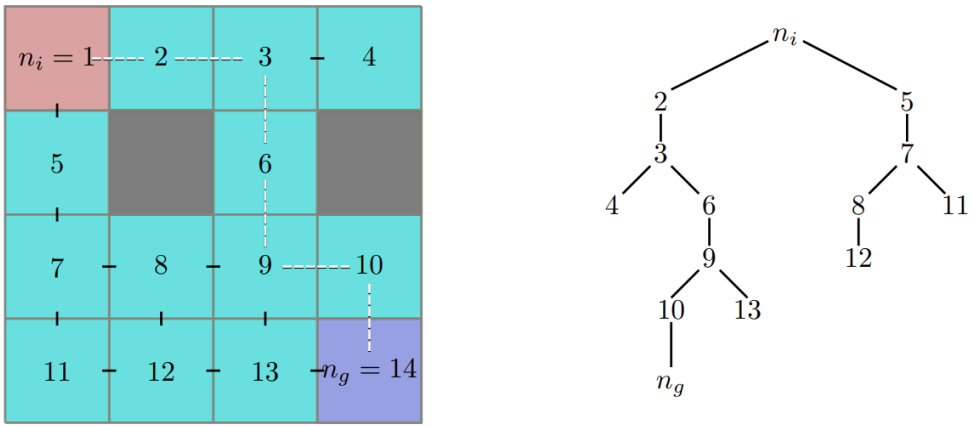


Figure 2.2: BFS node expansion example on a 4-connected grid. (Coenen)

Depth First Search (DFS)

This Search algorithm focuses on expanding each node to the deepest level of the graph or search tree. In this context, the deepest level means until the node has no successors. If the goal node is not found, then the next node with the lowest depth level is investigated until its deepest level. In an attempt to illustrate how the node expansion sequence is generated under a DFS algorithm, the reader is referred to figure 2.3. Furthermore, it can be seen that this algorithm is indeed complete, but non-optimal as it can be argued that minimum cost in not guaranteed. More details on this graph search algorithm is discussed in the work of (Coenen).

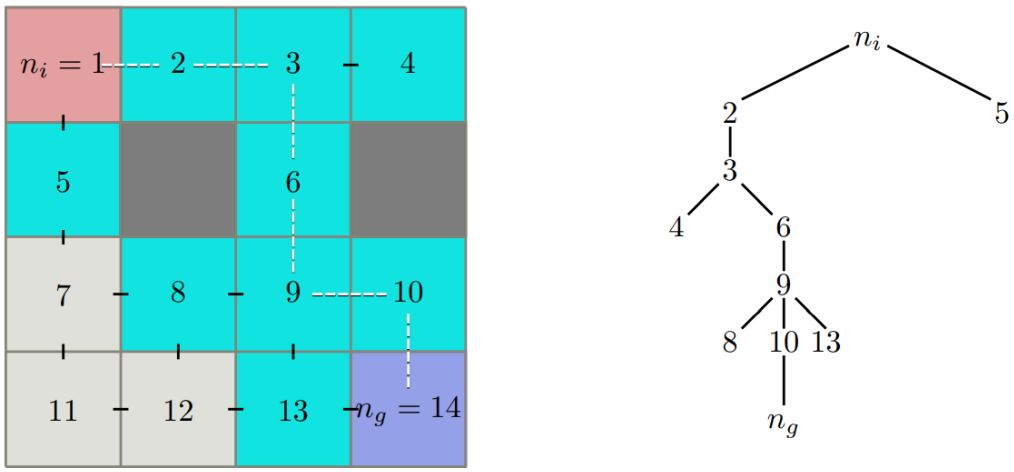


Figure 2.3: DFS node expansion example on a 4-connected grid. Coenen

A* Search

In order to understand how an A* Search algorithm works, it is important to first understand another famous graph search algorithm called **Dijkstra's algorithm**. This is due to the fact that both algorithms are equivalent in their working principle, yet the A* algorithm has an added heuristic. The working principle of those two algorithms is quite similar to that of the **BFS**'s, but here the positive edge costs are taken into account. Therefore, the nodes in a queue are ordered based on their total path costs. This working principle is indeed complete and optimal w.r.t dealing with varying edge costs as demonstrated in the work of (Coenen). Moreover, the addition of the heuristic allows the integration of additional knowledge about the graph. This gives a clear advantage when dealing with single to single node queries, meaning that when comparing A* and Dijkstra's algorithms over a graph search it is found that A* expands less nodes to reach the goal node. This is due to the added heuristic and can be seen in the comparative example illustrated in figure 2.4. An important point to note about the optimality of the A* algorithm, as mentioned in Coenen, is that it can only be considered optimal if the heuristic function $h(n)$ is admissible.

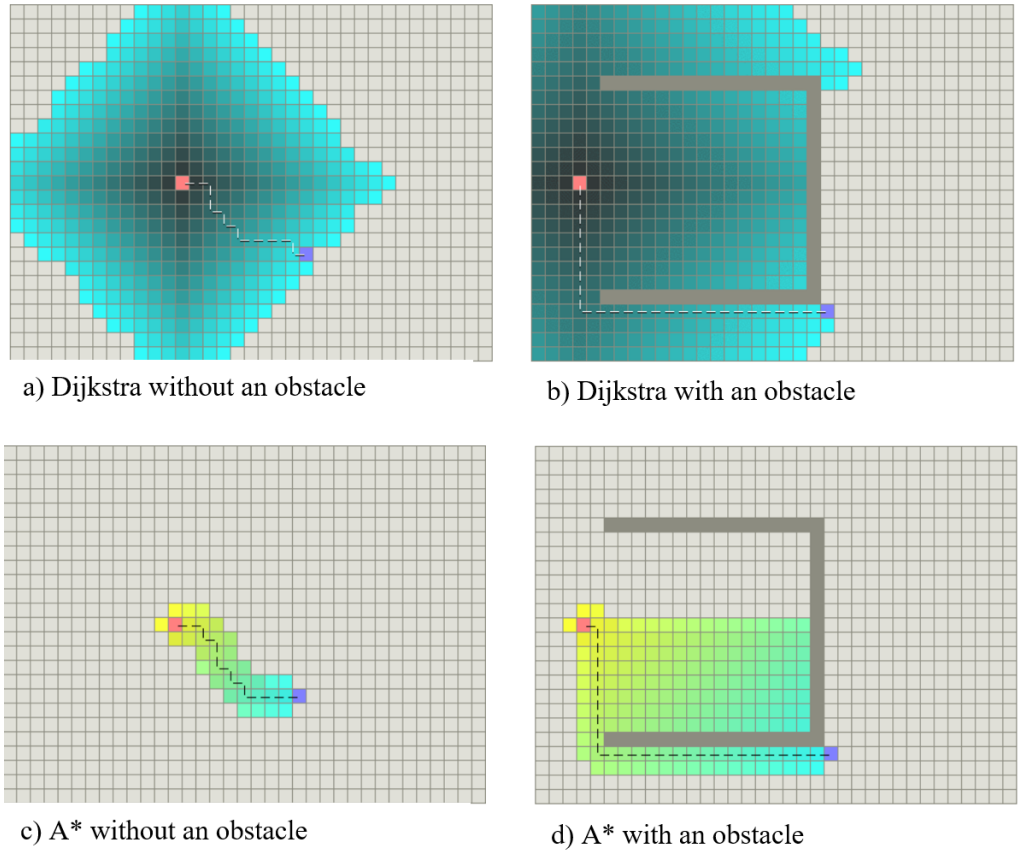


Figure 2.4: Comparative example between A* and Dijkstra's search algorithms. (Coenen)

It is important to note that the current implementation of the global planner in the TU/e Navigation stack as will be discussed in subsection 2.4.2 is based on A* search algorithm. Similarly, the implementation of the new high-level planner seen in section 4 is based on a greedy A* search algorithm as well, yet the heuristic function is far more complex.

2.3 Local Planning

As discussed in section B on planning spaces and the strategic organization of motion planning, the local planners are considered to be low-level planners that are utilized to allow the robot to drive through the geometric path planned by the global planner. In other words, the local planner gives low-level velocity commands to the robot to navigate safely in the environment and avoid obstacles. Therefore, such a low-level planner is not used in the current implementation and would not be discussed in depth. However, for completeness, it will be considered in the reformulated navigation stack that will be discussed in section (6) that emphasizes on future work.

Moreover, it is crucial to mention that the TU/e Navigation stack uses a **Dynamic Window Approach (DWA)** local planner. However, there are many other methods that will not be mentioned since they are out of scope of this project. This local planner works on the dynamic window of the control space $(\dot{x}, \dot{y}, \dot{\theta})$ of the robot, where it performs forward simulations to predict what would happen if a sample velocity is applied for a given time interval $\triangle t$. After that, each trajectory obtained from the forward simulations is evaluated based on a set cost function and the highest score trajectory is picked. More details on this local planner can be found in the work of (Fox et al.).

