DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Developing a System for Robust Planning using Linear Temporal Logic

NADINE DROLLINGER

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Developing a System for Robust Planning using Linear Temporal Logic

Master Thesis

Nadine Drollinger
nadined@kth.se
Systems,Control and Robotics
Automatic Control Department
Systems, Control and Robotics, KTH

Supervisor: Sofie Andersson
Examiner: Dimos Dimarogonas

Abstract

Human robot-collaborative search missions have gotten more and more attention in recent years. Especially in scenarios where the robot first scouts the scene before sending in human agents. This saves time and avoids unnecessary risks for the human agents. One possible configuration of such a rescue team is, a human operator instructing a unmanned aerial vehicle (UAV) via speech-commands how to traverse through an environment to investigate areas of interest. A first step to address this problem is presented in this master thesis by developing a framework for mapping temporal logic instructions to physical motion of a UAV.

The fact that natural language has a strong resemblance to the logic formalism of Linear-Temporal Logic (LTL) is exploited. Constraints expressed as an LTL-formula are imposed on a provided labeled map of the environment. An LTL-to-cost-map converter including a standard input-skeleton is developed. Respective cost maps are obtained and a satisfaction-measure of fulfilling these constraints is presented. The input-skeleton and the map-converter are then combined with a cost-map-based path planning algorithm in order to obtain solution sets. A clarification request is created such that the operator can choose which solution set should be executed. The proposed framework is successively validated, first by MATLAB-experiments to ensure the validity of the cost-map-creation followed by simulation experiments in ROS incorporating the entire framework. Finally, a real-world experiment is performed at the SML to validate the proposed framework.

Sammanfattning

Human-robot-collaborative search missions har fått ökad uppmärksamhet de senaste åren. Framför allt i scenarion där roboten först utforskar miljön innan mänskliga agenter inträder, vilket sparar både tid och undviker onödiga risker för de mänskliga agenterna. En möjlig konfiguration för ett räddningsteam är då en mänsklig operator instruerar en UAV via tal-teknologi hur den ska navigera genom en miljö för att undersöka områden av intresse. Den här uppsatsen ämnar att undersöka detta problem genom att utveckla ett ramverk för kopplingen mellan temporal-logiska instruktionen till fysiska rörelsemönster för en UAV. Faktumet att naturligt språk har en stark samhörighet mellan logisk formalist av Linjär Temporal Logik (LTL) är exploaterat. Constraints uttryckta som en LTL-formula tillförs på en tillhandahållen klassificerad karta av omgivningen. I denna rapport presenteras en LTL-cost-to-map-översättning som inkluderar ett standardiserat input-skelett. Vidare presenteras kostnadskartor med tillfredställelse-mått som uppfyller dessa krav. Input-skelettet och map-konverteraren kombineras sedan med en cost-map-baserad path planning algoritm för att erhålla lösningsset. En klarifikationsförfrågan är skapad så att operatorn kan välja vilket lösningsset som ska exekveras. Det föreslagna ramverket är successivt validerat, inledningsvis med MATLAB-experiment för valideringen av cost-map-kreationen följt av simuleringsexperiment i ROS som inkorporerar hela ramverket. Slutligen utförs ett real-time experiment vid SML för att validera det föreslagna ramverket.

Acknowledgement

To my grand-parents and my mum. Thank you for a life-time full of support and encouragement.

Nadine Drollinger
May 2018
Stockholm

Contents

1 Introduction **1**

1.1 Related work . . . 2

1.2 Report outline . . . 3

2 Preliminaries **3**

2.1 Natural Language . . . 3

2.2 Linear Temporal Logic . . . 4

2.3 Workspace Representations . . . 5

2.3.1 Labeled Maps . . . 5

2.3.2 Cost Maps . . . 5

2.4 Path Planning . . . 6

2.4.1 Search Strategies . . . 7

2.4.2 Breadth-First-Search . . . 8

2.4.3 Depth-First Search . . . 8

2.4.4 A^*-Search Algorithm . . . 9

2.5 Motivating Examples . . . 10

3 Problem Definition **11**

3.1 Assumptions . . . 11

4 Solution **11**

4.1 The Input-Skeleton . . . 12

4.2 The LTL-Map-Converter . . . 15

4.2.1 The Always-Not-Array . . . 15

4.2.2 The Weak-Eventually-Array . . . 16

4.2.3 The Eventually-Always-Array . . . 17

4.2.4 The Safe-Mode . . . 18

4.3 Path Planning using A^* . . . 18

4.4 The Clarification Request . . . 19

4.5 UAV Path Execution . . . 19

5 Experiments **20**

5.1 Map-Conversion- Experiments . . . 20

5.1.1 Experiment 1 . . . 20

5.1.2 Experiment 2 . . . 22

5.1.3 Experiment 3 . . . 23

5.1.4 Evaluation of Map-Conversion-Experiments . . . 25

5.2 Set-up and Realization for Framework- Experiments in ROS . . . 26

5.2.1 Overview of Framework . . . 26

5.2.2 The World Representation . . . 26

5.2.3 The Input Node . . . 27

5.2.4 Path Planning using A^* . . . 28

5.2.5 Adjustments for the Path . . . 30

5.2.6 The Clarification Request . . . 32

5.3 Framework Experiment in Simulation . . . 33
5.4 Framework- Experiment in SML . . . 35
5.4.1 Setup . . . 35
5.4.2 The Task . . . 36

6 Discussion and Conclusion **37**

7 Future Work **38**

References **39**

List of Figures

1 Intuitive semantics of temporal modalities, excerpt from [1] . . . 4
2 Simple Example of a Color Map . . . 6
3 Color Map and Corresponding Cost Map . . . 6
4 Nodes for Graph Search . . . 7
5 BFS-search tree ,excerpt from [2] . . . 8
6 DFS-Search Graph, excerpt from [2] . . . 8
7 A^*-Pseudo-code, excerpt from [2] . . . 9
8 Solution Overview . . . 12
9 Preference-diagram . . . 13
10 I/O-Scheme for LTL-Map Converter . . . 15
11 Color Map of an Environment . . . 16
12 Cost map for $\varphi = \square\neg orange$ . . . 16
13 Cost map for $\varphi = \square\neg yellow$ . . . 16
14 Color Map of an Environment . . . 17
15 Resulting cost-map for formula 18 . . . 17
16 Safe-mode on for formula 18 . . . 18
17 Safe-mode off for formula 18 . . . 18
18 Principle of the Clarification Request . . . 19
19 Principle of the Path Execution for the UAV . . . 20
20 Color Map of Environment 1 . . . 21
21 Resulting Cost Maps . . . 21
22 Resulting Maps for $\varphi_{\vee_1}$ with and without Safe-Mode . . . 22
23 Resulting Maps for $\varphi_{\vee_2}$ with and without Safe-Mode . . . 23
24 Color Map of Environment 3 . . . 23
25 Resulting Cost Maps for $\Gamma_{\wedge_1}$ and $\Gamma_{\wedge_2}$ with Active Safe-Mode . . . 24
26 Resulting Cost Maps for $\Gamma_{\wedge_1}$ and $\Gamma_{\wedge_2}$ with Not Active Safe-Mode . . . 25
27 ROS-Structure of the Framework . . . 26
28 By Areas Labeled Map of the Environment . . . 27
29 Labeled-by-color map displayed in Rviz . . . 27
30 Colored Grid Map with U-Shaped and Disjunct Region of Interest . . . 28
31 Resulting Path for the A^* -algorithm for the LTL-formula 29 . . . 29
32 Resulting Path for the A^*-algorithm for the LTL-formula 30 . . . 29

33 Cost Map and Corresponding Enlarged Cost Map 30
34 Line-of-Sight Path Smoothing, excerpt from [3] 31
35 Path and by Line-Of-Sight-Smoothed Path in Enlarged Map 31
36 Clarification Request and Confirmation for the case of three maps and paths 32
37 Labeled Map of the Environment 33
38 Top-down view of the Gazebo world with UAV at initial pose (0.5,0.5) 33
39 Path and Smoothed Path for formula 32 34
40 Path and Smoothed Path for formula 33 34
41 Path and Smoothed Path for formula 34 34
42 Issued Clarification Request 34
43 Chosen Map and Path after Clarification Request 35
44 Gazebo Simulation of path execution with UAV 35
45 Labeled Map and Crazyflie 2.0 36
46 Illustration of the three obtained paths and picture of path execution in the SML . . . 36

List of Tables

1 Look-up Table: Labeled by color Map to Cost Map 6
2 LTL representations for NL 10
3 LTL representations for NL 10

1 Introduction

Autonomous systems such as mobile robots incorporated into our daily life are no longer a far-fetched idea for the future. Taking advantage of these systems to assist or support humans in various tasks has received a lot of attention in recent years. Autonomous transportation systems are already in place, be it as self-driving cars, trucks or buses. For the health care sector, domestic robots are of interest as they can assist caretakers or elderly in many ways. Service robots are used for house hold tasks such as cleaning or assisting [4], [5]. Others are designed to keep humans company or interact with them [6].

In contrast to these comfort applications of robots, stands the increasing interest in taking advantage of mobile robots in search- and rescue missions.

Aerial, terrestrial and maritime robotic systems are of great assistance in disaster relief, search and rescue and salvage operations. There are various scenarios where the assistance of a mobile robot is an advantage as these assisting systems allow it to reduce risks for human operators and bear new potentials. If people go missing in remote areas, the search and rescue missions becomes difficult and time consuming, in a situation where time is critical to the health of the person involved. They are deployed quickly and can either be used to guide rescuers or be guided by users in order to collect data, deliver essential supplies or provide communication services.

At KTH such a Human-Robot Collaborative Search Mission project is developed based on the scenario, that a wildfire has broken out in a forest. A collaborative Search Missions is issued to find possibly endangered people in the forest. The search team consist of a unmanned aerial vehicle (UAV) that is equipped with a camera system and a human operator. The operator instructs the UAV via speech commands to investigate areas of interest to find possibly endangered people. The areas are displayed in form of colored sectors on a hand-held device. If the operator instructs the UAV to find cars, and the UAV finds multiple cars, a clarification request from the UAV to the operator is issued to determine which car it should traverse to. To make the project traceable it is split into multiple parts.

The first step towards translating natural language or speech commands into physical motion of a robot is presented within this work by mapping Linear Temporal Logic to physical motion, exploiting the resemblance to natural language. Hence this report focuses on Linear temporal logic, path planning, human-robot-interaction and the execution of motion with a real UAV. However, this work does not focus on speech-recognition but on providing a framework for the execution of the task, starting from LTL-constraints imposed on a map of the environment. The capabilities of the framework developed in this thesis are demonstrated by simulation experiments and real-world experiments performed at the Smart-Mobility-Lab (SML) at KTH.

1.1 Related work

To get a broader sense on already published work, an excerpt of related topics is presented. Among the topics deploying Linear Temporal Logic are motion and action planning through an environment of a robot. These environments, further referred to as workspace can be known, partially known and unknown. In [7] the combination of Temporal Logic based motion and action planning is addressed and an hierarchical approach to formal methods-based robot mission and motion planning under infeasible specifications is presented. The problem of dealing with the infeasibility of desired goals due to environmental constraints and action failures is addressed. This approach tries to find a plan to meet the specifications as closely as possible if actions fail by introducing a measure for the level of satisfaction regarding the goals. The optimal plan is then synthesized by employing the weighted automata-based approach.

In [8] Guo et al. present a generic framework for real time motion planning based on model checking and revision. For more details on model checking see [1]. Based on initially available information about the robot and its workspace a preliminary plan is created, by first modifying a nested-DFS algorithm to search for an accepting path. Then three types of real-time information are classified that might be obtained during a real-time execution. A criterion is introduced to verify if the current path remains valid or has to be revised. That is, while gathering real-time information the plan is verified and revised making this framework especially useful for operating in partially known workspaces and workspaces with uncertainties due to the iterative adjustments.

In [9] the authors analyze single- and multi-agent systems under local LTL-tasks that are infeasible. They describe how to synthesize the motion plan that maximally satisfies the infeasible task and how it could be relaxed. The relative weighting between the implementation cost of a motion plan and its distance to the original specification is evaluated. For multi-agent systems, a dependency relation and relative priorities are incorporated when the tasks are assigned independently to each agent.

Another interesting approach towards optimal path planning under LTL-constrains is presented in [10]. Based on high-level specifications formulated as LTL-formulas, this paper presents a method for automatic planning robust optimal paths for a group of robots. The motion of each robot is represented by a weighted transition system and the mission is formulated as an LTL-formula over a set of propositions satisfied by the regions of the workspace. An optimization proposition is incorporated additionally which must be satisfied repeatedly to ensure the optimality of the path.

[11] presents an approach that ensures the satisfaction of LTL-constraints by designing closed-loop hybrid controllers. That is, it ensures that the resulting continuous trajectories satisfy the LTL-specifications even in the presence of an adversary who repositions the robot within the workspace. This work unites high-level planning with low level control such that an interface between discrete path planning and continuous motion planning is created. Another contribution of this work relies in the composition of the mapping, that is mapping from LTL to physical motion. This is a first step towards the mapping from natural language to physical motion. It is achieved by translating the LTL-specifications into a realization of a so called Büchi-automaton to ensure consistency between a discrete plan and its continuous execution.

A method for automatically generating robot trajectories that satisfy high level- specifications is presented in [12]. Smith et al. present a method for planning the optimal motion of a robot given constraints imposed via LTL. The motion of the robot in the environment is modeled as a general transition system, enhanced with weights. To ensure optimality a single optimization proposition

must be visited repeatedly such that a valid trajectory for the robot can be obtained. This is done by minimizing the maximum time between satisfying instances of the optimization proposition.

In contrast to the previous introduced papers that either focus on obtaining controllers satisfying the constraints or employ automata theory and model checking, the framework presented in here focuses on the planning problem which is subject to the constraints formulated in LTL. The framework presented in this report employs LTL-constraints that are imposed on the environment by a user in order to create respective cost-maps. To evaluate the satisfaction of the constraints a satisfaction measure for the constraints is introduced. According to the constraints respective cost-maps are generated the path planning problem is solved. The results are sets of cost-maps and corresponding paths from which the user can then choose.

1.2 Report outline

A brief introduction into the necessary background knowledge is provided in chapter 2 concluding with some motivating examples to provide the reader with an intuitive understanding. In chapter 3 the problem definition is given and the solution approach is presented in chapter 4. The proof-of-concept for the map-converter is concluded to be successful and evaluated by the MATLAB-experiments described in the first part of chapter 5. The complete framework is then implemented in ROS and the Gazebo-simulation environment followed by the implementation as real world experiment in the SML. The complete implementation in the second half of chapter 5 provides an insight into the framework and its limitations. Finally the results of these experiments are then summarized, discussed and evaluated in chapter 6. Suggestions for possible future work that could build upon this master thesis are provided in chapter 7.

2 Preliminaries

In this chapter the reader can acquire background knowledge to enhance the understanding for the proposed solution. A brief overview for each topic will be given and concluded why this method is used. Readers familiar with the topics might skip this part.