2.4 Navigation Stacks

In this section, a set of motion planning stacks, that are currently in use for mobile robots, will be discussed. A navigation stack is a visualization of the set of components that execute different navigation services to allow a mobile robot to achieve it's navigational capabilities.

2.4.1 ROS Navigation stack

ROS has its own navigation stack which can be configured on a robot. This relatively simplistic 2D navigation stack takes in odometry data, sensor streams and end goal pose to achieve safe velocity commands for the robot base to follow (Guimarães et al. [2016]). A simplified overview of the ROS navigation stack architecture is depicted in figure 2.5.

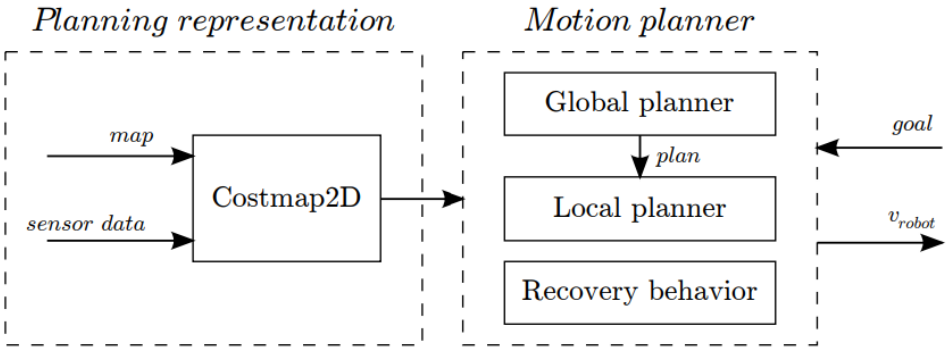


Figure 2.5: ROS Navigation Stack (Appeldoorn)

Firstly, the *costmap2D* is a planning representation of the world model, which is generated in a form of a low-level 2D Occupancy Grid map. Moreover, the *costmap2D* structure receives two inputs: a map from the map server which represents the environment, and sensor data for continuous integration for safety and collision avoidance.

Note that the map obtained from the map server can either be static or dynamic and in 2D or 3D. This is an important thing to take into account, especially when trying to solve the problem of navigating among movable objects, as not only manipulating the objects blocking the path changes the state of the world model, but also, the dynamic objects themselves can be moved by external entities. Consequently, it can be seen that a dynamic map is crucial in this case. However, more on this will be discussed in chapter 4.

Secondly, the motion planner can be considered as a state machine that shifts between the hierarchical motion planning and the recovery behavior. The latter is only executed when the robot is stuck and cannot proceed with the navigation task. This is mostly due to the fact that the local planner is not able to generate velocity commands anymore, either due to the shrinkage of the robot's control space or due to other issues. The hierarchical motion planner contains both a *global and local planners*, where the global planner utilizes a **Dijkstra's graph search algorithm** (section 2.2.2) and the local planner utilizes the **DWA** (section 2.3) to generate navigation commands v_{robot}, thus allowing the robot to drive safely along the path generated by the global planner.

2.4.2 TU/e Navigation stack for Hero Robot

The TU/e uses a different navigation stack architecture than the one discussed above. It not only utilizes a different planning representation, but also, decouples the internal state machine into a general more complex executive layer. This layer provides goals to the motion planner. A simplified overview of the architecture can be visualized in Figure 2.6.

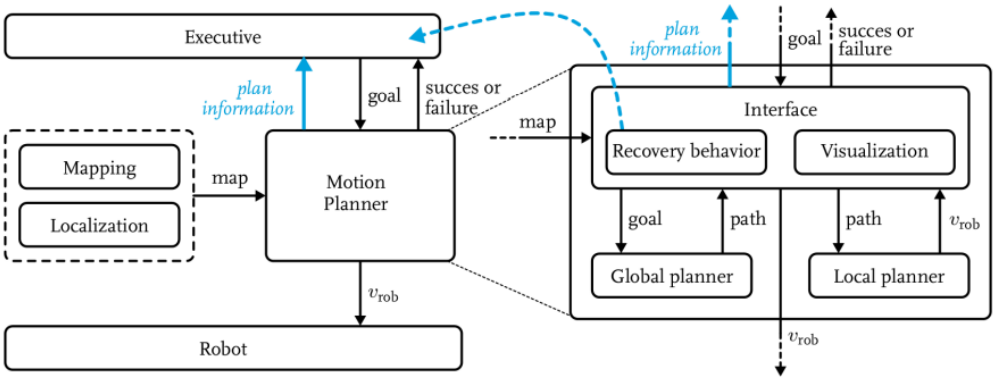


Figure 2.6: TU/e Navigation Stack (Appeldoorn)

The TU/e navigation stack exploits a more complex planning representation, where it applies a down-projection of an Octomap implementation to generate a 2D grid map (Hornung et al. [2013]). This allows the modeling of occupancy probability in every cell, where the maximum occupancy of a column in 3D representation is the occupancy value documented on a cell in the 2D representation.

Similarly, the motion planner still follows a hierarchical strategy that includes two components: *The Global planner*, which utilizes an **A*** **graph search algorithm** to generate path plans towards the end goal and *The Local Planner*, that uses a **DWA** implementation as well.

Moreover, the executive layer acts as an external state machine, which sends goals to the motion planner. It is essential to note that the recovery behavior is executed in this layer as it has more

knowledge with regards to the task being executed at this particular instance. This results in a more informed robot state and, therefore, promotes a better recovery behavior.

2.5 Navigational planners for movable objects

This topic is an extension of the basic motion planning problem mentioned in the work of Coenen, where the Navigation Among Movable Objects (NAMO) problem, as first defined by Stilman and Kuffner [2004], adds a level of complexity where the robot needs to account for disturbances due to movable objects such as doors, chairs, small boxes, etc. Therefore, a navigational planner that addresses this problem extension, must not only plan a path that avoids collisions, reduces travel distance and travel time, but also considers moving obstacles *if necessary*. Close attention must be taken to the reason why "if necessary" was put in italics. The reason behind this is that the planner must adapt to different scenarios where the robot might be put in and a path with risky obstacle manipulation is not always desired. This concept of a risky manipulation is based on the chosen cost function basis, where some planners desire to minimize travel distance others focus more on time and energy optimality. The **NAMO** problem extends the notion of objects in an environment, where it introduces the requirement to differentiate between dynamic and static objects in the robot's world model. Furthermore, this differentiation is essential in terms of the properties of the object representation in the world model. More on this in subsection 2.5.1.

As a result, in this section, a comparative review will be done on different NAMO approaches. The implementation of the planner discussed in chapter 4 is applied based on the approach of WU & Levihn (Wu et al. [2010]) and Stilman and Kuffner [2004]. The reason behind this is that these two approaches proved to be advantageous in aspects that will be discussed below.

2.5.1 World Modeling

NAMO planners require certain object attributes to be implemented for them to work. Attributes, such as, manipulablity, shape and position are considered important aspects. However, the first is a key attribute unique for the NAMO problem. Therefore, in literature NAMO concepts depend on **object-oriented, model-driven world models**, where the manipuability attribute can be added as semantic information (Chen and Hwang [1990], Stilman and Kuffner [2004], Stilman et al. [2007a,b], Nieuwenhuisen et al., Stilman and Kuffner [2008], Wu et al. [2010], Levihn et al. [2013, 2014], Scholz et al. [2016]). Conversely, other approaches have been adopted but their work was discarded as solution considerations to the NAMO problem. For instance, the work of Clingerman et al. [2015] revolved around representing movable objects as high value areas in the grid map representation. However, by doing so the algorithm cannot choose or control alternate obstacle placements. Therefore, a new requirement for NAMO planner considerations was contemplated in the sense that; a NAMO planner must be able to reason about obstacle placement in the world model. Meaning that, obstacles must be presented as separate entities, which was not possible in stand alone occupancy grid representations.[2]

Referring back to the semantic information approach, the basic implementation to the manipuability attribute is as a simple boolean indicating whether the object is movable or not. However, the way this input is given varied in literature. This variation depends on the level of abstraction considered in the framework, where for instance, most simulation based implementations (Chen and Hwang [1990], Stilman and Kuffner [2004], Stilman et al. [2007a], Nieuwenhuisen et al., Berg et al. [2010], Levihn et al. [2013], Moghaddam and Masehian [2016]) directly gave the attribute as input, while for real world implementations Wu et al. [2010], Levihn et al. [2014], Scholz et al. [2016], Meng et al. [2018] either used on board sensors for visual recognition or by manipulation tentative. In real life experiments, it was proven that more object based attributes need to be implemented to achieve successful manipulations. Attributes such as: *center of mass*, *Inertia* and

[2]This, in fact, was one of the difficulties faced during implementation: classifying different block zones in the grid and relate them to the knowledge base as an individual entity.

weight are examples that define the kinematics and physical dynamics of the object.On the other hand, the implementation with these attributes yielded inconsistent results when tested in real life (Meng et al. [2018]). This was due to the deficit in real life robot sensing capabilities that made these attributes hard to determine.

By all means, semantic and spatial knowledge of the robot environment and the objects in it are key aspects to solving the NAMO problem. In fact, only few approaches discussed in literature were able to address the problem with no preceding geometric knowledge of the obstacles in the environment (Wu et al. [2010], Levihn et al. [2014], Meng et al. [2018].

2.5.2 Cost Function definition and optimization

The concept of optimality is rather relative, as it depends on the requirements set for the planning algorithm. This was quite evident in literature as the cost function definition varied extensively, where some approaches used distance, time, number of moved obstacles, probability of success, fragility of object among many others. These cost function specimens were sometimes even combined or used alternatively to achieve higher levels of optimality.

Coupled with what is mentioned above, it can be intuitively realized that cost functions, based solely on object displacement distance, may not be the most propitious choice. This is due to the fact that, in reality, object manipulations result in considerable changes in the robot's energy allocation compared to basic navigation. Therefore, time and/or energy consumption are more suitable alternatives or additions. However, in literature, the use of distance based cost functions can be seen (Chen and Hwang [1990], Berg et al. [2010], Meng et al. [2018]). This adoption is validated on the condition that the objects to be manipulated have negligible weight compared to that of the robot.

In fact, the concept of a complete algorithm is rarely achieved in the NAMO problem and it is indeed rarely seeked. This is because even the simplified planar problem consisting of n-number of dynamic objects was proven to be NP-Hard based on the computational complexity theory(Wilfong [1991]). However, in 2014, Levihn, a leading researcher in the field of NAMO and navigation planning, claimed that his algorithm, presented in Levihn et al. [2014] can achieve local optimality for a simplified problem consisting of a single dynamic object. Moreover, the search to achieve global optimality is still a challenge.

2.5.3 Object Manipulation Techniques

Object manipulation of different objects can be a computationally extensive task, where the robot needs to continuously iterate over its manipulation action space. Therefore, a reduction of the search space was needed for NAMO problems. Many techniques have been developed, but there are mainly three strategies that are adopted.

The first approach is to limit the problem to a level of first order linearity, where only one obstacle may be manipulated at once. This results in no cascading effect on nearby objects. In other words, a path can be opened by manipulating a sequence of independent obstacles that do not hinder each others manipuability. Secondly, an approach adopted initially by Stilman and Kuffner [2004] considered defining a limited set of contact points (CP) that the robot can manipulate the object from. This drastically reduces the problem by facilitating the definition of robot pose for manipulation. Thirdly, an approach that limits the action space to the planar domain, where object manipulations are constrained to translations.

Moreover, the idea of object manipulation is very broad and needs further definition. For instance, the manipulation procedure conducted by the robot; it may be grasping, pushing or even by

manipulation primitives. However, due to the complexity of the latter it hasn't been implemented in real life tests yet.

2.5.4 Planning Algorithms

For starters, the need to distinguish the different aspects of a NAMO solution is essential, as this not only prevents the over fitting of algorithms to certain scenarios, but it also gives a clearer structure to requirements of the general algorithm. Therefore, the aspects of a NAMO planner have been defined as follows:

- Iterating over movable objects.
- Iterating over possible actions.
- Path Computations.

Accordingly, it can be seen that, in most literature, the algorithms usually contains a high-level planner that composes of path planning subroutine and a manipulation planning subroutine. These high-level planners may be based on already existing search algorithms such as: A^* (2.2.2), *BFS* (2.2.2) and many others. However, these high-level planners still need to significantly reduce the computation time, hence prioritizing of which obstacle to move is important and is done quite frequently. This is achieved using a heuristic path planner that ignores the movable obstacles to find the object blocking the desired path (follows the map layering conceptual approach in subsection 3.1.3). Moreover, the concept of obstacle placement after manipulation is more complex and usually decided through forward search[3] algorithms(Stilman and Kuffner [2004], Wu et al. [2010], Stilman et al. [2007a], Levihn et al. [2014], Scholz et al. [2016]).

To conclude, NAMO motion planners require a lot of aspects that account for world representation, cost function definition, manipulation techniques and search algorithms for path and object placement planning. A full overview of the different approaches utilized in practice can be visualized in table C.1 in appendix C.

[3]incremental process of applying motion primitives, like small translations and rotations.

Chapter 3

Conceptual Approaches

3.1 Conceptualization

After the thorough research on navigational components and different planning and search algorithms, a much better understanding of the problem at hand can be formulated. This allows the construction of a couple of conceptual planners and approaches that can be utilized to solve the problem. All the planner concepts discussed in this section are based on the differentiation between static and dynamic objects in the robot's environment.

3.1.1 Conceptual approach 1 - Set Point Definition:

The idea behind this solution is that it defines two set points, one before the dynamic object and one after. The global planner in that case can be augmented to allow generating paths along movable objects and precisely along the predefined set points. Navigation to the first set point can be achieved by a regular navigation planner. However, once the robot reaches set point 1, the object manipulation algorithm needs to override to allow the robot to alter the state of the world model and clear the path. After that, the system can continue navigating to the end goal. Therefore, it can be seen that the motion planning problem, where a robot needs to move from its initial pose to the end goal pose given a geometrical world model with different objects (Latombe [1991]), can be divided into three sub-components executed between two modules the navigational module and the object manipulation module. This not only increases the problem complexity, but also puts safe motion planning at risk. In an attempt to visualize this, one can look at figure 3.1, where the designed solution can be seen.

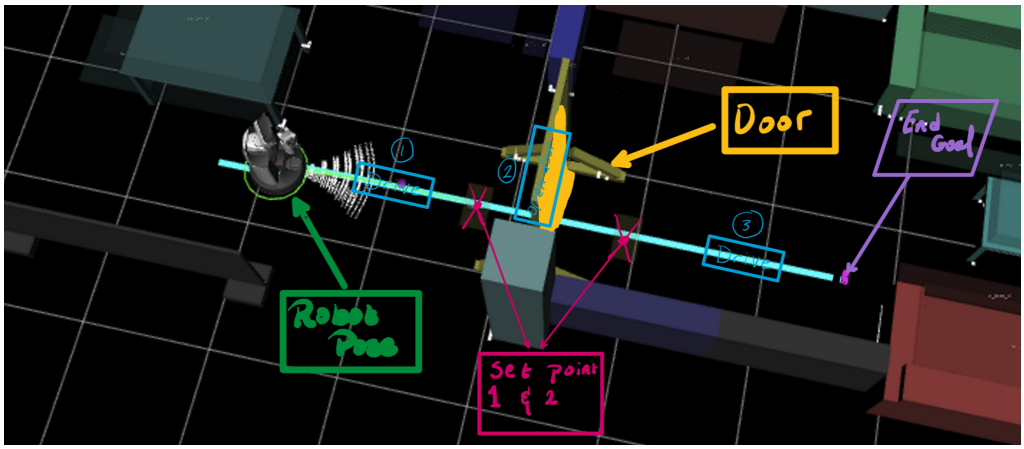


Figure 3.1: Visualization of conceptual design1 with pre-defined set points

Moreover, it can be seen that this concept is a hard-coded approach, where each dynamic object must have two or more pre-defined set points added to it in the world model. This is very unpractical in terms of adaptability and robustness of the algorithm, since it cannot deal with unknown objects and it requires the integration of set points for all possible object orientations.

3.1.2 Conceptual approach 2 - Configuration space Partitioning:

In this Conceptual solution, the robot's free space that is governed by the robot's configuration space is implicitly partitioned into disjoint subsets: $C_A = [C_1, ..., C_M]$ and $C_B = [C_1, ..., C_N]$, where each subset falls along the edge of the dynamic object blocking the connection between the two configuration spaces. A simplified version of this is represented in figure 3.2, where the door is the separating movable object between configuration space A & B. Therefore, if the robot's pose R_1 exists in C_A ($R_1 \in C_A$) and the goal pose S_1 exists in C_B ($S_1 \in C_B$), which means that the robot wants to move from one room/configuration space to the other, then the robot can predict that there is a movable object in between that needs to be manipulated. Therefore, upon detecting the door as a movable object it will attempt to manipulate it, to connect the two sub-spaces.