2.1 Natural Language

Natural Language Processing (NLP) refers to the concept of making machines understand natural language and is a much researched field [13]. However Natural Language (NL) itself refers to the way humans communicate with each other and is highly expressive and highly ambiguous. In contrast to Natural Language, formal languages on the other hand are of limited expressive power but admit automated checking [14]. Natural language constantly evolves and therefore language models are considered to be approximations. Interested readers may find more on this topic from language models, grammar, semantics, models up to smoothing techniques in [15], [13] and [14].

In the context of the Human-Robot collaborative search and rescue mission-project which this Master thesis is part of, the final goal is to translate a command into an LTL-formula. Therefore the common ground of LTL and NL on which the framework presented here relies on are the signal words occurring in Temporal Logic such as *not, always ,eventually , and, or, etc.*

2.2 Linear Temporal Logic

In this section a short introduction to Linear Temporal Logic is provided. Readers not familiar with this formalism may read up in [1] and [16]. Linear Temporal Logic further referred to as LTL is a logic formalism, which allows it to specify linear time properties. Temporal logic assumes that constraints hold at particular times and that those times (which may be points or intervals) are ordered. It allows expressing constraints in a logic way and it is very close to natural language [1], [16]. Therefore constraints formulated in *Natural Language* and indicated by the respective signal words of LTL can be directly translated into an LTL-formula. A brief overview of the later employed operators and their essential properties for this report will be given. For further reading see [1]

As the overall project where this master thesis is a part of, wants to employ speech control and commands (formulated in natural language) to communicate with a UAV, the resemblance of LTL to natural language provides an advantage. Hence using LTL to express specifications is suitable in this thesis given the requirements.

An LTL-formula consists of boolean and temporal modifiers as well a as set of atomic propositions AP.

$$\varphi ::= true | \; a \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \; \bigcirc \varphi \mid \varphi_1 U \varphi_2 \tag{1}$$

where a $\in$ AP, $\bigcirc$ denotes the *next* operator, and U denotes the *until* operator.
From these basic operators two more essential operators can be derived as can be seen in [1]. That is the $\Box$ (*always*) operator and $\Diamond$ (*eventually*) operator, which have an intuitive meaning. $\Box\varphi$ holds if φ is always true. $\Diamond\varphi$ ensures that φ will be true eventually in the future. An overview over the essential operators can be seen in figure 1.

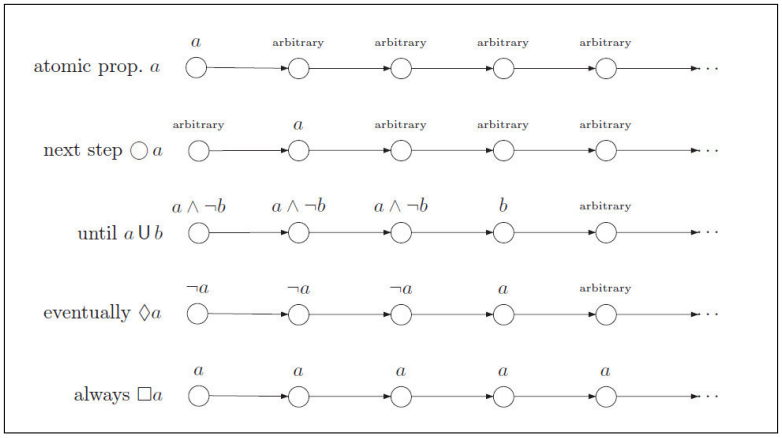


Figure 1: Intuitive semantics of temporal modalities, excerpt from [1]

However to formulate commands that can be executed given a map and a path-finding task, we employ the following operators. Conjunction is also known as the *and*-operator and represented by the logic symbol $\wedge$. A formula $\varphi = a \wedge b$ is true, if a *and* b are true.

Disjunction is also known as the *or*-operator and represented by the logic symbol $\vee$. A formula $\varphi = a \vee b$ is true, if either a *or* b is true. The negation or *not*-operator is represented by the logic symbol $\neg$. A formula $\varphi = \neg a$ is true, if a is not true. This operator binds stronger than any other

which is crucial to consider. The *always*-operator is represented by the logic symbol $\Box$. A formula is true if $\varphi = \Box a$ if a *always* holds, as can be seen in figure 1 The final operator to be introduced here is the *eventually*-operator represented by the logic symbol $\Diamond$. The intuition behind these operators can be seen in figure 1.

The Positive-Normal-form (PNF) introduced in [1] describes a form any LTL-formula can be transformed in to. In this form negations are only allowed adjacent to a literal, e.g. a and $\neg a$ and the operators $\wedge$ and $\vee$. For example $\varphi = \neg a \wedge ((\neg b \wedge c) \vee \neg a)$ is on PNF while $\varphi = \neg(a \wedge \neg b)$ is not.

The Until- and Next-operator are not considered as their effects can be recreated by entering a new formula. The until operator would be of interest if one wants to explore an area *until* one reaches a certain point or meets a constraint, which is not the case here as the converter is used in combination with path planning. The *next*-operator that could be of use in a situation where one is stuck in an area and should only be allowed leaving trough a specific area but can be replaced by setting a new goal. This could be replicated by issuing a second query.

The operators *eventually* and *always* are combined such that we get *eventually-always*, represented by the symbol-combination $\Diamond\Box$. *Eventually-always* resembles that something has to be true at some point in time and always stay true. Another combination employed in this report is the *always-not* represented by the symbols combination $\varphi = \Box\neg$ which is employed to ensure that something can never happen. For more on this see the safety-property in [1]. Note that a new operator is introduced in chapter 4 as the classical LTL-operators do not have the property needed to solve the considered problem.

2.3 Workspace Representations

Digital systems work on discretized abstractions of the real, continuous world hence a suitable representation of the world is needed. Such a representation is a grid map. A grid map uses a uniform subdivision of the world into regular shapes, also known as "tiles". These can be shaped square, triangular or hexagonal [17]. There are different types of grid maps and those used in this report are briefly introduced.

2.3.1 Labeled Maps

To interact with the real world, autonomous systems need a model that represents the environment or workspace respectively. One such map is a labeled grid map where the grid cells define locations on the map using Cartesian coordinates.

A grid map can for example be labeled by color, numbers or strings. Figure 2 shows a labeled-by-color map. The color pattern is based on a heat-map pattern.

2.3.2 Cost Maps

Cost maps are maps in which each grid cell has an integer value representing a cost of visiting that cell. Cost maps provide the advantage that free, marked and occupied space can be represented explicitly. A map labeled by colors can be translated to a cost map by for example using a look-up table.

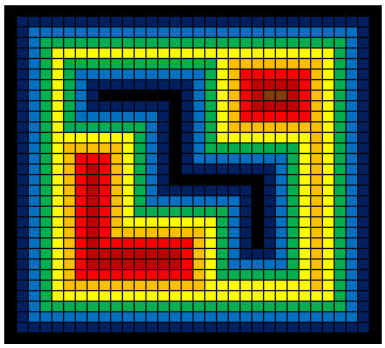
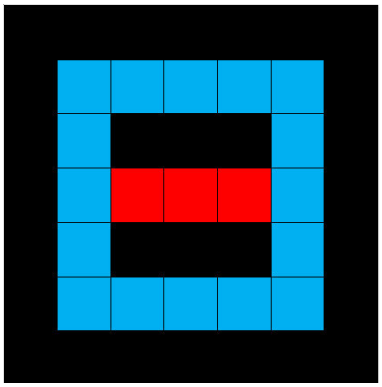


Figure 2: Simple Example of a Color Map

labeled map	cost map
blue	10
red	50
black	100

Table 1: Look-up Table: Labeled by color Map to Cost Map



(a) Color Map

100	100	100	100	100	100	100
100	10	10	10	10	10	100
100	10	100	100	100	10	100
100	10	50	50	50	10	100
100	10	100	100	100	10	100
100	10	10	10	10	10	100
100	100	100	100	100	100	100

(b) Cost Map

Figure 3: Color Map and Corresponding Cost Map

In this report the environment of the UAV is represented by a 2D labeled-by-color grid map. Therefore the method of choice is to use the look-up table transformation to go from the labeled-by-color map to the cost map. Translating this map into a respective cost map is a fast process in 2D and can be implemented straightforward when using the look-up-table transformation. Hence this direct translation has the advantages of straightforward implementation and fast computation.

2.4 Path Planning

Path planning is a widely spread research topic which affects all kinds of autonomous systems. The navigation path planning refers to finding a solution of going from the robots/systems current position to a goal position while avoiding obstacles, provided some localization. It is one of the

key components of autonomous systems and there are different solution approaches towards it. The purpose of a path planning algorithm is, given a map of the environment, that is the workspace of a robot, to subsequently attempt to create free paths for the robot to traverse without colliding with obstacles. To perform path planning one needs a localization system, a start and a goal position and a map. A path planning problem also referred to as a search problem, can be either represented by a search-graph or a search-tree. The search results can vary depending on which representation is used. More about this can be found in [2] Some of the most known algorithms for path planning are A*, RRT, D*, Voronoi, BFS and DFS. There are various modified versions of the above mentioned and others.

One way to evaluate the performance of such algorithms is presented in [2] by the four indicators completeness, optimality, time complexity and space complexity. These indicators allow a first evaluation of which search-algorithm suits best for a problem. Note that these can vary for the same algorithm depending on which method is used (graph-search or the tree-search), see figure 4 or 5 respectively.

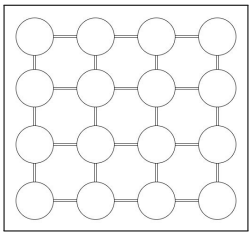


Figure 4: Nodes for Graph Search

2.4.1 Search Strategies

The major distinction between different types of search strategies is between uninformed and informed search. A search-algorithm is said to be uninformed if it does not use any information about how far the robot has traveled or how far away the robot is from the goal. Uninformed search also called blind search refers to the fact that the search strategy has no additional information about states beyond than provided in the problem definition. These algorithms generate successors and distinguish a goal state from a non-goal state. Two representatives of this type of search are for example Breadth-first search and Depth-first search. These search strategies are just an excerpt of many, the interested reader may find more in [2].

Informed search strategies on the other hand employ problem-specific knowledge which leads to solutions in a more efficient way. This, also known as a heuristic, can be for example implemented in form of a cost function in which additional knowledge of the problem is imparted to the search algorithm.

A common representative of the informed search algorithms is the Best-First-Search. This Best-First-Search selects which node to expand according to an evaluation function $f(n)$. This evaluation function is designed as a cost estimate such that the node with the lowest cost gets expanded first. For Best-First-Search-algorithms the evaluation function contains a heuristic function $h(n)$ = estimated cost and finds cheapest path from the state at node n to a goal state. One famous descendant of this algorithm is the A*-algorithm.

2.4.2 Breadth-First-Search

Given a search-tree the BFS-algorithm starts by expanding the root-node (starting point). When all direct successors to the root node are expanded, it continues with their successors expansion in the next layer and does not change layer before not all nodes within are expanded, see figure 5. Hence always the shallowest node is chosen for expansion by using a first-in-first-out queue for the frontier. New nodes are stacked to the back of the queue and old nodes (which are shallower) are expanded first. It keeps a list of already visited nodes and does not count them as new successors if they have been visited before.

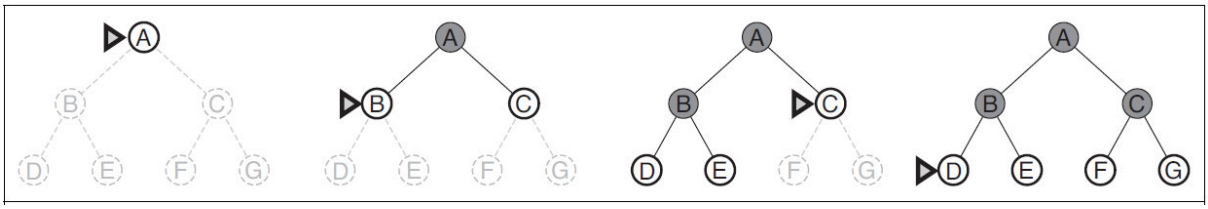


Figure 5: BFS-search tree ,excerpt from [2]

The BFS-search algorithm is complete as, it always eventually finds a solution if there exists one. It is optimal if the path cost is a non-decreasing function of the depth of the node. In terms of time complexity and space complexity the BFS is not the best choice for big search problems as each every node generated needs memory [2].

2.4.3 Depth-First Search

Given a search-tree the DFS-algorithm starts by expanding the root-node (starting point) and then continuing with the first deepest node. The search continues in this branch and until there are no succeeding nodes left. When arrived at this frontier the search "backs up" to the next deepest node that still has unexplored successors as can be seen in figure 6. In contrast to BFS, DFS uses a

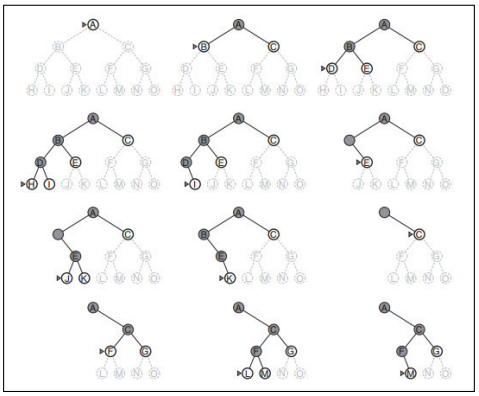


Figure 6: DFS-Search Graph, excerpt from [2]

last-in-first-out queue, meaning the recently visited node is expanded. The properties of DFS vary depending on whether the graph-search or tree-search method is used.

In conclusion DFS applied to a graph search has no clear advantages over BFS, but in tree search DFS needs to only store one branch with its unexplored nodes. The trick is to remove a node from memory as soon as it was fully explored and it has no further successors. So in terms of space complexity in tree-search DFS is superior to BFS.

A full comparison of uninformed search strategies is provided in [2] where the authors compare BFS, Uniform-cost search, DFS and Iterative deepening depth-first and some variations of them against each other in terms of completeness, time consumption, space consumption and optimality.

2.4.4 A^*-Search Algorithm

The most widely known form of best-first-search is called "A-star" and belongs to the informed search algorithms. It is the method of choice in this report as it is optimal and complete, meaning it always finds an optimal solution if there exists one. Another advantage that comes with this algorithm is that there are many already working implementations available.

A* searches for a path over a discrete state space. One such state space representation is a grid map. Note that the success of the algorithm depends in the discretization-step size. It performs a graph search where each node in the graph represents a grid cell. Given a cost-map A* evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the cost to get from the node to the goal: $f(n) = g(n) + h(n)$. Here $g(n)$ represents the path cost from the start node to the current node n, and $h(n)$ is the estimated cost of the cheapest path from n to the goal. Hence $f(n)$ = estimated cost of the cheapest solution through n.