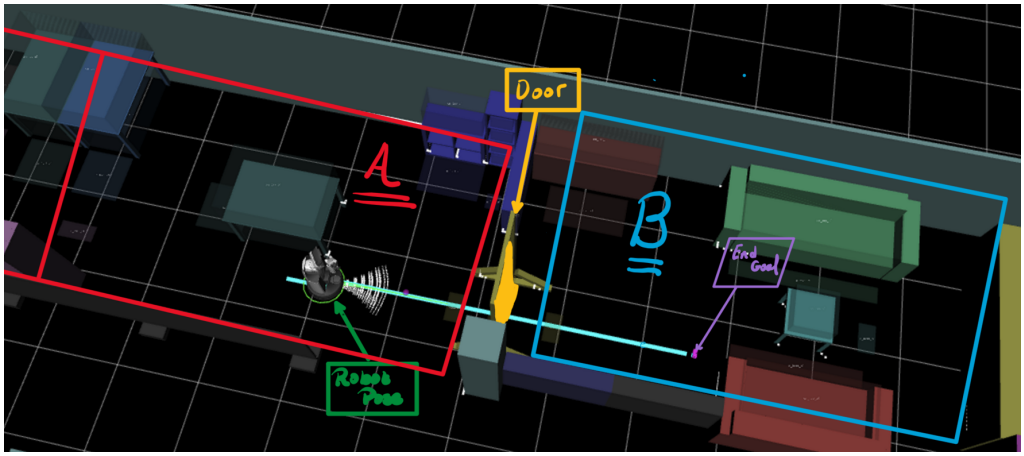


Figure 3.2: Visualization of conceptual design1 with pre-defined set points

Furthermore, the problem drastically increases in complexity when more than one manipulable object is added to the world model. In fact, this concept will be proven to be promising as will be seen in next chapters, yet it is incomplete to be used on its own and still needs to be extended upon to be implemented in practice.

3.1.3 Conceptual approach 3 - Local & Global map separation:

This concept lies upon the idea of map layering, as discussed in the work of Levihn et al. [2014], Wu et al. [2010], where the global and local planners are given different world models/maps. The goal here is that the global planner is only fed a map that contains the static objects and the map boundaries. This allows the global planner to still produce the most desired path based on the algorithm of choice (in the case of Hero, an A* algorithm) without taking into account the dynamic obstacles. The local planner on the other hand is given a more detailed map, which also includes the dynamic objects, where each dynamic object has a proximity tolerance region so that the robot can navigate without object interference up until this region. However, once this region is reached the object manipulation algorithm overrides and alters the state of the world model by removing this proximity region away from the path generated by the global planner. This concept

can be visualized in figure 3.5, where both the maps for the global and local planners are shown to contain different aspects of the world model.

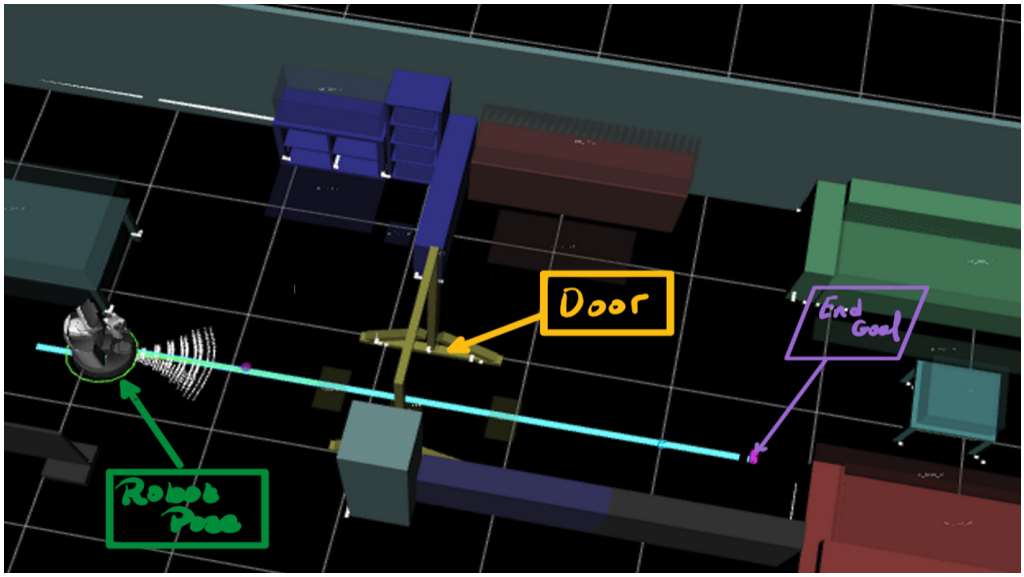


Figure 3.3: a)

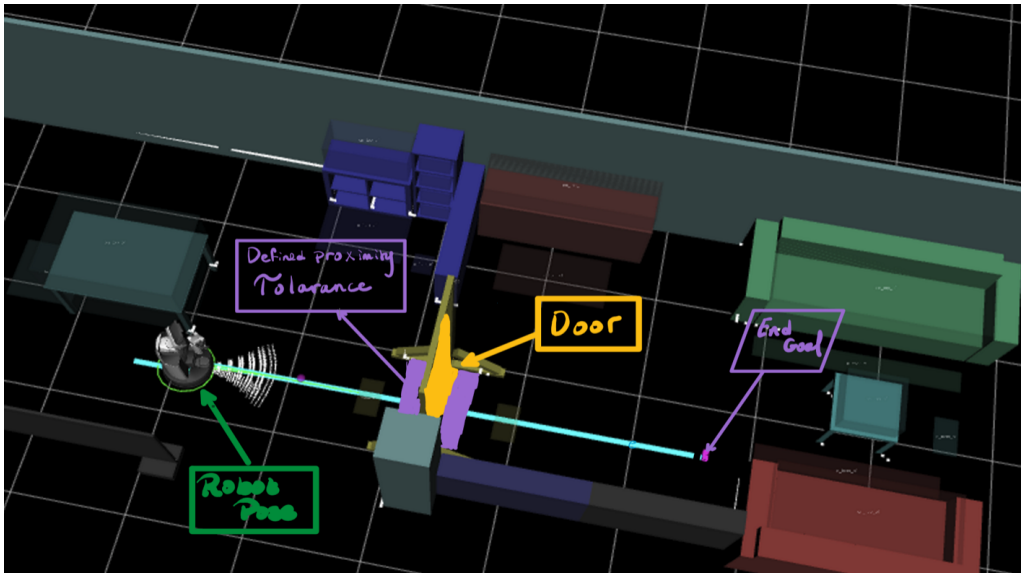


Figure 3.4: b)

Figure 3.5: separation between a) map for global planner and b) map for local planner

3.2 Extension & Reformulation

As seen in each conceptual approach, they all need further extension in order to achieve adaptability and robustness in terms of variations in the world model of the robot. In the work of Stilman et al. [2007b] and Wu et al. [2010], similar approaches may be seen which combine all three conceptual designs to reach a more complete solution. Therefore, in order to express the reformulated approach, one must start by expressing the configuration space of the environment,

where C_{space} defines the configurations of the obstacles in the planner space. This world model includes:

- **M** - a set of Movable objects that the robot can and may manipulate from predefined contact points.
- **S** - a set of Static obstacles that the robot must avoid in the current task.

Note that each object $O_i \in M$ contains a number of predefined contact points, where the robot can manipulate the object from.It is important to mention that those predefined contact points construct the relation between the final chosen approach and the conceptual approach discussed in sub-section3.1.1.

Furthermore, given the robot pose R_1 and the end goal G_1, it is possible to define the robot's free space (C_{free}) into individual components of disjoint sub-spaces$[C_1, ..., C_n]$ such that if robot pose $R_1 \in C_i$, any other configuration $R_2 \in C_i$ can be reached by a non-NAMO navigational planner. In other words, there is no object separating R_1 and R_2. Therefore, the disjoint sub spaces are designed in such a way that they fall on each side of the movable object's edges. This is true since, if the object does not block the path between two disjoint configuration spaces, then there is no need to manipulate it and the robot can simply navigate around it. It can be clearly seen that this part of the approach utilizes the idea of configuration space partitioning seen in 3.1.2. In an attempt to illustrate the benefit of this approach, one can see the robot as a point that moves through C_{space} and, hence, the problem can be solved by a regular search planner.

Consequently, a high-level planner (global planner equivalent) searches for a sequence of free space components(C_{free}) that the robot can connect to reach the end goal. Similarly, it also searches for the corresponding set of obstacles that can be manipulated to allow the robot to navigate through the components without collision. Note that this topological planner still needs a local planner to execute the navigation plans:

1. To the predefined contact points in one component of C_{free}
2. Between two components of C_{free}

As previously noted, this approach combines the conceptual ideas discussed in section 3.1 to achieve a more complete and robust solution of the problem. A more detailed discussion on the implementation process of this theoretically complete planner is exhibited in chapter 4 regarding the implementation. However, note that as mentioned in the problem definition 1.2, only the high level planner will be implemented in this project as solving the whole problem is complex and includes different robotic modules. To add more, the other aspects that need to be integrated to reach a fully functional navigation stack will be addressed in chapter 6, section 6.1.

Chapter 4

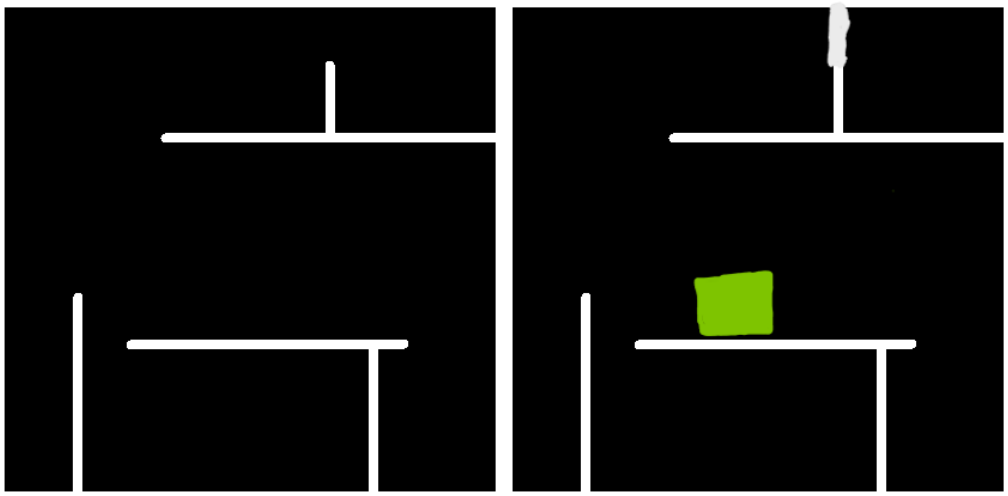
Implementation

In this section, the algorithmic implementation will be discussed. It was chosen to build the implementation on the concepts proposed by Wu and Levihn (Wu et al. [2010], Levihn et al. [2014] and reformulating some of the behavior based on concepts proposed by Stilman (Stilman and Kuffner [2004]). The reason behind choosing this solution method to address the simplified problem definition formulated in section 1.2 is that as seen in table C.1, it is locally optimal. This is true, as long as, the problem is solvable by a single object manipulation in any given direction. Another important point is that it was designed for unknown environments, which covers plan invalidation, making it more suitable for real-world applications. The theoretical bases of the Wu et al. [2010] and Stilman and Kuffner [2004] approaches were discussed in section 3.2 on reformulation and extensions to the conceptual approaches.

The Implementation was done in Python and can be found with execution description in the GitHub repository (1). Furthermore, the algorithm generated currently runs on a low-level grid-map representation and utilizes an A* graph search algorithm to generate plans. Since the algorithm works to solve the simplified problem described in section 1.2, it only considers one dynamic obstacle in the environment, but can withstand n number of static obstacles. The algorithm 1 presents a procedure called TOPO_PLANNER, which takes in the initial position (R_{init}), the end goal (R_{goal}), the grid map without obstacles 4.1a (GM_o) and the grid map with obstacles 4.1b (GM_i) and returns a plan that comprises of manipulation and navigation plans.

In an attempt to illustrate the inner workings of the algorithm, one can firstly look at line (6) of the pseudo code (algorithm 1), where the first call of the A* search is executed. It takes in the initial pose (R_{init}), the goal pose (R_{goal}) and the grid map with obstacle (GM_i). If it finds a path, then the algorithm returns the generated path. However, if the desired path is blocked, so both maps (GM_i) and (GM_o) are not the same. In that case, the algorithm iterates through the grid map using the enumerate function, which takes the occupancy grid map data an outputs an index and a value. Since the grid map is an nxn matrix of occupancy values, the enumerate function needs to be iterated twice, where the first accesses a row and the second accesses a value in that row.

After that, in line 13, each occupancy value in the grid map is tested, whether its 0 (meaning its empty space) or falls out of the dynamic object range (meaning that its a static object or a boundary). Therefore, if the condition in line 13 is satisfied, then the object that is occupying the grid position with coordinates(x,y) is a dynamic object. Consequently, after nxn, it is safe to say that the dynamic object in the environment has been mapped completely and stored in the variables obj_x and obj_y. However, as mentioned in section 2.5.1:footnote 2, this approach is limited to one dynamic object as it cannot express multiple dynamic obstacles as separate entities.



(a) Map without Obstacle (b) Map with Obstacle

Figure 4.1: Examples of maps

Thenceforth, the function PLAN_OBJECT is called. This function takes in the dynamic object coordinates(obj_x and obj_y) and the empty grid map (GM_o) and utilizes it to generate a set point on the blocking dynamic object, depending on its orientation. Since the initial pose (R_{init}) and the defined set point are both elements of the same configuration space subset, a simple navigation path planner can achieve a path to that set point. Therefore, the A* search algorithm is called upon for the second time in order to generate a path plan ($path_{obj}$) towards the generated set point.

Once the $path_{obj}$ is added to the *plan* array, the function MANIP_OBJECT can be called to construct a manipulation plan (obj_manip_plan) and outputs a boolean indicator to test whether the object has been manipulated successfully or not. In a situation where the object is manipulated successfully, the graph search algorithm is called again for the third time to generate the final plan ($plan_{goal}$) between the set point and the end goal and the procedure returns the plan composed of the three sub-plans. Otherwise, the procedure returns null.

Algorithm 1 Algorithm for solving the simplified NAMO problem

procedure TOPO_PLANNER(R_{init}, R_{goal},GM_o,GM_i)
  $empty\_map \leftarrow GM_o$
  $obj_x \leftarrow \emptyset$
  $obj_y \leftarrow \emptyset$
  $plan \leftarrow \emptyset$
  $path \leftarrow A^*(R_{init}, R_{goal}, GM_i)$
  if $path \neq null$ **then**
    return $plan \leftarrow path$
  else
    if $GM_i \neq empty\_map$ **then**
      for $y, row \leftarrow$**Enumerate**($GM_i.data$) **do**
        for $x, val \leftarrow$**Enumerate**(row) **do**
          if $val \neq 0$ AND $threshold.O_{static} < val < threshold.O_{boundary}$ **then**
            obj_x **APPEND** x,
            obj_y **APPEND** y
          end if
        end for
      end for
      $(x_{manip}, y_{manip}) \leftarrow$ PLAN_OBJECT(obj_x,obj_y,GM_o)
      $path_{obj} \leftarrow A^*(R_{init}, (x_{manip}, y_{manip}), GM_i)$
      $plan \leftarrow plan + path_{obj}$
      $(obj_{Manip}, obj\_manip\_plan) \leftarrow$ MANIP_OBJECT(obj_x,obj_y)
      if obj_{Manip} = TRUE **then**
        $plan \leftarrow plan + obj\_manip\_plan$
        $path_{goal} \leftarrow A^*((x_{manip}, y_{manip}), R_{goal}, GM_i)$
        if $path_{goal} \neq null$ **then**
          Return $plan \leftarrow plan + path_{goal}$
        else
          Return null
        end if
      end if
    end if
  end if
end procedure

Chapter 5

Experimental Results

In this section the experimentation results will be discussed to prove the validity of the algorithm proposed in chapter 4. The experiment was designed to test 5 different scenarios that the robot may face in the simplified problem formulated in section 1.2. These scenarios tested the algorithm for robustness and adaptability, by varying the blocking object parameters.

5.1 Experimental setup

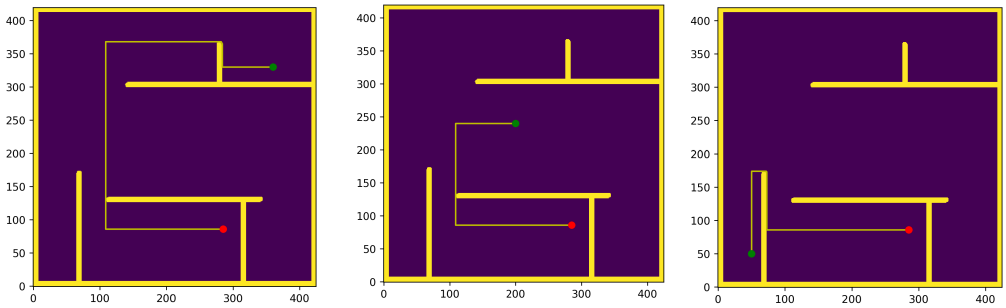
The experimental setup defines a couple of different parameters that need to be taken into account for the algorithm to work appropriately:

- The grid map was inputed as a PNG file and reformulated to a occupancy grid map format using a pre-defined function.
- The occupancy grid maps are 425x425 pixels with a cell size of 1.
- The occupancy threshold value that defines whether a cell is occupied or not is 0.49.
- Boundary walls were given the occupancy value of 0.92 ($threshold.O_{static}$).
- Dynamic objects were given the occupancy range from 0.8 to 0.92 (excluding).
- Static objects were given an occupancy value below 0.8 ($threshold.O_{boundary}$).
- The initial pose ($R_{initial}$) is indicated as a red dot.
- The Goal pose (R_{goal}) is indicated as a green dot.