The purpose of the heuristic function is to return an estimate of the distance from a node to the goal node. The heuristic function determines the output and affects the time complexity. The idea is to choose $h(n)$ such that it is well-balanced and reflects the domain, while not draining too heavily on available resources. When using A* for path planning in a grid environment, the most common heuristics are according to [18] Manhattan distance, Euclidean distance and Chebyshev distance.

Algorithm 1: A*

```
   Input: start, goal(n), h(n), expand(n)
   Output: path
1  if goal(start) = true then return makePath(start)
2
3  open ← start
4  closed ← ∅
5  while open ≠ ∅ do
6  |   sort(open)
7  |   n ← open.pop()
8  |   kids ← expand(n)
9  |   forall the kid ∈ kids do
10 |   |   kid.f ← (n.g + 1) + h(kid)
11 |   |   if goal(kid) = true then return makePath(kid)
12 |   |   if kid ∩ closed = ∅ then open ← kid
13 |   closed ← n
14 return ∅
```

Figure 7: A^*-Pseudo-code, excerpt from [2]

There are more advanced search-methods such as the sampling based Rapidly-Exploring Random Trees (RRT) [19], that is specifically designed to handle high degrees of freedom and converges towards an optimal solution over time. This more advanced method is not needed as the A*-algorithm is sufficient for the given search problem in this report.

2.5 Motivating Examples

The goal of this master thesis is to present a framework for planning based on Linear-Temporal-Logic-formulas. These formulas represent instructions that were originally formulated in natural language. As there is no speech recognition or translator yet available to deal with this task, and natural language is highly ambiguous, a first step towards this would be to have a fixed-input structure where a user can manually enter his instructions. Henceforth there is need for a standardized structure that allows natural language instructions being represented as an LTL-formula. We therefore search for common natural language signal words to express the meaning of linear temporal logic-operators.

The following motivating examples emphasize what is meant by searching for signal words in the natural language formulation and finding corresponding LTL-operators.

These examples provide an idea how instructions formulated in natural language can be represented as a Linear-Temporal-Logic formula by exploiting the fact stated in [14] that LTL has a certain resemblance to NL. The task considered here is that a human instructs someone to go from A to B, by using natural language to express the task.

Example 2.1. The task expressed in NL: 'Go to school and avoid the highway'.
Exploiting the resemblance-relation between NL and LTL and we can find corresponding LTL-operators. We identify 'go to' and 'avoid' as signal words in NL, that correspond to the LTL-operators 'eventually-always' and 'always-not'. Employing the notation introduced in the preliminaries we get

NL-expression	LTL-expression
'Go to'	$\Diamond\Box$
'avoid'	$\Box\neg$

Table 2: LTL representations for NL

Hence, based on table 2 the task can be expressed as the LTL-formula

$$\varphi = \Diamond\Box school \wedge \Box\neg highway. \tag{2}$$

However when formulating such instructions there is often more to it, than just stating the start and the goal. Instead of taking the shortest path regarding time or distance it can be desirable to take a different one. Be it for safety reasons or just convenience as the following example shows.

Example 2.2. Let the task formulated in NL be: 'Go to school, prefer sidewalk 1 and avoid the highway.' Based on the signal words in 2 we obtain

NL-expression	LTL-expression
'Go to'	$\Diamond\Box$
'avoid'	$\Box\neg$
'prefer'	?

Table 3: LTL representations for NL

We note that there is no LTL-operator that represents 'prefer' but there is the 'eventually'-operator, which would lead to the following LTL-formula when employed:

$$\varphi = \Diamond\Box school \wedge \Box\neg highway \wedge \Diamond sidewalk1. \tag{3}$$

However this formula 3 would only be true, if sidewalk 1 was visited at some point along the path. But if it was for some reason not possible to use sidewalk1, be it due to traffic or construction, the formula would not be true when deploying the $\Diamond$-operator and hence the task would not be fulfilled. So we can conclude that the standard 'eventually'-operator introduced in the preliminaries does not express the desired property. In order to steer someone by recommendation but giving a higher priority to the task fulfillment, another operator is needed.

3 Problem Definition

The problem considered in this master thesis is to develop a framework that, given a labeled map and constraints expressed as an LTL-formula, converts the labeled map into a cost-map that can be used to plan a path. If several solutions are obtained a clarification request is issued such that the user can choose which map-path combination should be used and executed.

3.1 Assumptions

In order for a framework to be feasible and to produce the desired result in form of a cost map that can then be used to plan a path the following assumptions and restrictions are necessary:

(i) The environment is known (a pre-labeled map is provided)

(ii) Start and goal are provided

(iii) The constraints are given as a valid LTL-formula

(iv) A path from start to goal exists

4 Solution

To solve the given problem a skeleton is constructed which provides a fixed structure for handling the constraints extracted from an LTL-formula. This skeleton allows the constraints indicated by signal words such as *go to, prefer* and *avoid* to be entered directly. It provides a standardized structure where the constraints must be put in and is then fed together with a labeled-map into the map-converter. Using the given LTL-constraints, the map converter translates the labeled map into a cost map. This cost map is then used to plan a path from the start to the goal position. Should the given LTL-formula contain various options that lead to an output of multiple maps and respective paths, a clarification request is issued to the user. The maps, paths and corresponding formulas are then displayed and the user must choose which one should be executed. After the user has made his choice the desired path is send as a list of way-point to a controller such that it can be executed.

The following sections give a detailed insight into the main parts of the solution as indicated in figure 8.
The five main parts are:

1. The Input-Skeleton
2. LTL-to-Cost-Map Conversion
3. Path Planning
4. Clarification Request

5. Execution of the Path

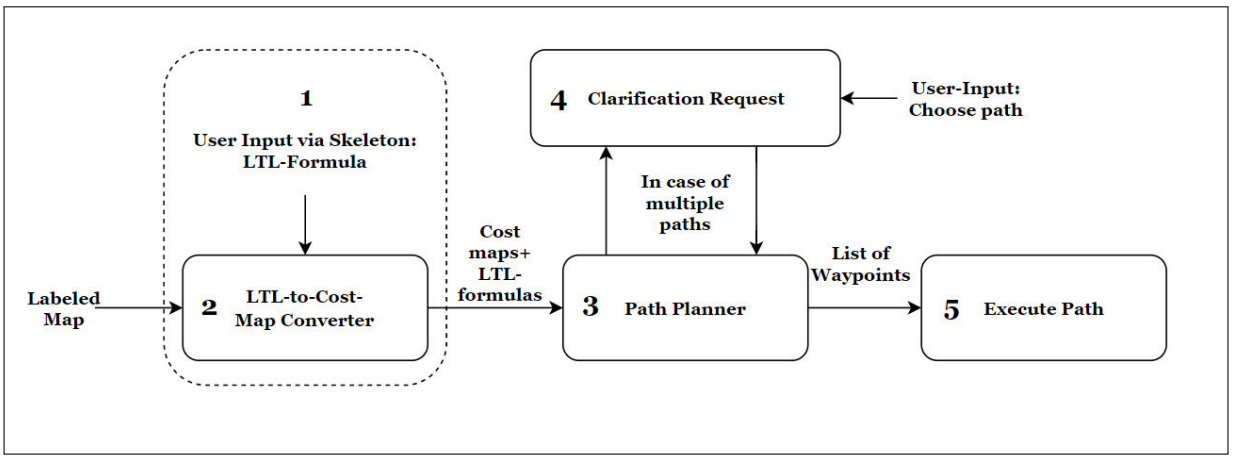


Figure 8: Solution Overview

4.1 The Input-Skeleton

The input-skeleton provides a fixed structure for entering the constraints into the framework. It consists of three parts, further referred to as arrays. The *always-not*-array, the *preference*-array and the *eventually-always*-array. The corresponding logic symbols are '$\square\neg$', '$\blacklozenge$' and '$\lozenge\square$'. The names of these operators are the signal-words, indicating which constraint is subject to the respective array. In order to combine these logic operators and to clarify their interpretation, the following definitions are given.

Definition 4.1.1. *Always-Not* expressed by '$\square\neg$', is defined such that the formula $\varphi = \square\neg[blue]$ is true, if and only if blue is never visited.

Definition 4.1.2. *Eventually-Always* expressed by '$\lozenge\square$', is defined such that the formula $\varphi = \lozenge\square[blue]$ is true, if and only if blue is visited and can never be left.

As shown in the motivating examples there is need for a modified version, a weakened version, of the 'eventually'-operator defined in [1] to represent the 'preference'-expression. Therefore a new operator is introduced based on it.

Definition 4.1.3. We denote the *Preference*-operator as *weak-eventually* by '$\blacklozenge$' that comes with a '*count*'. The count is defined as a preference measure.
Let $\blacklozenge\varphi \vDash true$ such that $\blacklozenge$ always holds and gets $+1$ in count, when $\lozenge\varphi = true$ is satisfied as can be seen in figure 9.

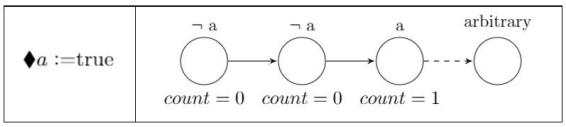


Figure 9: Preference-diagram

By violating the classical semantics and assuming that $\blacklozenge\varphi = true$, we create ourselves the opportunity to introduce a preference measure. This preference-measure states how many of the imposed constraints within this array are met, while still assuring that the overall-formula representing the task is true. This comes in handy when combining the converter with the path planning algorithm.

Based on these definitions 4.1.1, 4.1.2 and 4.1.3 the skeleton for the map conversion is constructed. There are two expressions which are defined to be the basic equations with the distinction of disjunction or conjunction between the '$\Box\neg$-array' and the '$\blacklozenge$-array'.

Definition 4.1.4. Therefore the basic equation in the skeleton for the conjunction relation is defined as

$$\Gamma_{\wedge} = \Big[\Box\neg[constraints] \wedge \blacklozenge[constraints]\Big] \wedge \Diamond\Box[goal]. \tag{4}$$

Definition 4.1.5. The basic equation for the disjunction relation in the skeleton is defined as

$$\Gamma_{\vee} = \Big[\Box\neg[constraints] \vee \blacklozenge[constraints]\Big] \wedge \Diamond\Box[goal]. \tag{5}$$

We will therefore only consider formulas of the form defined by 4.1.4 and 4.1.5. The notation of $\Gamma_{\wedge}$ is used to indicate that the skeleton definition 4.1.4 is employed and $\Gamma_{\vee}$ for definition 4.1.5 respectively. The input structure to the skeleton is fix and each array is initiated with one of the following operators, $\Box\neg, \blacklozenge, \Diamond\Box$. That is, instructions formulated as an LTL-formula must indicate for each contained constraint to which array it belongs. Each temporal operator-array that is part of the formula is connected with either *and*($\wedge$) or *or* ($\vee$). Note that for each disjunction within a temporal operator, the respective array gets expanded by one additional row and for each respective conjunction the array gets expanded by one column. If multiple constraints are given the expansions of the equations 4.1.4 and 4.1.5 are given by the following:

Definition 4.1.6.

$$\Gamma\wedge = \Bigg[\Box\neg \begin{bmatrix} n_{11} & \dots & \wedge & \dots & n_{1j} \\ \downarrow\vee & & & & \\ \vdots & \dots & \wedge & \dots & \wedge \\ \downarrow\vee & & & & \\ \vdots & & & & \\ n_{j1} & \dots & \wedge & \dots & n_{ij} \end{bmatrix} \wedge \blacklozenge \begin{bmatrix} p_{11} & \dots & \wedge & \dots & p_{1j} \\ \downarrow\vee & & & & \\ \vdots & \dots & \wedge & \dots & \wedge \\ \downarrow\vee & & & & \\ \vdots & & & & \\ p_{j1} & \dots & \wedge & \dots & p_{ij} \end{bmatrix} \Bigg] \wedge \Diamond\Box \begin{bmatrix} goal \\ \vdots \\ \downarrow\vee \\ \vdots \\ goal \end{bmatrix} \tag{6}$$

and respectively

Definition 4.1.7.

$$\Gamma\vee = \left[\Box\neg \begin{bmatrix} n_{11} & \dots & \wedge & \dots & n_{1j} \\ \downarrow_\vee & & & & \\ \vdots & \dots & \wedge & \dots & \wedge \\ \downarrow_\vee & & & & \\ \vdots & & & & \\ n_{j1} & \dots & \wedge & \dots & n_{ij} \end{bmatrix} \vee \blacklozenge \begin{bmatrix} p_{11} & \dots & \wedge & \dots & p_{1j} \\ \downarrow_\vee & & & & \\ \vdots & \dots & \wedge & \dots & \wedge \\ \downarrow_\vee & & & & \\ \vdots & & & & \\ p_{j1} & \dots & \wedge & \dots & p_{ij} \end{bmatrix}\right] \wedge \Diamond\Box \begin{bmatrix} goal \\ \vdots \\ \downarrow_\vee \\ \vdots \\ goal \end{bmatrix} \tag{7}$$

where n:never-constraints, p:preference constraints

Remark 1. *The notation given in definition 4.1.4 should be read as*

$$\Gamma\wedge = \left[\begin{bmatrix} \Box\neg n_{11} & \dots & \wedge & \dots & \Box\neg n_{1j} \\ \downarrow_\vee & & & & \\ \vdots & \dots & \wedge & \dots & \wedge \\ \downarrow_\vee & & & & \\ \vdots & & & & \\ \Box\neg n_{j1} & \dots & \wedge & \dots & \Box\neg n_{ij} \end{bmatrix} \wedge \begin{bmatrix} \blacklozenge p_{11} & \dots & \wedge & \dots & \blacklozenge p_{1j} \\ \downarrow_\vee & & & & \\ \vdots & \dots & \wedge & \dots & \wedge \\ \downarrow_\vee & & & & \\ \vdots & & & & \\ \blacklozenge p_{j1} & \dots & \wedge & \dots & \blacklozenge p_{ij} \end{bmatrix}\right] \wedge \begin{bmatrix} \Diamond\Box goal \\ \vdots \\ \downarrow_\vee \\ \vdots \\ \Diamond\Box goal \end{bmatrix} \tag{8}$$

Meaning we denote LTL-formulas by φ. When referring to the formulas plugged in into the skeleton we refer to $\Gamma_\vee$ or $\Gamma_\wedge$

Depending on the user's input constraints either equation 4 or equation 5 is used and the map conversion is executed. It is recommended that the desired LTL-formula is manipulated into positive normal form (PNF), meaning $\neg(a \wedge b)$ should be expressed as $\neg a \vee \neg b$. See [1] for more on PNF. However there are many ways to represent the same expression, meaning formulas of different lengths and different operators used can have the same resulting cost map(s). The essential idea behind the skeleton is that, for all constraints belonging to one temporal-operator there is one array in which they must be filled in. Within this array the boolean connectors *conjunction* and *disjunction* are of great importance, as the array is to be filled with the respective constraints. The *disjunction*-operator leads to different results than the *conjunction*-operator. as the array's dimension expands depending on the constraints. For each $\vee$-constraint it is expanded by one additional row and for each $\wedge$-constraint by one column. This follows for all arrays. The total number of resulting maps is given by: $n_{maps} = rows_{not} \cdot rows_{weak-eventually} \cdot rows_{goal}$. The following examples show how to enter constraints into the skeleton. In definitions 4.1.7 and 4.1.6 the extended skeleton for $\Gamma_\wedge$ and $\Gamma_\vee$is presented including its expansion by rows and columns. To ensure that there is always a goal brackets are used to guarantee that the formula is true throughout the entire run.