5.2 Scenarios

5.2.1 Scenario - 1 - No Constraining Object

The first scenario investigates the behavior of the implemented navigation algorithm, when there is no obstacles in the environment. This test scenario was formulated to prove that the extensions applied to the navigation planner does not hinder its initial basic functionality. Therefore, it can be seen if figure 5.1 that the planner generates the shortest path from the initial pose to the goal pose.

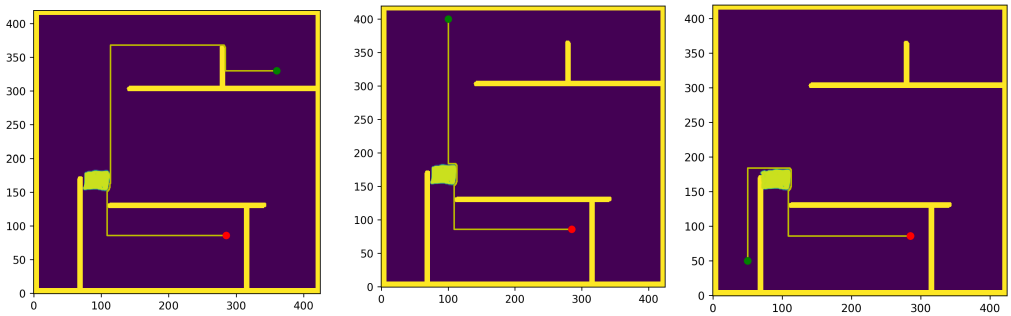


(a) Goal pose 1 (b) Goal pose 2 (c) Goal pose 3

Figure 5.1: Scenario -1- results

5.2.2 Scenario - 2 - Constraining Object not Blocking Path

The second scenario investigates the concepts of necessity and optimality, where the algorithm is given an object that disrupts the shortest path but does not necessarily block it completely. This scenario shows that the algorithm will not manipulate the dynamic object unless absolutely necessary. This notion of necessity was explicitly pointed out in section 2.5, when it comes to costly/risky object manipulations. The results of this experiment can be seen in figure 5.2



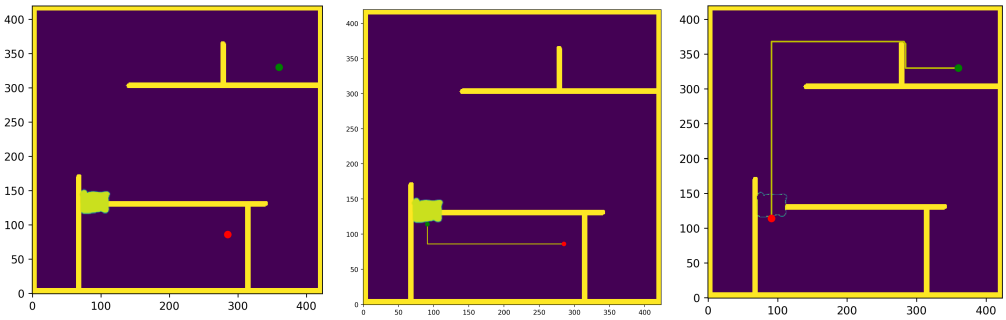
(a) Goal pose 1 (b) Goal pose 2 (c) Goal pose 3

Figure 5.2: Scenario -2- results

5.2.3 Scenario - 3 - Constraining Object Blocking Path

The third scenario is the first investigation of the NAMO problem, where a dynamic obstacle is completely blocking the path to the end goal. The behavior of the planner Can be seen in figure 5.3, where the navigation process is separated into three edges:

1. detection of dynamic object blocking the path. (sub figure 5.3a)
2. Navigating to the object location and manipulating it.(sub figure 5.3b)
3. navigating to the end goal.(sub figure 5.3c)

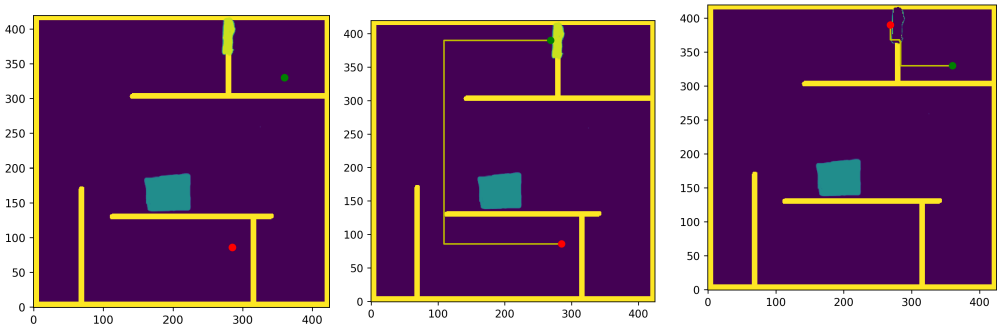


(a) dynamic object detected (b) navigated to dynamic object and attempting manip. (c) Navigation to goal pose

Figure 5.3: Scenario -3- results

5.2.4 Scenario - 4 - Constraining Object at different location

The fourth scenario tests the notions of adaptability and robustness when it comes to dynamic object location and orientation. The algorithm needs to detect the location of the object blocking the path and its orientation to facilitate manipulation. Therefore, the location of the dynamic robot blocking the path has been changed and a static object was added. this can be visualized in figure 5.4.

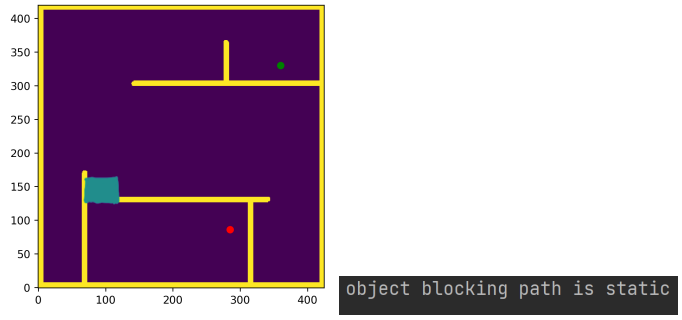


(a) dynamic object detected (b) navigated to dynamic object and attempting manip. (c) Navigation to goal pose

Figure 5.4: Scenario -4- results

5.2.5 Scenario - 5 - Static Constraining Object

The fifth algorithm shows the ability of the algorithm to differentiate between static and dynamic obstacles in the environment. This is evident as, when the path is blocked by a static object and the algorithm cannot find a path anymore it outputs the message visualized in sub fig 5.5b



(a) Static object blocking th path (b) Outputed message

Figure 5.5: Scenario -5- results

5.3 Discussion

Based on the requirements and performance measures defined in section 1.2.2. It can be deduced that the implemented algorithm is an appropriate realization to the given simplified problem, where it travels the shortest distance possible, it is robust and adaptable to different blocking objects and locations, and it uses the least amount of work by not manipulating object when not necessary.

Chapter 6

Future Work and Conclusion

6.1 Future Work

As discussed in chapter 4 on implementation, only the construction of a High-level planner was generated. This High-level planner works on the low-level world representation and outputs a plan on what needs to be done to solve a simplified NAMO problem. However, there is much more to be integrated. As seen in literature (section 2.5). Fully implemented NAMO planners, need to have an object placement planning algorithm, an object manipulation algorithm and a path planning algorithm. Therefore, an extension to this work would be to firstly extend the problem to more than one dynamic object, where an object placement algorithm can be integrated to allow the robot to reason about the placement location that would open up the path. After that the concept of Object manipulations can be integrated. In fact a, complete reformulation of the navigation stack in ROS would be required. A proposed Navigation stack could be visualized in figure 6.1.

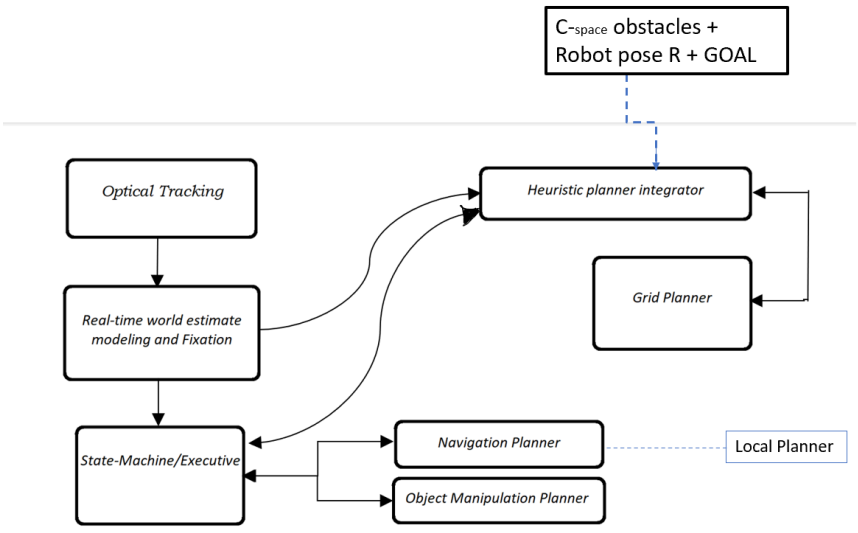


Figure 6.1: Proposed Navigation Stack Architecture

Note that the heuristic planner does not explicitly yield desired walking paths for the robot, the grid planner only guarantees that such a plan is possible due to the existence of free space.

Therefore, a separate navigation planner needs to exist to ensure that this plan can be executed successfully without collision.

6.2 Conclusion

In Conclusion, the NAMO problem is a pretty complex problem when investigated as a whole. However, as seen in literature, when the problem is dissected and broken down to its most basic forms, breakthroughs start to happen. This report and its implementation are an example of that, where when the problem was simplified to a linear planar problem with one dynamic obstacle to be investigated, a solution that combines aspects from the work of two leading researchers in the field(Stilman & Levihn) was implemented. However, this solution is nothing but a scratch on the surface and the state of the art NAMO planners are still far from being globally optimal.

Bibliography

R P W Appeldoorn. Master a volumetric object-oriented world model applied in robot navigation. 10, 11

Jur Van Den Berg, Mike Stilman, James Kuffner, Ming Lin, and Dinesh Manocha. Path planning among movable obstacles: A probabilistically complete approach. volume 57, pages 599–614, 2010. ISBN 9783642003110. doi: 10.1007/978-3-642-00312-7_37.

Pazlg C Chen and Yong K Hwang. Practical path planning among movable obstacles*, 1990.

Christopher Clingerman, Peter J. Wei, and Daniel D. Lee. Dynamic and probabilistic estimation of manipulable obstacles for indoor navigation. volume 2015-December, pages 6121–6128. Institute of Electrical and Electronics Engineers Inc., 12 2015. ISBN 9781479999941. doi: 10.1109/IROS.2015.7354249.

S A M Coenen. Motion planning for mobile robots-a guide.

Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance.

Héctor H González-Baños and Jean-Claude Latombe. Navigation strategies for exploring indoor environments.

Rodrigo Longhi Guimarães, André Schneider de Oliveira, João Alberto Fabro, Thiago Becker, and Vinícius Amilgar Brenner. Ros navigation: Concepts and tutorial. *Studies in Computational Intelligence*, 625:121–160, 2 2016. ISSN 1860949X. doi: 10.1007/978-3-319-26054-9_6.

Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34:189–206, 4 2013. ISSN 09295593. doi: 10.1007/s10514-012-9321-0.

Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt *, 2011.

Jean-Claude Latombe. *Robot Motion Planning*. Springer US, 1991. ISBN 978-0-7923-9206-4. doi: 10.1007/978-1-4615-4022-9. URL http://link.springer.com/10.1007/978-1-4615-4022-9.

Martin Levihn, Jonathan Scholz, and Mike Stilman. Hierarchical decision theoretic planning for navigation among movable obstacles. volume 86, pages 19–35. Springer Verlag, 2013. ISBN 9783642362781. doi: 10.1007/978-3-642-36279-8_2.

Martin Levihn, Mike Stilman, and Henrik Christensen. Locally optimal navigation among movable obstacles in unknown environments, 2014.

Zehui Meng, Hao Sun, Ken B.H. Teo, and Marcelo H. Ang. Active path clearing navigation through environment reconfiguration in presence of movable obstacles. volume 2018-July, pages 156–163. Institute of Electrical and Electronics Engineers Inc., 8 2018. ISBN 9781538618547. doi: 10.1109/AIM.2018.8452226.

Shokraneh K. Moghaddam and Ellips Masehian. Planning robot navigation among movable obstacles (namo) through a recursive approach. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 83:603–634, 9 2016. ISSN 15730409. doi: 10.1007/s10846-016-0344-1.

Dennis Nieuwenhuisen, A Frank Van Der Stappen, and Mark H Overmars. An effective framework for path planning amidst movable obstacles.

Andrzej Pronobis and Patric Jensfelt. Large-scale semantic mapping and reasoning with heterogeneous modalities. In *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*, Saint Paul, MN, USA, May 2012. doi: 10.1109/ICRA.2012.6224637. URL http://www.pronobis.pro/publications/pronobis2012icra.

Benoit Renault, Jacques Saraydaryan, Olivier Simonin, Insa Lyon, and Cpe Lyon. Towards s-namo: Socially-aware navigation among movable obstacles. pages 241–254, 2019. doi: 10.1007/978-3-030-35699-6_19. URL https://hal.archives-ouvertes.fr/hal-02293242.

André M. Santana, Kelson R.T. Aires, Rodrigo M.S. Veras, and Adelardo A.D. Medeiros. An approach for 2d visual occupancy grid map using monocular vision. volume 281, pages 175–191, 12 2011. doi: 10.1016/j.entcs.2011.11.033.

Jonathan Scholz, Nehchal Jindal, Martin Levihn, Charles L. Isbell, and Henrik I. Christensen. Navigation among movable obstacles with learned dynamic constraints. volume 2016-November, pages 3706–3713. Institute of Electrical and Electronics Engineers Inc., 11 2016. ISBN 9781509037629. doi: 10.1109/IROS.2016.7759546.

Mike Stilman and James Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. volume 1, pages 322–341, 2004. ISBN 0780388631. doi: 10.1109/ichr.2004.1442130.

Mike Stilman and James Kuffner. Planning among movable obstacles with artificial constraints. volume 27, pages 1295–1307, 11 2008. doi: 10.1177/0278364908098457.

Mike Stilman, James J Kuffner, Christopher G Atkeson, Matthew T Mason, and Jean-Claude Latombe. Navigation among movable obstacles, 2007a.

Mike Stilman, Koichi Nishiwaki, Satoshi Kagami, and James J. Kuffner. Planning and executing navigation among movable obstacles. volume 21, pages 1617–1634, 10 2007b. doi: 10.1163/156855307782227408.

Sebastian Thrun. Robotic mapping: A survey, 2002.

Jiankun Wang, Weinan Chen, Xiao Xiao, Yangxin Xu, Chenming Li, Xiao Jia, and Max Q.-H. Meng. A survey of the development of biomimetic intelligence and robotics. *Biomimetic Intelligence and Robotics*, 1:100001, 6 2021. ISSN 26673797. doi: 10.1016/j.birob.2021.100001.

Gordon Wilfong. Motion planning in the presence of movable obstacles, 1991.

Hai Ning Wu, Martin Levihn, and Mike Stilman. Navigation among movable obstacles in unknown environments. pages 1433–1438, 2010. ISBN 9781424466757. doi: 10.1109/IROS.2010.5649744.

Appendix A

Examples of Bio-mimetic technologies

The field of biometrics has always focused on the development of novel theories and technologies that are inspired by the biological systems found in nature. This continuous monitoring and investigation of biological systems induces and promotes technological development in many fields. Particularly, in the field of robotics this biomemetic behavior has been noticed quite frequently. In fact the field of biomimetic robotics focuses solely on the development and integration of biomimetic mechanisms, actuators and sensors to improve the performance of robotic systems. The reason behind all this interest, is that biological systems have proved over the years that they can function and adapt in complex environmental settings. Therefore, a lot of examples of biomimetic robots can be seen in practice across different scenarios, such as humanoid robots, quadruped robots,bird-like flying robots and many more. (A.1)

Moreover, the field of robotic biomimicry is not just limited to physical functions, it is extended to the software aspect using biomimetic intelligence. This allows the development of series of efficient algorithms and breakthroughs such as Neural Networks that mimic the neurons in a human brain, and genetic algorithms that were inspired by the human genetic code (Wang et al. [2021]). These algorithms further enhance the performance of robotics deployed in complex environments. Despite, the advancement of these intelligent algorithms is quite remarkable the strive to achieve artificial general intelligence and acquiring continuous self -learning is still a dream under development.

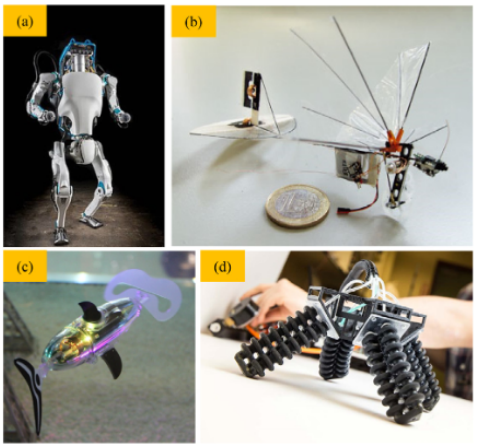


Figure A.1: (a)Atlas an intelligent humanoid robot. (b) DelFly micro a flapping wing robot. (c) JESSIKO an under water fin propelled robot. (d) four legged soft turtle robot (Wang et al. [2021])

Appendix B

Planning and Organization

This project was started on 17th of Jan 2022 and continued till the 4th of July 2022. The planning of the project followed the scrum methodology for project management. In fact, this approach was chosen as it proved to be a good approach for complex software development projects, as it follows an iterative and incremental process that is adaptable and fast. Moreover, this framework is designed to allow the breaking down of complex projects into small manageable tasks, which can be completed in short time iterations throughout the project's time cycle. These short time iterations are called sprints and are designed to be 2 weeks in length. In the beginning of the project the problem description and project deliverables were clearly defined, allowing the generation of a project overview. This not only resulted in better comprehension of the project requirements, but also, prompted the formulation of manageable tasks.