Example 4.1. Skeleton-Input:
'*Avoid* a and *prefer* c and *end-up in* d' **OR** '*avoid* b and *prefer* c and *end-up in* d. A possible

LTL-expression for this task is given by:

$$\varphi = (\square\neg a \wedge \blacklozenge c \wedge \lozenge\square d) \vee (\square\neg b \wedge \blacklozenge c \wedge \lozenge\square d) \tag{9}$$

in order to plug this formula 9 into the converter one must first bring it onto the form fitting the skeleton, which in this case is given by equation 4

$$\Gamma_\vee = \Big[\square\neg[constraints] \wedge \blacklozenge[constraints]\Big] \wedge \lozenge\square[goal]$$

and results in the skeleton input array

$$\Gamma_\vee = \left[\square\neg \begin{bmatrix} a \\ b \end{bmatrix} \wedge \blacklozenge[c]\right] \wedge \lozenge\square[d] \tag{10}$$

Hence for the resulting array-expression we get different maps to choose from. As in the context of path-planning a goal is always required, the first two operators are grouped together with brackets to ensure the validity of the expression.

4.2 The LTL-Map-Converter

In this section the map conversion from a labeled map to a cost-map is described with respect to each part of the array. The map conversion is subject to constraints that have to be formulated as an LTL-formula. The map of the environment is assumed to be a grid map with fix grid size, as briefly introduced in the preliminaries and to be labeled-by-color. An I/O-Scheme of the LTL-to-Cost-Map-converter is illustrated in figure 10.

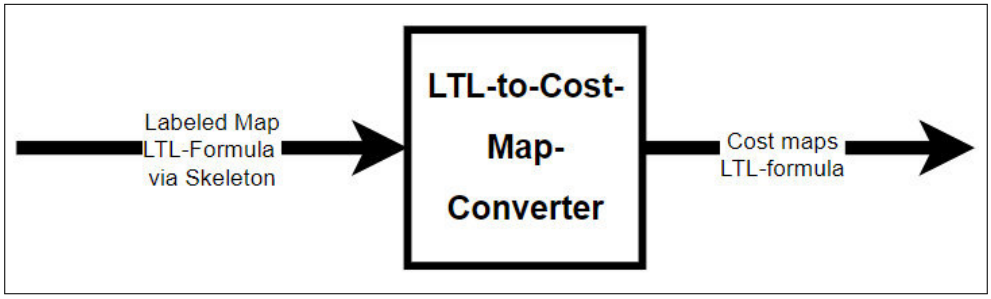


Figure 10: I/O-Scheme for LTL-Map Converter

4.2.1 The Always-Not-Array

This section focuses on the $\square\neg[\,]$ part of equation 6. It is described how the array can be expanded for several always-not-commands.

$$\Gamma = \Big[\square\neg[\,...] \wedge \blacklozenge[\,...]\Big] \wedge \lozenge\square[\,...] \tag{11}$$

This expression '$\square\neg[\,]$' represents a safety-feature and is treated with the highest priority. It ensures that something can *'always-not'* happen which is equivalent to the signal word *'never'* or *'avoid'*. In

the context of the map-conversion this means that the area inside this part can never be visited and hence gets assigned the highest cost. The constraints entered in this part of the array are considered hard constraints.

Example 4.2. Given the instructions to *'never'* visit either the yellow or orange areas, a possible LTL-translation is given by

$$\varphi = \Box\neg(yellow \wedge orange). \tag{12}$$

which corresponds to $\varphi = \Box\neg yellow \vee \Box\neg orange$. Rewriting it in terms of the skeleton for the map conversion leads to

$$\Gamma = \Box\neg \begin{bmatrix} yellow \\ orange \end{bmatrix}. \tag{13}$$

Note that here we only state which areas are not to be visited as this is sufficient to illustrate the effect of this part of the skeleton. The array is expanded due to the fact that for $\vee$-operator one additional option given, that is one additional map will be obtained. This leads to the fact that given the color map in figure 11 and the skeleton input 13, two cost maps are obtained. These cost maps illustrate areas that must never be visited by black, representing a very high cost. Note that the cost value can be set manually in the converter according to the user's intention.

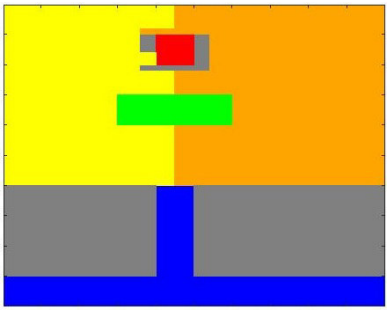


Figure 11: Color Map of an Environment

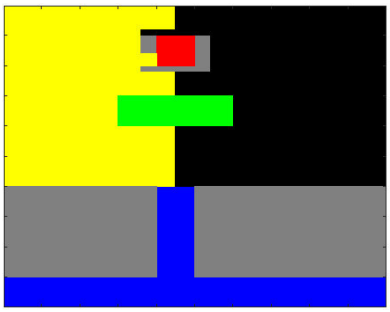


Figure 12: Cost map for $\varphi = \Box\neg orange$

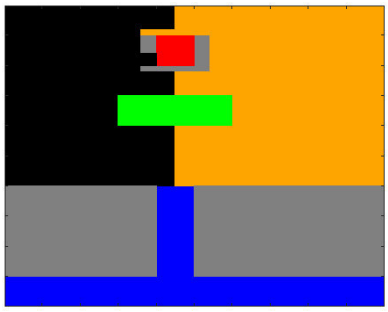


Figure 13: Cost map for $\varphi = \Box\neg yellow$

Note that the figures 12 and 13 illustrate the effect of the converter on the original map. Areas associated with the *always-not*-array become black which represents a high cost.

4.2.2 The Weak-Eventually-Array

The *weak-eventually*-operator defined in 4.1.3 enables the user to state a preference regarding the constraints. In contrast to the hard constraints entered in the other parts of the array, the constraints in here can be violated. The preference-measure indicates how many of the constraints are met or violated.

$$\Gamma_\wedge = \left[\Box\neg[\ \dots\ ] \wedge \blacklozenge[\ \dots\ ]\right] \wedge \Diamond\Box[\ \dots\ ]. \tag{14}$$

$$\Gamma_\vee = \left[\Box\neg[\ \dots\ ] \vee \blacklozenge[\ \dots\ ]\right] \wedge \Diamond\Box[\ \dots\ ]. \tag{15}$$

This part is also represented by a matrix that with each $\vee$-connector that is used (expands the matrix by one row), resulting in a number of new maps. The effect of this array can be seen in figure 15.

This is beneficial as any cost-map-based path planner prefers areas stated in this array. It assigns the same cost as the eventually-always-array regarding the map-conversion but adds a count depended on the path.

4.2.3 The Eventually-Always-Array

In this array the goal-region must be defined. To demonstrate the effect of the eventually-always-array it is sufficient to only state the final region of interest. Note that when combined with the path planning the goal must lie within this region. The map conversion itself does not need a specific goal expressed by coordinates to perform map-conversions. Given the basic equation 4 now the third part, namely the eventually-always part is discussed

$$\Gamma = \left[\Box\neg[\ ...] \wedge \blacklozenge[\ ...]\right] \wedge \Diamond\Box[\ ...]. \tag{16}$$

Constraints subject to the input of this array must be true *eventually-always*, meaning they are hard constraints. Hence this part contains the area or region of interest where the goal or goals must lay within. If there are multiple regions of interest entered, the result will be multiple maps. A demonstration of the effect of the *weak-eventually* and *eventually-always*-array is given by the following. Note that in this array only $\vee$ are allowed as $\wedge$ is impossible to satisfy.

Example 4.3. Instruction are given by 'Prefer blue and go to orange.' Rewriting this as an LTL-formula leads to

$$\varphi = (\blacklozenge blue \wedge \Diamond\Box orange) \tag{17}$$

Inserting this into the skeleton

$$\Gamma = \left[\blacklozenge \left[blue\right]\right] \wedge \Diamond\Box \left[orange\right] \tag{18}$$

Given the color map 14 of the environment, the resulting cost map is given by figure 18, where the white area represents zero-cost.

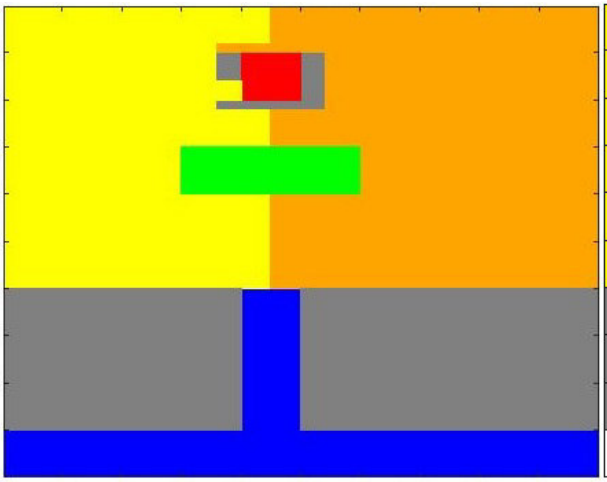


Figure 14: Color Map of an Environment

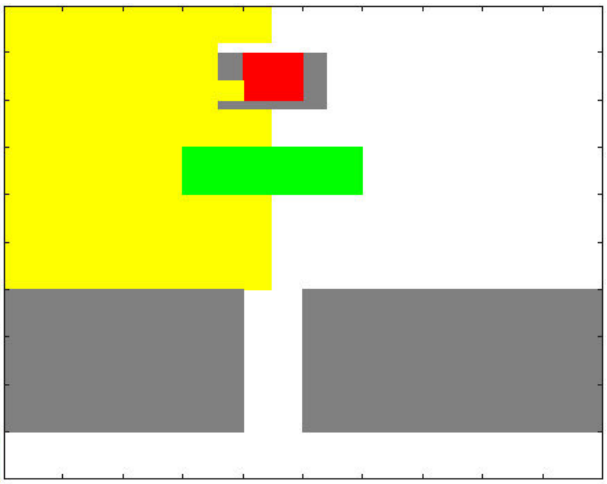


Figure 15: Resulting cost-map for formula 18

The two equations 6 and 7 are the core part of the map conversion as based on these operators the map conversion is executed. Note that the value each operator assigns its respective areas defined in the constraints, can be set according to the user's preference. That is, the range of values in the cost-maps.

4.2.4 The Safe-Mode

The safe-mode option provides the possibility to choose how to handle not-specified areas in the map conversion. Moreover it ensures that each cell in the map has been taken into account and assigned a specific value the user is aware of. If the safe-mode is active, areas that are not explicitly specified in form of constraints within the skeleton are treated as occupied and get therefore a high cost assigned. This ensures that with an active safe-mode the LTL-formula is always true given a reasonable user input. This effect of an active safe-mode can be seen in figure 16 when applied to the formula 18.

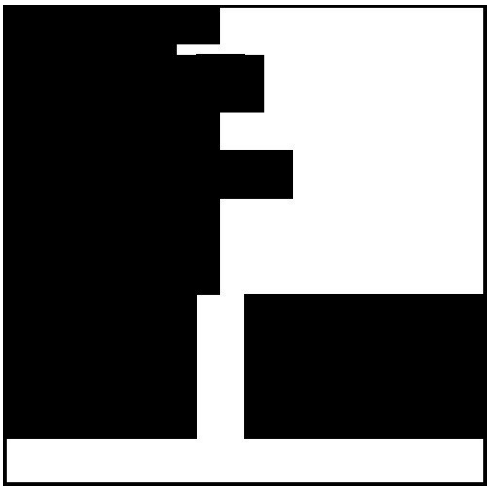


Figure 16: Safe-mode on for formula 18

Figure 17: Safe-mode off for formula 18

Disabling the safe-mode is a valid option too. In this case, areas that are not specified within the constraints are assigned a low cost (e.g.10%), while the preferred areas are free of cost. This allows to steer the path planning algorithm via the constraints within the *weak-eventually*-array and comes in handy when combined with the path planner afterwards. As the path planning algorithm used in this report tries to find the shortest i.e. cheapest path, an inactive safe-mode enables the user to guide it through detours that are of interest and are not the shortest path but of a cheaper cost.

This approach is constructed in such a way that its output, which is given by cost maps can be plugged into any path planner. Therefore it is universally applicable and can be used as a tool box ready for plug-and-play.

4.3 Path Planning using A^*

The path planning problem aims to find a path through an environment given a start and a goal position. There are numerous approaches and algorithms available that solve this problem as discussed in chapter 1. For this framework the A*-algorithm was the method of choice as it is complete and optimal. This cost-map based algorithm belongs to the informed search algorithms as it considers

problem specific information and includes it into the decision making process by emplyoing a heuristic function. The pseudo-code for this algorithm can be found in figure 7 in the preliminaries of this report. Start and goal position as well as a cost map of the environment are essential information that must be provided. This information is subject to the input of the path planner. Note that in this framework the cost-map is created by the LTL-to-cost-map-Converter. The path planner takes the cost-map and expands the obstacles in it with respect to the size of the UAV before performing the path search. The UAV itself is treated as a point mass. For more details see section 5.2.4. Based on this expanded obstacle map and the provided start and goal position the path planner searches for the shortest or cheapest path with respect to the underlying heuristic. Due to its completeness, the A*-algorithm is guaranteed to find a path if there exists one. The obtained path is presented as a list of way points. This way-point list is then smoothed by the line-of-sight-method to facilitate the latter path execution. Note that smoothing is recommended but not necessary. For a concrete example see section 5.2.5.

However this part of the framework is not limited to this specific search algorithm. It is designed in such a way that the path planner can be exchanged to any other cost map-based planner, for example RRT or D*.

4.4 The Clarification Request

In the case of multiple solutions in form of resulting cost maps and corresponding paths, a clarification request is issued. As illustrated in figure 18 the user then has to choose which path should be executed. For a concrete example see section 5.2.6.