Therefore, in the beginning of the project the focuses was mainly on constructing a better understanding of the problem at hand by conducting extensive research on the main aspects of the project. This was called the preparation period, where the following was achieved:

1. Research on main principles of Navigation
2. Research on the concepts of world modeling in a navigational context.
3. Reformulating and defining the problem description.
4. Investigating the TU/e navigation stack and the ROS navigation stack.
5. Refreshing the knowledge on ROS
6. Constructing the Planning for implementation which can be seen in Figure B.1.
7. Started research on navigation among movable objects.

This progress can be seen in the first BEP progress presentation on the github repository link[1]

Moreover, the progress continued by generating a number of conceptual approaches for navigation among movable objects. These concepts were compared to ideas in literature and investigated for their completeness, finally an approach was chosen and steps for software implementation were taken. This phase was called the theorizing phase, where the following was achieved:

1. Research on navigation among movable objects.
2. Constructing conceptual algorithmic approaches
3. Constructing a new theorized navigation stack architecture.

[1] *https://github.com/ismailelmasry/BEP_ismail*

4. Adapting conceptual approaches to ideas found in literature
5. Formulating pseudo-code based on Stilman's approach in 2004.
6. Investigating the switch to Levihn's approach in 2010 for benefits discussed in subsection 2.5.

This progress can be seen in the second BEP progress presentation on the github repository.

Last but not least, the implementation was started, where a lot of issues were faced due to integration errors, missing parts of code that needed, such as manipulating objects or object detection. During this implementation phase the following was achieved:

1. Investigating and understanding the topological planner created previously by TechUnited @Home.
2. Implementation of Levihn's approach.
3. Constructing 2D Grid maps from PNG pictures.
4. Formulated an A search algorithm.
5. Creation of example Gazebo worlds.
6. Attempted to integrate and test the implemented code.

This progress can be seen in the github repository.

In conclusion, the project followed a very structured approach where the tasks were divided among the sprints and iterated throughout the whole project duration.

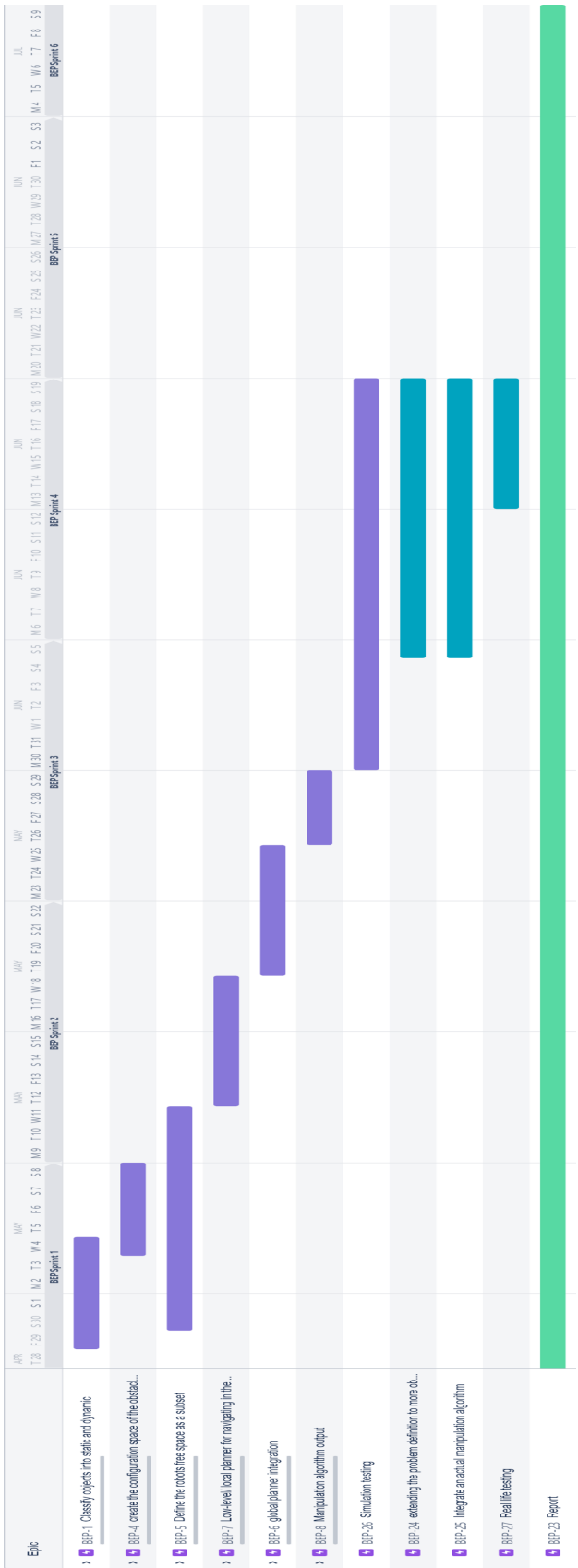


Figure B.1: implementation planning

Appendix C

Comparative Table of NAMO Planning Algorithms

In this appendix, a comparative table is created based on the work of Renault et al. [2019], where the different approaches found in ten research papers was analysed for 9 different parameters:

- How object movability attribute is given.
- Completeness level of the approach.
- Optimality level of the approach.
- Cost function definition.
- how is the configuration space (C_{space}) is defined.
- what algorithm is used to conduct the task planning.
- what algorithm is used to conduct the motion planning.
- what algorithm is used to conduct the object allocation planning.
- The manipulation technique used to move the object.

The Table can be visualized below.

Reference	Movability	Completeness	Optimality	Cost	$\mathcal{C}_{space}$	Task planning Alg.	Motion Planning Alg.	Allocation planning Alg.	Manip. technique
Chen and Hwang [1990]	Given	--	--	D	Disc.	Dij.+GD	N/A	N/A	Prim.
Stilman and Kuffner [2004]	Given	RC	--	E+NMO	Disc.	DFS	A*	BFS	Prim.
Stilman et al. [2007b]	Given	RC	--	E+NMO	Disc.	DFS	A*	BFS	Prim.
Nieuwenhuisen et al.	Given	PC	--	D+PS	Cont.	Custom	RRT	RRT	Grasp
Stilman and Kuffner [2008]	Given	--	--	E+NMO	Disc.	DFS	A*	BFS	Grasp
Berg et al. [2010]	Given	PC	--	(D)	Disc.	Custom	N/A	N/A	Grasp
Kakiuchi [2010]	Manip.	--	--	(D+NMO)	Cont.	Custom	RRT	N/A	Push
Wu et al. [2010]	Manip.	--	--	(D ‖ T ‖ E)	Disc.	Custom	A*	DFS	Push
Levihn et al. [2014]	Manip.	--	LO	(D ‖ T ‖ E)	Disc.	Custom	D* Lite	DFS	Grasp
Levihn et al. [2013]	Given	--	--	PS	Disc.	MDP+MCTS	N/A	N/A	Prim.
Levihn et al. [2013b]	Given	--	--	T+E	Cont.	MDP+MCTS	PRM	RRT	Prim.
Moghaddam and Masehian [2016]	Given	CO	--	E	Cont.	DFS	Dij.+VG	Dij.+VG	Grasp
Scholz et al. [2018]	Recog.	--	--	T+E	Cont.	MDP+MCTS	PRM	RRT	Prim.
Meng et al. [2018]	Recog.	--	--	D	Cont. MP	Custom	RRT	RRT	Grasp ‖ Push

Legend: () = Not given but likely; '+' = Combination of; '‖' = Alternative to; Manip. = Found through manipulation; Recog. = Found through visual recognition; '--' = Depending on columns, either Not Optimal or Not Complete; CO = Complete; RC = Resolution-Complete; PC = Probabilistically Complete; LO = Locally Optimal; D = Distance; E = Energy; T = Time; NMO = Number of Moved Obstacles; PS = Probability of Success; Disc. = Discrete; Cont. = Continuous; Dij. = Dijkstra; GD= Generalized Distance; VG = Visibility Graph; NG = Not Given; N/A = Not Applicable; Prim. = Motion Primitive

Figure C.1: Comparative table of NAMO planning algorithms. (Renault et al. [2019])