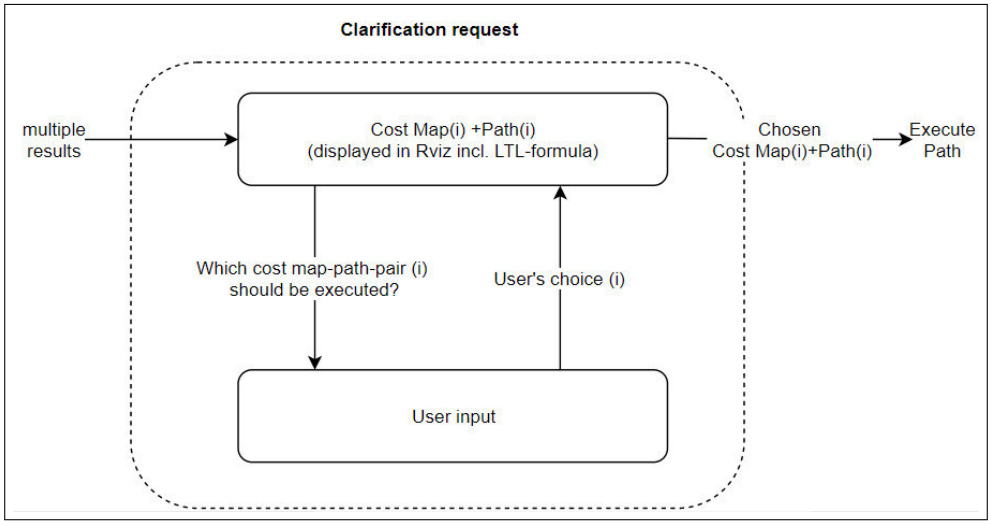


Figure 18: Principle of the Clarification Request

4.5 UAV Path Execution

The way-point list obtained by the path planner is subject to the input of the path-execution, that is the controller of the UAV. The UAV traverses towards the first way point and as soon as it has

reached its close proximity, it will aim for the next one. This is repeated until it reaches the final way-point, that is the goal position. Figure 19 shows this principle for the path execution.

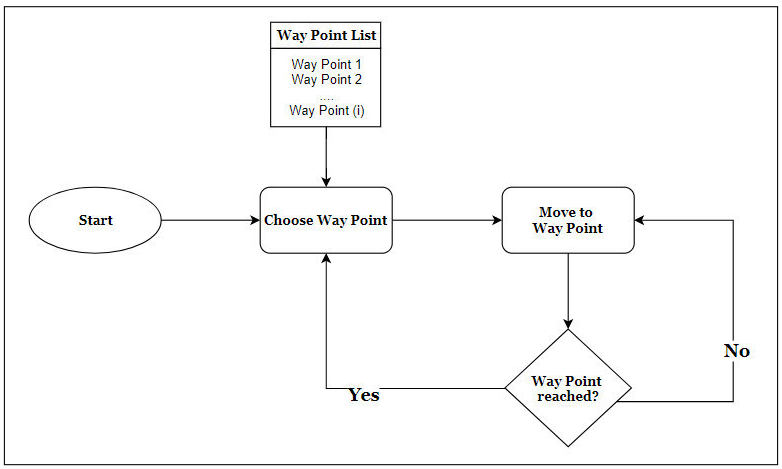


Figure 19: Principle of the Path Execution for the UAV

5 Experiments

The validation of the proposed method is performed in two steps. The final result is shown by a real world experiment. First a proof-of-concept is done by implementing the map-converter in MATLAB and experiments based on the motivating examples are performed to ensure its performance. In the second step, the framework is implemented in ROS and the GAZEBO simulation environment to see how a UAV traverses through an environment. The cost-maps are obtained based on the task. These cost-maps are then fed into the path planner and then outputs are displayed in RVIZ. The resulting way-points are displayed in form of the paths and simulations on how a UAV follows the obtained path are performed. As a final step a real-world-experiment is performed where a real UAV within an arena is used. The resulting discrete paths are represented by a list of way points and displayed. To indicate which basic equation is used during the LTL-map-conversion, a formula gets assigned either a lower case $\vee$ or $\wedge$, leading to the notation $\varphi_{\wedge}$ and $\varphi_{\vee}$ or in terms of the input skeleton $\Gamma_{\wedge}$ and $\Gamma_{\vee}$ respectively.

5.1 Map-Conversion- Experiments

In this section the solution approach introduced in the previous chapter is implemented in MATLAB as proof of concept.

5.1.1 Experiment 1

Going back to the motivating examples 2.1-2.2 one can observe that the instructions contain signal words like *not, avoid, prefer* and *eventually-always, go to, end-up*:
'Go to school, do **not** walk on the highway and **prefer** the sidewalk', can be translated to the LTL

formula

$$\varphi = \square\neg highway \wedge \blacklozenge sidewalk \wedge \lozenge\square school \tag{19}$$

and the resulting equation for the framework is then given by

$$\Gamma_{\wedge} = \Big[\square\neg[highway] \wedge \blacklozenge[sidewalk]\Big] \wedge \lozenge\square[school] \tag{20}$$

An illustration of labeled-by-color map of an environment can be seen in figure 20

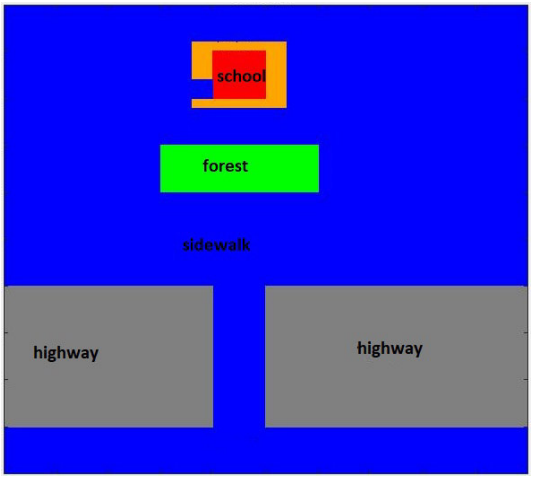
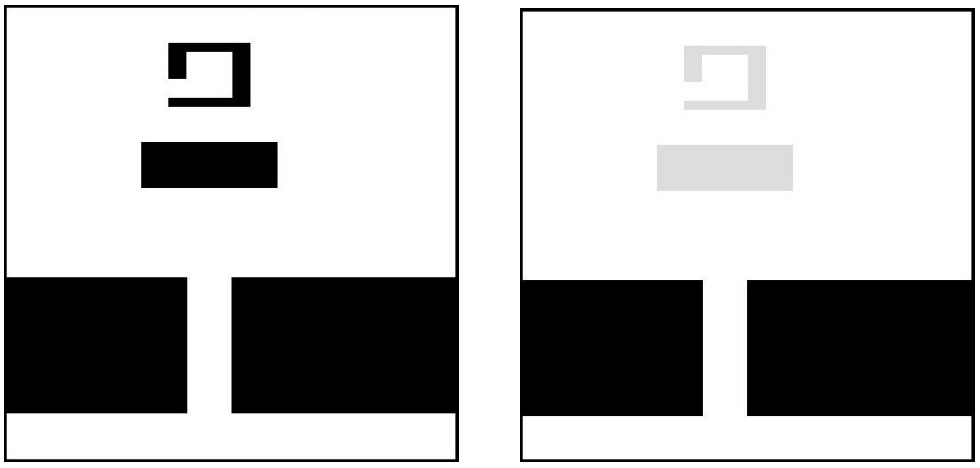


Figure 20: Color Map of Environment 1



(a) Active Safe-Mode (b) Non-Active Safe-Mode

Figure 21: Resulting Cost Maps

Given the labeled-by-color map illustrated in figure 20, applying the LTL-formula 19 a resulting cost map is obtained. However to demonstrate the effect of the safe-mode, the resulting cost map with and without safe-mode is shown 21. Whether a user wants or wants not to make use of the safe-mode depends on the task and the users' preferences. The safe-mode allows the user choose

how the LTL-map-converter handles labels that are not defined in the LTL-formula, but occur in the labeled-by-color map.

5.1.2 Experiment 2

Now moving to another example. The labeled-by-color map of the environment remains the same as well as the task. Instead of the boolean *and*-connector, the boolean *or* connector is used, which leads to two possible formulations according to the definition of the LTL-map-converter.

$$\varphi = (\square\neg\ highway \vee \blacklozenge sidewalk) \wedge \lozenge\square school \tag{21}$$

Rewriting the task formulation in the adequate form such that the framework can handle it, leads to

$$\Gamma_{\vee} = \Big[\square\neg\ [highway] \vee \blacklozenge [sidewalk]\Big] \wedge \lozenge\square [school] \tag{22}$$

The converter separates the formula 21 due to the $\vee$-connector.The goal which is initiated by the $\lozenge\square$- operator stays the same and hence the resulting LTL-expressions:

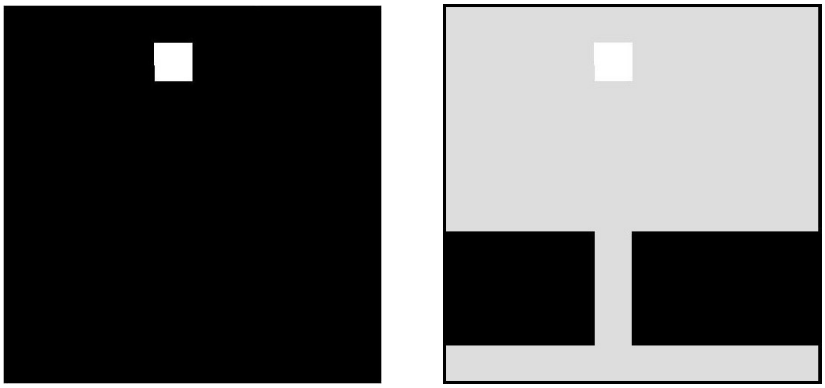
$$\varphi_{\vee_1} = \square\neg highway \wedge \lozenge\square school \tag{23}$$

and

$$\varphi_{\vee_2} = \blacklozenge sidewalk \wedge \lozenge\square school \tag{24}$$

For each of the two expressions 23 and 24 one respective cost-map is generated, leading to a total of two resulting cost maps. Again for the sake of completeness the result for the two safe-modes are displayed which leads to a total of four maps.

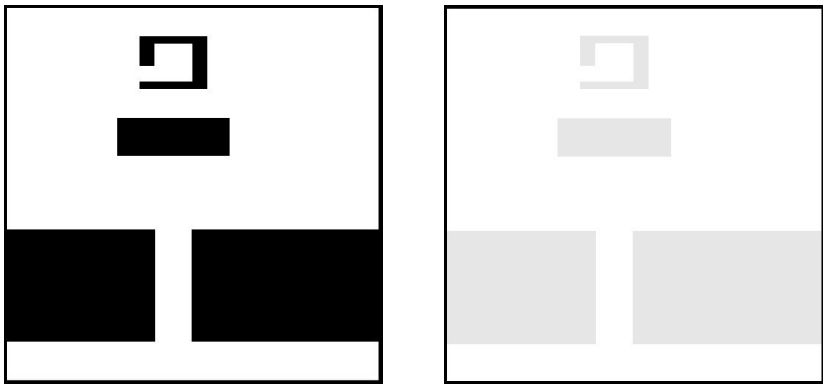
It is obvious that figure 22(a) represents an active safe-mode, meaning all area that is not specified within the formula is assumed to be occupied (and gets therefore automatically assigned a high cost). The safe-mode being not active results in a low cost (grey) assignment for the not specified labels, see figure 22(b).



(a) $\varphi_{\vee_1}$ Safe-Mode Active (b) $\varphi_{\vee_1}$ Safe-Mode Not Active

Figure 22: Resulting Maps for $\varphi_{\vee_1}$ with and without Safe-Mode

Figure 23 shows the results for an active and inactive safe-mode for equation 24.



(a) $\varphi_{\vee_2}$ Safe-Mode Active (b) $\varphi\vee_2$ Safe-Mode Not Active

Figure 23: Resulting Maps for $\varphi_{\vee_2}$ with and without Safe-Mode

5.1.3 Experiment 3

The task remains the same as before in example 5.1.1 and 5.1.2 but the LTL-formula 25 for this example contains more details. A more detailed labeled-by-color map of the environment is used as can be seen in figure 24.

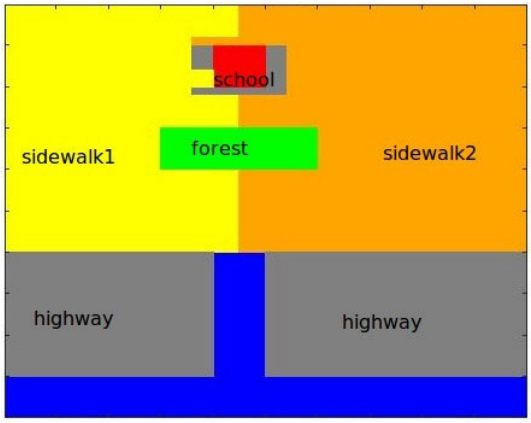


Figure 24: Color Map of Environment 3

The task formulation is given by: 'Go to school and never walk on the highway and never in the forest and prefer to stay on either sidewalk1 or sidewalk 2.'

$$\Gamma = \Big[\square\neg\,[highway \wedge forest] \wedge \blacklozenge[sidewalk1 \vee sidewalk2]\Big] \wedge \lozenge\square school \tag{25}$$

or written on the general array form on how the converter interprets it:

$$\Gamma_{\wedge} = \Big[\square\neg \begin{bmatrix} highway & \wedge & forest \end{bmatrix} \wedge \blacklozenge \begin{bmatrix} sidewalk1 \\ \\ sidewalk2 \end{bmatrix} \Big] \wedge \lozenge\square \Big[school\Big] \tag{26}$$

The converter splits the formula 26 due to the $\vee$-disjunction inside the $\blacklozenge[instructions]$,leading to the following two expressions in the skeleton:

$$\Gamma_{\wedge 1} = \Big[\Box\neg[highway \wedge forest] \wedge \blacklozenge[sidewalk1]\Big] \wedge \Diamond\Box[school] \tag{27}$$

and respectively

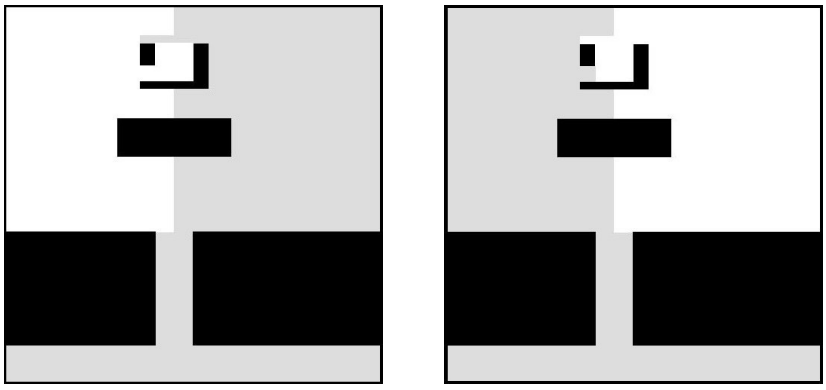
$$\Gamma_{\wedge 2} = \Big[\Box\neg[highway \wedge forest] \wedge \blacklozenge[sidewalk2]\Big] \wedge \Diamond\Box[school] \tag{28}$$

Again four resulting cost-maps are presented to demonstrate the effect of the safe-mode. Figure 25 shows the effect of an active safe-mode for both formulas 27 and 28. In formula 27 it is not explicitly stated how to treat sidewalk 2, therefore is gets assigned a high cost as can be seen in figure 25a. The same happens to sidewalk1 in the resulting cost-map for formula 28 as can be seen in figure 25b. The area labeled as sidewalk1 gets assigned a high cost (black).

(a) $\Gamma_{\wedge_1}$ Safe-Mode Active (b) $\Gamma_{\wedge_2}$ Safe-Mode Active

Figure 25: Resulting Cost Maps for $\Gamma_{\wedge_1}$ and $\Gamma_{\wedge_2}$ with Active Safe-Mode

The resulting cost maps without the safe mode are illustrated in figure 26. They grey areas are of low cost and only the white areas specified in the skeleton are of zero cost. Areas stated within the *always-not*-array of the framework are forbidden and get therefore assigned a high cost.



(a) $\Gamma_{\wedge_1}$ Safe-Mode Not Active (b) $\Gamma_{\wedge_2}$ Safe-Mode Not Active

Figure 26: Resulting Cost Maps for $\Gamma_{\wedge_1}$ and $\Gamma_{\wedge_2}$ with Not Active Safe-Mode

5.1.4 Evaluation of Map-Conversion-Experiments

The presented examples 5.1.1 to 5.1.3 confirm the expected results. The map converter creates several cost maps, based on a natural language formulation of a task which is translated into the presented skeleton by using LTL-formulations. For each formula corresponding cost-maps is obtained. This enables the user not only to express instructions in a natural way and in return getting modified maps that respect these instructions but also state a preference. If the user provides a set of instructions the converter considers all of them and derives the corresponding maps from which the user then can chose.

It is important to note that the operators introduced before have certain strength, henceforth the introduced brackets are of great importance. Due to the setup of this converter, the strongest binding operator is the '$\square\neg$'- operator. Everything within its input gets threated with the highest priority and is modified first. Once a cell was modified by the $\square\neg$-operator it cannot be overwritten by the entries of any other operator.

The black area within the maps represents occupied areas or areas of high cost while the white one represents free areas. They grey areas are of low cost. Note that the resulting maps are generated with and without usage of the safe-mode option. This depends on the preferences of the latter user and the task. Nevertheless is the safe-mode a valid option, it can be used for safety-reasons. A first conclusion based on the above performed experiments is, that the proof of concept was successful. Hence given a labeled map and an LTL-formula entered via the skeleton, the framework generates respective cost maps that can be used to steer a path planning algorithm. However there are limitations to this framework depending on the provided maps and formulas. The environment is assumed to be completely known and an immaculate map is provided.

Translating the natural language instructions into the LTL-instructions that are subject to the input of the skeleton relies within the user's responsibility. The resulting output is a direct translation of the users input. So in terms of the converter, there are no incorrect input formulas. However it is recommended to specify each label within the formula or to use the safe-mode to ensure the expected results.

5.2 Set-up and Realization for Framework- Experiments in ROS

In here the set-up and implementation of the complete framework will be described. This includes the input-node, the implementation of the search-algorithm, the clarification request and how the path is executed. We use the ROS-environment including the Gazebo-Simulator and RVIZ for visualization.

5.2.1 Overview of Framework

An overview of the complete framework is illustrated in figure 27. It can be seen that the map-converter only requires an LTL-formula via the skeleton and a labeled map to create cost maps. This structure makes it independent of system specifications and hence its application is not limited to only this framework.

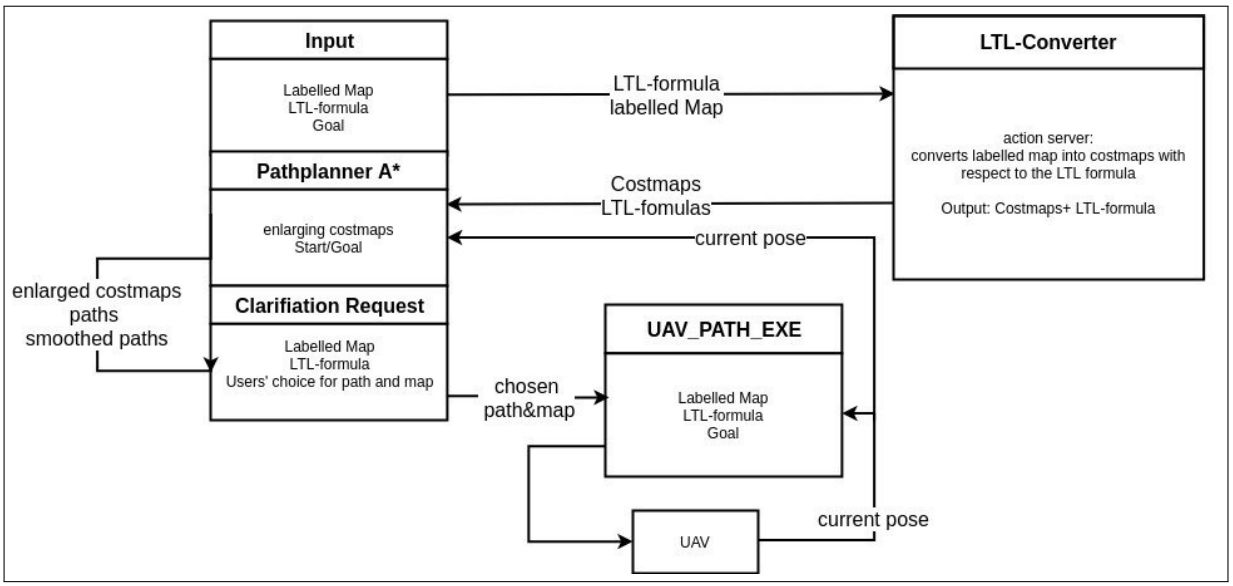


Figure 27: ROS-Structure of the Framework

5.2.2 The World Representation

To perform experiments a representation of the environment is needed. Such a representation is given by a grid map. Therefore a 2-dimensional labeled grid map is created manually. The map is separated into areas of different values such that they can be addressed with respect to their values.

The illustration in figure 28 shows areas of different colors that represent different values. Areas that are black represent occupied space and are addresses with $A10$. The values of the other areas are assigned such that the map is divided in different areas that can be addressed by them. The corresponding Rviz-representation of this map can be seen in figure 29. Note that in there are no coordinates nor enumerations displayed. Therefore we will further refer to the map illustrated in figure 28. However as the simulation employs Rviz the effects of the path planning and respective adjustments are based on this map.

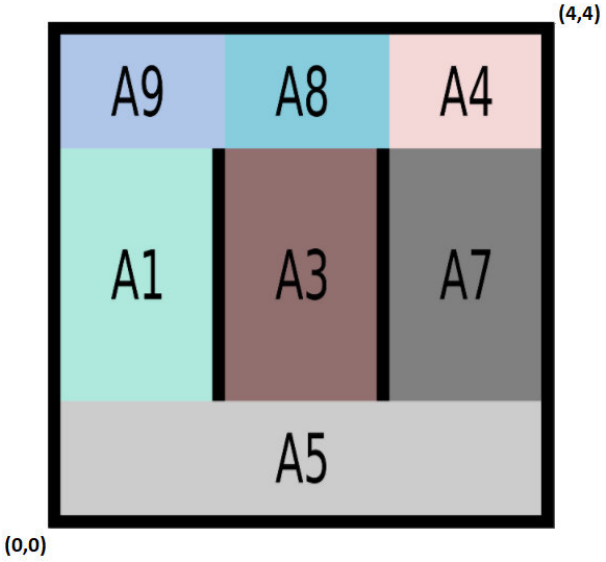


Figure 28: By Areas Labeled Map of the Environment

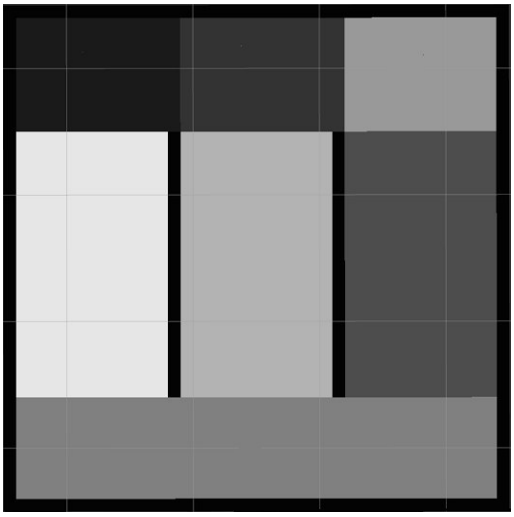


Figure 29: Labeled-by-color map displayed in Rviz

5.2.3 The Input Node

The input node serves as a user interface, based on the skeleton. The user puts in the LTL-constraints, the start and the goal as well as the specification regarding the safe-mode. The labeled map of the environment is contained in the input node too and can be adjusted if needed. It is the user's responsibility to ensure the feasibility of the task, that is the user must reason if the constraints lead to the expected result. It is possible though that the map conversion is executed but path planning was not successful. In that case the user will receive a notification.

The question why the goal cannot be entered directly into the $\square\lozenge$-array might appear. The reason for this is that in the *eventually-always* operator only the region of interest is defined. This region can be shaped in any way and does not have to be necessarily connected as can be seen in 30. If one then would assign the task that the UAV should reach *eventually-always* [yellow], it is not clear how to interpret it. It could either be interpreted to choose the grid cell with the lowest grid number, in which case the final position would depend on the enumeration of the cells. Another way would be to calculate the center of gravity of the area, which again might lead to a point outside the actual region of interest as can be seen in figure 30. For the blue U-shaped area the center of gravity lies outside the actual area. Hence the goal of the path would end up somewhere in the red area, which could be in an obstacle. As this question needs further investigation it is suggested as future work and the goal area has to be entered separately to ensure the path ends up at the expected position.

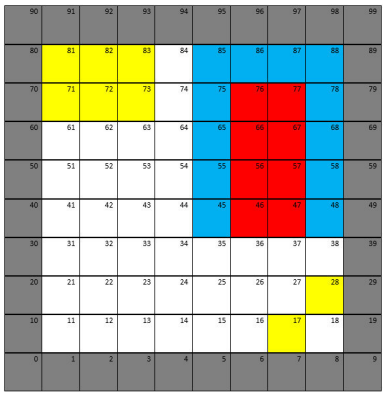


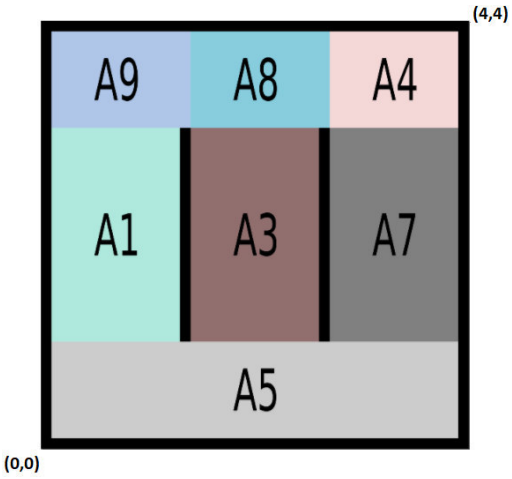
Figure 30: Colored Grid Map with U-Shaped and Disjunct Region of Interest

5.2.4 Path Planning using A^*

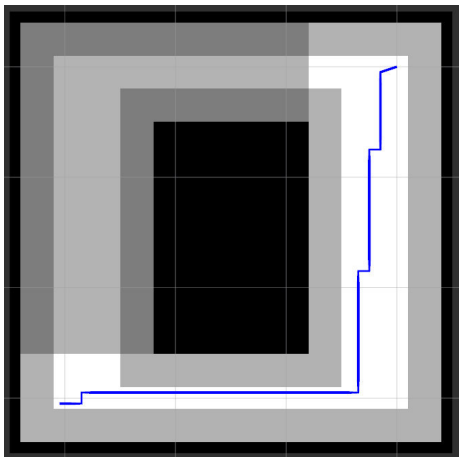
In this section the implementation of the A*-path planning algorithm into the framework is described. As already mentioned in the preliminaries, the A*-algorithm is the method of choice due to its completeness and optimality, meaning if there exists a path, it finds it. It planes the shortest path through an environment given a start and a goal position by evaluating the provided cost maps with respect to the underlying heuristic. The chosen heuristic in here is the Manhattan-distance. The two following examples show how the A*star algorithm combined with the LTL-map-converter creates a path through the newly obtained cost maps.
The task here is given by: 'Go from start position(0.5,0.5) to the goal position in $A4$ (3.5,3.5). Do not pass through $A3$ and $A10$, prefer $A5$ and $A7$ and eventually reach $A4$'. This can be translated into the skeleton:

$$\Gamma = [\square\neg[A10 \wedge A3] \wedge \blacklozenge[A5 \wedge A7]] \wedge \lozenge\square[A4] \tag{29}$$

Note the goal has to be within the area of interest, that is $\lozenge\square[A4]$.



(a) Labeled Input Map



(b) Resulting Cost Map for Equation 29 and Path

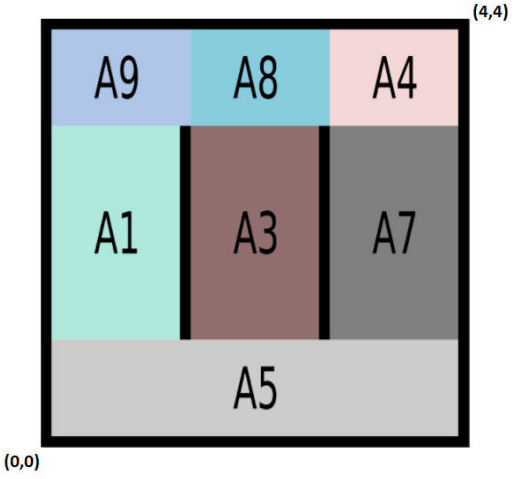
Figure 31: Resulting Path for the A^* -algorithm for the LTL-formula 29

Figure 31 shows the converted cost map according to the skeleton-input 29. The black areas represent areas of a high cost, such as occupied or forbidden areas while the white area is of zero cost. The grey areas are of low cost. The A*-algorithm finds the shortest/cheapest path with respect to its heuristic.

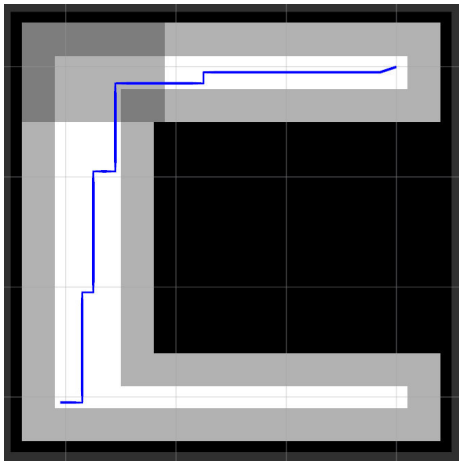
Now given the same start and goal position but a modified LTL-formula

$$\varphi = [\Box\neg[A10 \wedge A3 \wedge A7] \wedge \blacklozenge[A5 \wedge A1 \wedge A8] \wedge \Diamond\Box[A4] \tag{30}$$

the resulting cost map and respective path can be seen in figure 32.



(a) Labeled Input Map



(b) Resulting Cost Map for Equation 30 with Path

Figure 32: Resulting Path for the A^*-algorithm for the LTL-formula 30

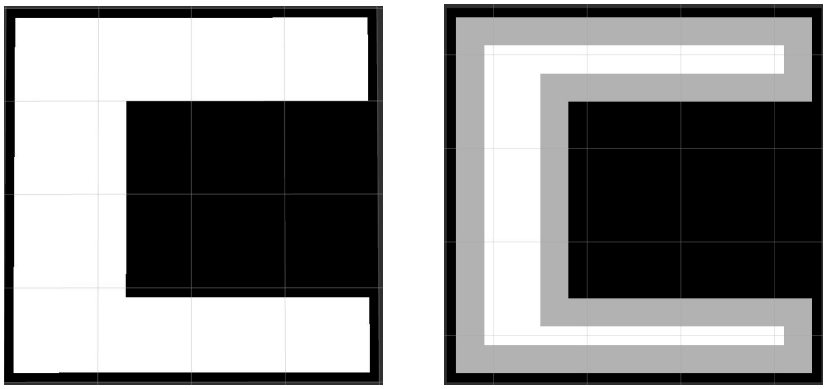
As mentioned before the Manhattan-distance is used, meaning the resulting path can only traverse vertically or horizontally but not diagonal. This leads to edgy paths as can be seen in figures 31a and 31b, that can be difficult to follow for a UAV.
Note that the slightly diagonal part of the path close to the goal position, occurs due to the discretization. The grid size here is 10cm leading to a cell size of 10-by-10cm and the algorithm treats each way point to be in the middle of a grid cell. The goal position however is an absolute position and the algorithm aims to exactly arrive at it, so its last part close the goal becomes diagonal regardless of the heuristics.

5.2.5 Adjustments for the Path

Before performing simulations within the Gazebo simulator or in the SML, the path is adjusted such that it is more suitable for a UAV. Therefore the two adjustments are included:

- Obstacle enlargement
- Path smoothing

For the obstacle enlargement we assume that the UAV has the size of a point-mass (here the size of one grid cell respectively). Furthermore we assume that an occupied cell has the original size of the UAV. Then before feeding a cost map into the path planning-algorithm each grid cell in the map is checked for its value. If the value indicates that a cell is not occupied it moves on to the next one. However if the value of a cell indicates that it is occupied the enlargement-function comes to act. That is, the cells adjacent to the center cell (the one that is currently checked) are assigned the value for occupied areas. It is furthermore assumed that the UAV is centered above the currently checked cell. The number of modified adjacent cells depends on the size of the UAV.



(a) Costmap after Map Conversion

(b) Enlarged Cost Map with Respect to the Size of the UAV

Figure 33: Cost Map and Corresponding Enlarged Cost Map

Figure 33(a) shows a cost map and figure 33(b) the respective enlarged map, where each occupied cell (here illustrated in black) was enlarged with respect to the size of the UAV. The grey area represents the enlarged obstacles.

The obtained enlarged cost-map is then fed into the path planning algorithm. Given a start position and the goal area, the obtained path can be seen in figure 35(a). The blue path uses the

Manhattan-distance to traverse through the map and is edgy compared to the green one. Such a path with many turns and edges can be time-and energy consuming, therefore the path is smoothed by the so called Line-of-sight-method resulting in the green path in figure 35(b). The method checks whether a way point and its second successors can be connected by a straight line. Obstacles interrupt that line-of sight. That is, if there is line of sight from the way point i to point i+2, remove point i+1, otherwise keep it. Repeat this until there is no line of sight between adjacent points in the path. This method results a reduced number of way points as can be seen in figure 34.

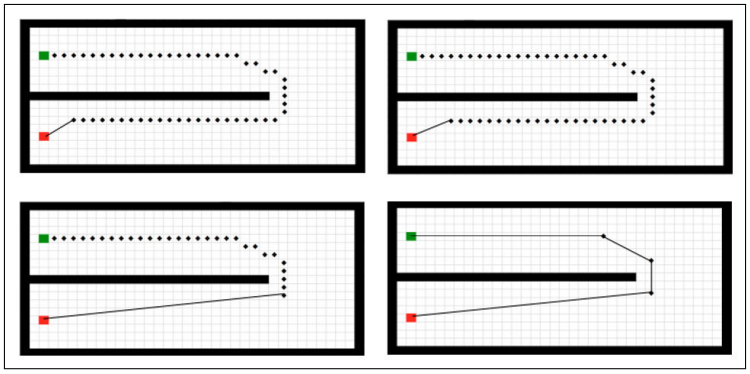
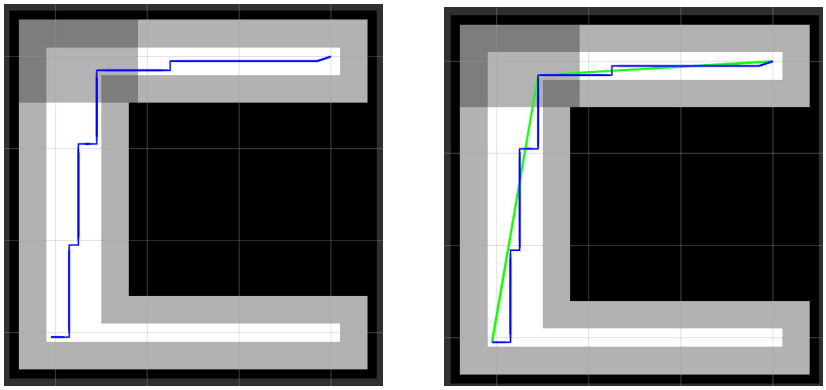


Figure 34: Line-of-Sight Path Smoothing, excerpt from [3]



(a) Path in the Enlarged Cost Map

(b) Obtained Path (blue) and Smoothed Path (green) in Enlarged Map

Figure 35: Path and by Line-Of-Sight-Smoothed Path in Enlarged Map

Figure 35 shows the effect of the applied smoothing technique. The blue path displays the expected output of the A*-algorithm which is hard to follow for a UAV. However the green path shows the smoothed one which also reduced the number of way points for the resulting path. Note that the path is not completely smooth as this would mean there are no edges but the number of edges is significantly reduced. The green path with its direct links averts multiple turns of the UAV which saves time and energy. Nevertheless smoothing is not necessary but it can facilitate following the path depending on the controller. Note that the path planning algorithm itself is not aware of

the smoothing as it is performed subsequently.

5.2.6 The Clarification Request

The LTL-converter creates a number of cost-maps, depending on the LTL-formula, the provided labeled map and the number of goals. As described in chapter 4 the number of resulting maps is given by $n_{rows}\Box\neg[...] \cdot n_{rows}\blacklozenge[...] \cdot n_{rows}\Diamond\Box[...]$. Therefore in the case of multiple solutions in form of cost-maps and corresponding paths, a clarification request is issued. This request asks the user to choose a map as can be seen in figure 36. The LTL-formulas, the total number of maps and paths will be displayed and the user will be asked to choose, see figure 42

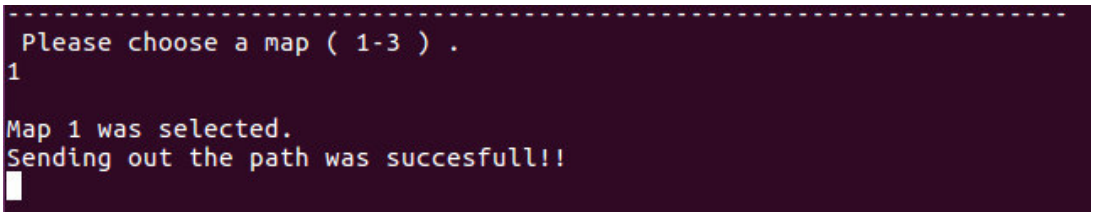


Figure 36: Clarification Request and Confirmation for the case of three maps and paths

This clarification request resembles the initially described scenario in chapter 1, if two regions of interest are found and the UAV must issue the clarification request for the operator to confirm which path to follow.

5.3 Framework Experiment in Simulation

In this section the final simulation experiment is presented. It combines all previously introduced components. The provided map of the environment remains the same as before, see figure 37. Based on this map a model-world within the Gazebo-simulation environment is created. Figure 38 shows the top-down view on a UAV inside a Gazebo-world matching the map of the environment. The start position is at $(0.5, 0.5)$ and the goal position at $(3.5, 3.5)$. The task is given by: '*End-up in* A4, *prefer* either $A1$ or $A3$ or $A7$ and *avoid* $A10$.' This can be represented by an LTL-formula:

$$\varphi = \varphi_1 \vee \varphi_2 \vee \varphi_3 \tag{31}$$

where

$$\varphi_1 = \square\neg A10 \wedge \blacklozenge A1 \wedge \lozenge\square A4 \tag{32}$$

and

$$\varphi_2 = \square\neg A10 \wedge \blacklozenge A3 \wedge \lozenge\square A4 \tag{33}$$

and

$$\varphi_3 = \square\neg A10 \wedge \blacklozenge A7 \wedge \lozenge\square A4 \tag{34}$$

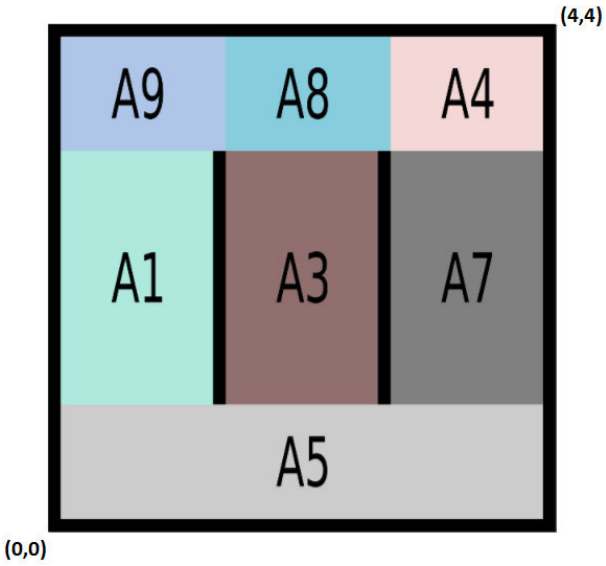


Figure 37: Labeled Map of the Environment

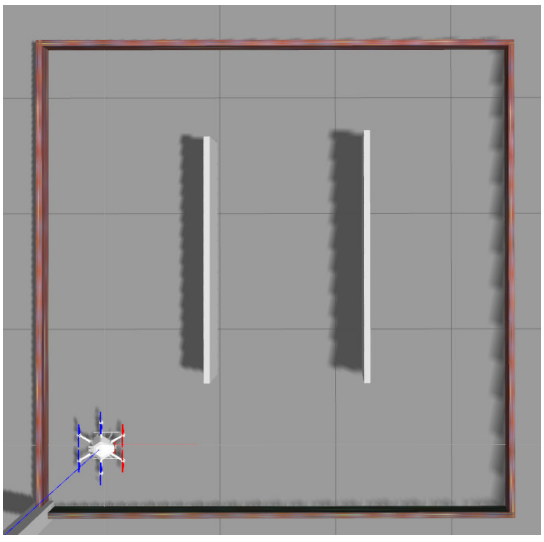


Figure 38: Top-down view of the Gazebo world with UAV at initial pose (0.5,0.5)

Translating this task into the skeleton results

$$\Gamma_\wedge = \left[\square\neg\ \left[A10\right] \wedge \blacklozenge \begin{bmatrix} A1 \\ A3 \\ A7 \end{bmatrix}\right] \wedge \lozenge\square\left[A4\right] \tag{35}$$

The formula 35 leads a total of three resulting cost maps, as there are three rows within the weak-eventually operator and one row within the not-operator and one within the eventually-always-operator. After having entered the LTL-formula into the frame work the user will get an output

on the screen displaying each one of the respective LTL-formulas. Furthermore it states which map belongs to which formula as well as the p-count as a preference measure, as can be seen in figure 42. The white areas represent the preferred areas which are subject to the *weak-eventually*-array and the *eventually-always*-array. The safe-mode is not active, meaning areas not specified such as $A2$, $A5$, $A8$ and$A9$ are assigned a low cost.

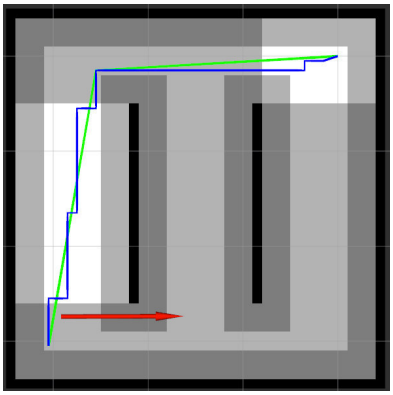


Figure 39: Path and Smoothed Path for formula 32

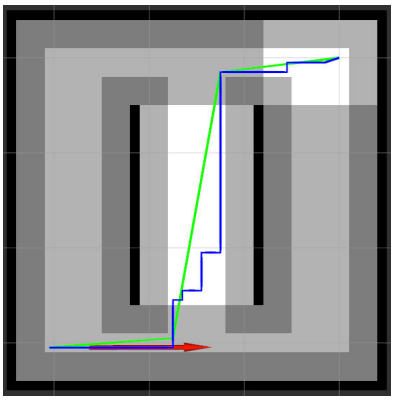


Figure 40: Path and Smoothed Path for formula 33

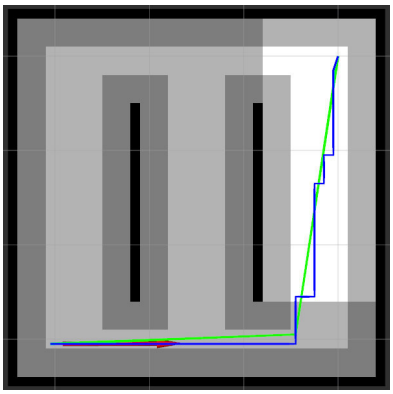


Figure 41: Path and Smoothed Path for formula 34

This summarized information and the corresponding maps including paths that are obtained enable the user to choose its preferred path In the terminal the clarification request is displayed including the corresponding LTL-formula and the total number of paths and maps for each option and it is up to user to choose. After the user has made its choice, the chosen path is executed, seen

```
++++++++++++++++++++Clarification request++++++++++++++++++++++++++++
------------------------------------------------------------------------
Number maps: 3
Number path: 3
------------------------------------------------------------------------
LTL-formula
Map 1 ALWAYS-NOT[ 99] WEAK-EVENTUALLY[ 10] EVENTUALLY-ALWAYS[ 40] SAFE:0 P-count: 1/1
Map 2 ALWAYS-NOT[ 99] WEAK-EVENTUALLY[ 30] EVENTUALLY-ALWAYS[ 40] SAFE:0 P-count: 1/1
Map 3 ALWAYS-NOT[ 99] WEAK-EVENTUALLY[ 70] EVENTUALLY-ALWAYS[ 40] SAFE:0 P-count: 1/1
------------------------------------------------------------------------
 Please choose a map ( 1-3 ) .
```

Figure 42: Issued Clarification Request

in figures 43 and 44. The p-count states of how many of the preferences are met, that is 1/1 in this example.

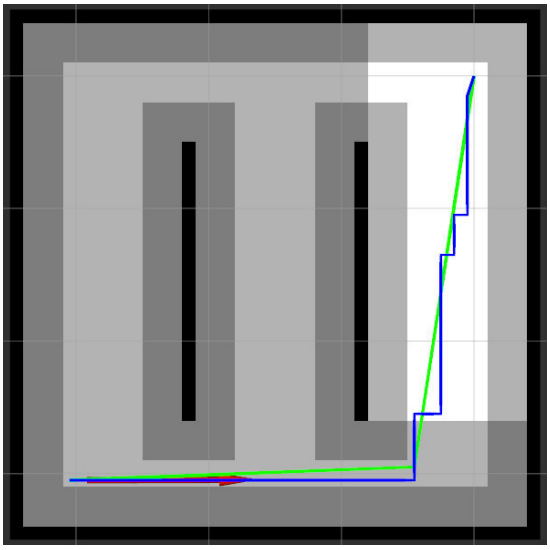


Figure 43: Chosen Map and Path after Clarification Request

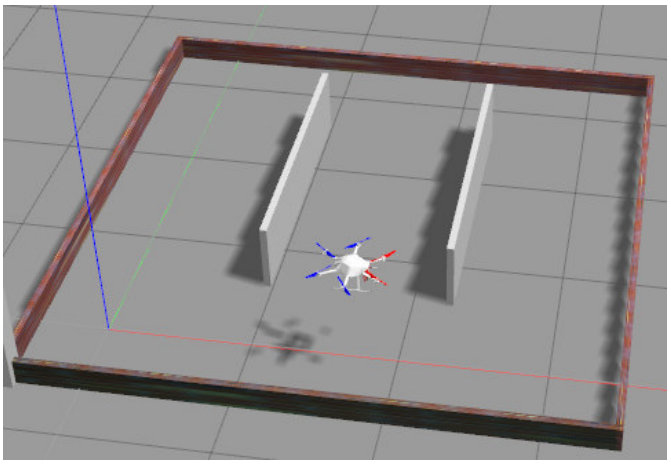


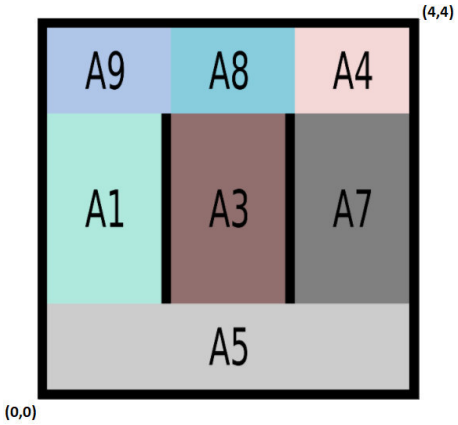
Figure 44: Gazebo Simulation of path execution with UAV

5.4 Framework- Experiment in SML

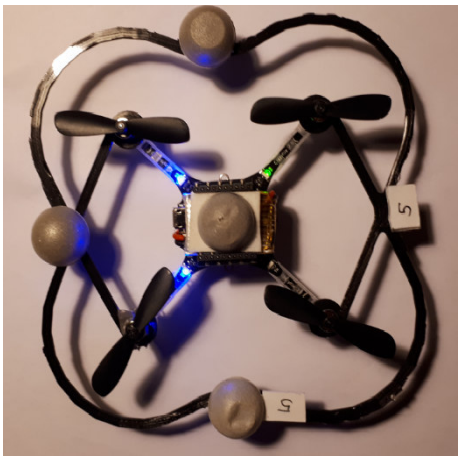
In order to validate the simulations, the same experiment is performed as a real world experiment in the Smart Mobility Lab (SML) at KTH.

5.4.1 Setup

The deployed UAV for this experiment is the Crazyflie 2.0 which can be seen in figure 45(b). Its technical specifications can be found in [20], [21] and [22]. The experimental environment is represented by the SML-arena where the labeled map 45(a) was projected onto the ground. The arena can be seen in figure 45(b) and is represented by a square-area of 4m-by-4m. The localization of the UAV is provided by the Qualisys Motion Capture System in the SML.



(a) Labeled map for the experiment in the SML

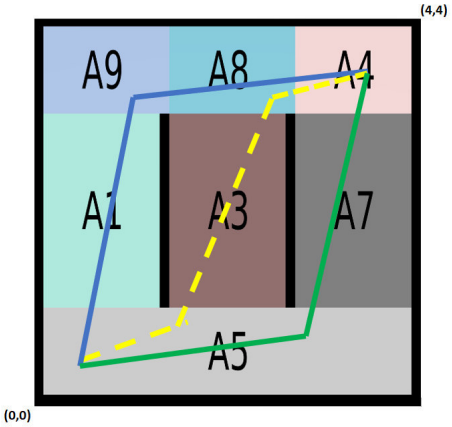


(b) The Crazyflie 2.0

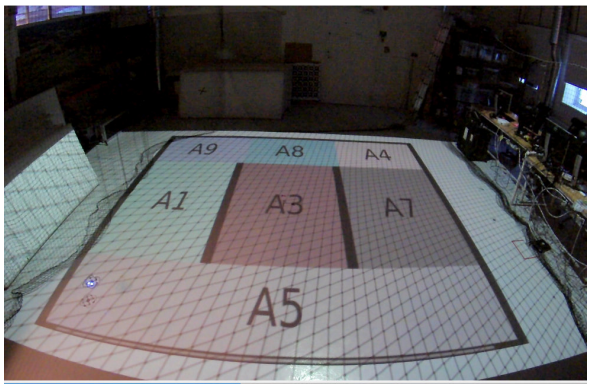
Figure 45: Labeled Map and Crazyflie 2.0

5.4.2 The Task

Given the map 45(a) the task is to find a path from the coordinates $(0.5, 0.5)$ to $(3.5, 3.5)$ whilst preferring either area $A1$ or $A3$ or $A7$. The obtained results are three different paths represented as waypoint-lists. The clarification request asks the user the choose which path should be executed. After the user has made a choice, the UAV traverses through the environment according to the the obtained path.



(a) Labeled map with three resulting paths



(b) The Crazyflie in the SML-Arena

Figure 46: Illustration of the three obtained paths and picture of path execution in the SML

This experiment was successfully repeated such that each of the three paths marked as green, yellow and blue in 46(a) was executed once. The path was selected via the clarification request. Figure 46(b) shows real world set-up in the arena of the SML. The labeled map of the environment is projected onto the ground. In the lower left corner is the Crazyflie ready to traverse through the environment on the obtained paths.

6 Discussion and Conclusion

In conclusion this work has shown that given a labeled map of an environment, start and goal position and constraints formulated in natural language represented by an LTL-formula, a framework for planning has been created. The mapping from LTL-constraints to physical motion is the first step towards the mapping from natural language to physical motion. We therefore take advantage of the resemblance between NL and LTL and first address the problem of mapping LTL-constraints to motion as stated in [11]. In order to have a standardized input form and a clear definition on what kind of formulas the framework can handle, an input-skeleton was created. This skeleton contains arrays for each temporal operator employed and handles them respectively. The newly defined operator, namely the *weak-eventually* allows the user to specify a preference and contemporaneously ensures that it's part of the formula is always true. The p-count that comes with it measures the satisfaction of the respective constraints. The map-converter benefits from this skeleton as it is subject to its input. Henceforth the map conversion is executed based on the users input and can result in obtaining multiple cost-maps. Experiments to evaluate the performance of the skeleton and the map-conversion were performed and concluded successful.

Based on the obtained results, a grid-map based path planning algorithm was incorporated. For the sake of performance regarding speed A^* was the method of choice. Its performance is fast, such that it can be applied to any obtained cost map. The beneficial effect the *weak-eventually* can be seen in here, as it allows the user to steer path according to the preferences of the user. Hence it is possible to obtain paths that would not exist, since they are not the shortest/cheapest from start to goal position. But the cheapest by taking the user's preference via the *weak-eventually* into account.

Note that for visibility reasons in RVIZ, the framework was tested for handling up to sixteen maps and obtaining the respective paths and no lack in performance was detected. However, this depends on the performance of the used workstation.

In the next step a clarification request was constructed, enabling the user to choose its preferred set of cost-map and path. As shown in this report, the clarification request combined with the map converter provides the user with the option to choose the preferred solution from all obtained cost-map-path pairs. The final simulation experiments have clearly shown that the framework works as expected. These results are confirmed by the real-world experiment performed at the SML.

One advantage of this approach is that the user himself specifies the constraints by entering them into the skeleton. By taking advantage of the *weak-eventually*-operator the user's preference is taken into account. The introduced preference measure provides a direct evaluation of the satisfaction of those for each obtained path. The clarification request allows the user to verify and specify the task. There is no fix pre-set-up on which areas get assigned a high cost. Therefore one could also use this approach to invert a map like figure 37 and obtain a cost map where the UAV is only allowed to stay in the black area. This could be used for example to design a border-patrolling task.

Another advantage of this framework is that it can be applied to any kind of autonomous system under the assumption that start and- goal pose, labeled map of the environment and a task formulation represented by an LTL-formula is given. That is, it is independent of the platform. The method of choice regarding the path-planning is the A*-algorithm, but the framework is not limited to this algorithm. It is possible to exchange the path-planning algorithm to any other cost map-based planner. The presented solution benefited from this algorithm as the planning itself was performed fast for every obtained cost-map including the path adjustments such as obstacle enlargement and

path smoothing. The number of possible entries which the skeleton can handle depends on the available computational power. The current framework is limited to 2d-maps as the provided input map is as well. The map converter itself should be able to handle 3d-maps but the availability and implementation of a 2.5d-planner or 3d-planner is very limited.

7 Future Work

One way to extend the work presented in here would be to apply it to a multi-agent-system. For example on a larger-scale environment. This is therefore reasonable as the presented converter generates multiple cost-maps and corresponding paths depending on the input-formula. Therefore instead of only using one map-path-pair, all that are of interest to the user could be used. Hence, given a labeled map of an environment and formulating a task as it was done before, the clarification request could be extended, enabling the user to assign each set (map and path) of interest to one agent. This assigning could either be done by manually or automatically depending on the task and the users' preferences. One challenge would be to avoid collisions. This could be done by assigning either different levels of heights or having on-board-collision avoidance controllers. This extension allows to cover more ground based on the same information, which considered the initial problem of Human-Robot-Collaborative-Search-Missions could be beneficial.

Another possible extension would be to extend the map-converter with an exploring algorithm. That is, sending the UAV to a goal pose and then switching to an exploring algorithm if a certain area should be investigated further.
A third aspect would be to take the time aspect into account, meaning a task has to be executable within a fixed time frame.

References

[1] Christel Baier and Joost-Pieter Katoen. *Principles Of Model Checking*, volume 950. 2008. ISBN 9780262026499. doi: 10.1093/comjnl/bxp025. URL http://mitpress.mit.edu/books/principles-model-checking.

[2] Peter Russel, Stuart; Norvey. *Artificial intelligence: a modern approach*. 2009. ISBN 0136042597. doi: 10.1017/S0269888900007724. URL http://web.cecs.pdx.edu/{~}mperkows/CLASS{_}479/2017{_}ZZ{_}00/02{_}{_}GOOD{_}Russel=Norvig=ArtificialIntelligenceAModernApproach(3rdEdition).pdf.

[3] Patric Jensfelt. Dd 2425, robotics and autonomous systems, slides on navigation.

[4] Roomba vacuum cleaner. http://www.irobot.com/global/sv/roomba_range.aspx(2013). Accessed: 2018-13-01.

[5] Rethink robotics. ethinkrobotics.http://www.rethinkrobotics.com/(2013). Accessed: 2018-13-03.

[6] Pepper robot. https://www.ald.softbankrobotics.com/en/robots/pepper/find-out-more-about-pepper120418. Accessed: 2018-13-03.

[7] Jana Tumova. Control strategy synthesis under infeasible goals: A maximally-satisfying approach. Presentation, KTH, 2015.

[8] Meng Guo, Karl H. Johansson, and Dimos V. Dimarogonas. Revising motion planning under Linear Temporal Logic specifications in partially known workspaces. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 5025–5032, 2013. ISSN 10504729. doi: 10.1109/ICRA.2013.6631295.

[9] Meng Guo and Dimos V. Dimarogonas. Reconfiguration in motion planning of single- and multi-agent systems under infeasible local ltl specifications. pages 2758–2763, 12 2013.

[10] Alphan Ulusoy, Stephen L. Smith, Xu Chu Ding, and Calin Belta. Robust multi-robot optimal path planning with temporal logic constraints. 02 2012.

[11] Georgios Fainekos, Hadas Kress-Gazit, and George Pappas. Hybrid controllers for path planning: A temporal logic approach. 2005:4885 – 4890, 01 2006.

[12] Stephen L. Smith, Jana Tumova, Calin Belta, and Daniela Rus. Optimal path planning for surveillance with temporal-logic constraints*. 30:1695–1708, 12 2011.

[13] Gobinda G. Chowdhury. Natural language processing. *Annual Review of Information Science and Technology*, 37(1):51–89. doi: 10.1002/aris.1440370103. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/aris.1440370103.

[14] From natural language to ltl. https://www.cs.uic.edu/~shatz/SEES/gunter.paper.pdf. Accessed: 2018-02-01.

[15] Peter Russel, Stuart; Norvey. *Artificial intelligence: a modern approach*. 2009. ISBN 0136042597. doi: 10.1017/S0269888900007724. URL http://web.cecs.pdx.edu/{~}mperkows/CLASS{_}479/2017{_}ZZ{_}00/02{_}{_}GOOD{_}Russel=Norvig=ArtificialIntelligenceAModernApproach(3rdEdition).pdf.

[16] Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from temporal logic specifications. In *IEEE Transactions on Automatic Control*, 2008. ISBN 3-540-33170-0. doi: 10.1109/TAC.2007.914952.

[17] Map representations for path planning. http://theory.stanford.edu/~amitp/GameProgramming/MapRepresentations.html, . Accessed: 2018-13-03.

[18] A star comparison. http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html, . Accessed: 2018-02-01.

[19] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[20] El2425 drone search and rescue. https://github.com/EL2425/Drone_Project_2017, . Accessed: 2018-02-01.

[21] Bitcraze. https://github.com/bitcraze. Accessed: 2018-02-01.

[22] Crazyflie ros. https://github.com/whoenig/crazyflie_ros7, . Accessed: 2018-02-01.

TRITA EECS-EX-2018:99
ISSN 1653-5146