

# Developing High-level Behaviours for the Boston Dynamics Spot Using Automated Planning

---

*Utveckling av högnivåbeteenden för Boston Dynamics Spot med  
hjälp av automatisk planering*

**Nisa Andersson**

Supervisor : Mariusz Wzorek  
Examiner : Cyrille Berger

## Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>.

## Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: <http://www.ep.liu.se/>.

## **Abstract**

Over the years, the Artificial Intelligence and Integrated Computer Systems (AIICS) Division at Linköping University has developed a high-level architecture for collaborative robotic systems that includes a delegation system capable of defining complex missions to be executed by a team of agents. This architecture has been used as a part of a research arena for developing and improving public safety and security using ground, aerial, surface and underwater robotic systems. Recently, the division decided to purchase a Boston Dynamics Spot robot to further progress into the public safety and security research area. The robot has a robotic arm and navigation functionalities such as map building, motion planning, and obstacle avoidance. This thesis investigates how the Boston Dynamics Spot robot can be integrated into the high-level architecture for collaborative robotic systems from the AIICS division. Additionally, how the robot's functionalities can be extended so that it is capable of determining which actions it should take to achieve high-level behaviours considering its capabilities and current state. In this context, higher-level behaviours include picking up and delivering first aid supplies, which can be beneficial in specific emergency situations. The study was divided and done in an iterative approach.

The system was tested in various scenarios that represent its intended future use. The result demonstrated the robot's ability to plan and accomplish the desired high-level behaviours. However, there were instances when achieving the desired behaviours proved challenging due to various limiting factors, including limitations posed by the robot's internal controller.

# Acknowledgments

First, I want to express my gratitude to Mariusz Wzorek, my supervisor, who not only provided me with the initial idea for this thesis but also offered invaluable guidance and support throughout the challenging and sometimes confusing testing phase.

Secondly, I am very grateful to my unofficial supervisor, Piotr Rudol, for your help, guidance and patience during the entire thesis. This thesis would have been even more challenging and less enjoyable without your insights and help with the implementation. I would also like to extend my thanks to Cyrill Berger, my examiner, for his valuable advice and constructive feedback during the course of this thesis.

Furthermore, I would like to acknowledge and express my deep gratitude to the AIICS division for giving me the opportunity to test the full implementation in the remarkable setting of Gränsö as part of the Wallenberg AI, Autonomous Systems and Software Program (WASP). The presence of innovative people and the delicious food make it a truly memorable experience.

Lastly, a special thanks to Filip Schwerger, my partner, for all the support and encouragement given to me when it gets tough.

# Contents

<b>Abstract</b>	<b>iii</b>
<b>Acknowledgments</b>	<b>iv</b>
<b>Contents</b>	<b>v</b>
<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>ix</b>
<b>1 Introduction</b>	<b>2</b>
1.1 Motivation . . . . .	2
1.2 Aim . . . . .	3
1.3 Research Questions . . . . .	3
1.4 Delimitations . . . . .	3
<b>2 Background</b>	<b>4</b>
2.1 Task Specification Tree . . . . .	4
2.2 The Delegation System . . . . .	7
2.3 ROS/ROS2 . . . . .	10
2.4 Automated Planner . . . . .	11
2.5 Vicon System . . . . .	11
<b>3 Boston Dynamics Spot</b>	<b>12</b>
3.1 General Description . . . . .	12
3.2 Communication with Spot . . . . .	12
3.3 Network Compute Bridge . . . . .	14
3.4 GraphNav . . . . .	15
3.5 Coordinate Frames in the Spot World . . . . .	16
<b>4 Theory and related work</b>	<b>18</b>
4.1 Coordinate Frame Transformations . . . . .	18
4.2 Task Specification Tree and Automated Planner in Collaborative Robotic Systems	20
4.3 Other Task Specification Languages . . . . .	21
4.4 Automated Planning . . . . .	21
<b>5 Method</b>	<b>23</b>
5.1 Pre-study . . . . .	23
5.2 The Implementation of Subsystems . . . . .	23
5.3 Experimental Scenarios . . . . .	29
<b>6 Results</b>	<b>31</b>
6.1 Overview of the Subsystem Implementation . . . . .	31
6.2 Experimental Evaluations . . . . .	37

<b>7</b>	<b>Discussion</b>	<b>41</b>
7.1	Results . . . . .	41
7.2	Method . . . . .	43
7.3	The Work in a Wider Context . . . . .	44
<b>8</b>	<b>Conclusion</b>	<b>45</b>
<b>A</b>	<b>Domain Definitions</b>	<b>47</b>
A.1	Initial Domain . . . . .	47
A.2	Final Domain . . . . .	51
	<b>Bibliography</b>	<b>55</b>

# List of Figures

1.1	The Boston Dynamics Spot robot acquired by the AIICS division . . . . .	3
2.1	An example of a TST in a collaborative mission . . . . .	6
2.2	An overview of the delegation process . . . . .	8
2.3	An illustration of a delegation module associated with each agent . . . . .	8
3.1	Images of the Spot robot . . . . .	13
3.2	An illustration of how the SDK abstracts the use of gRPC . . . . .	14
3.3	An overview of the communication between the robot and the external computation server . . . . .	15
3.4	An illustration of Spot's body frame . . . . .	17
4.1	An example scenario with two coordinate frames, the reference coordinate frame (W) and the fiducial coordinate frame (F) . . . . .	18
5.1	An overview of the implementation of the robot's navigation . . . . .	24
5.2	An example of the environment setup . . . . .	24
5.3	AprilTag setup . . . . .	25
5.4	An overview outlining the steps in the implementation for finding and picking up an object, in this case a first-aid kit . . . . .	26
5.5	An overview of how the delivering behaviour was implemented . . . . .	27
5.6	An overview of commanding the arm to move along a trajectory process (e.g. to implement <i>pick-up</i> and store object in a container mounted on the Spot robot) . . .	27
5.7	An illustration of the robot command request formation for commanding the arm to follow a trajectory . . . . .	27
5.8	An illustration of the SE3TrajectoryPoint(s) defined in the action that adds a picked-up object to the container on the robot's body frame (z-axis point out from the page). $p_1, p_2, p_3$ , and $p_4$ represent positions, and the yellow triangles represent the orientation of the gripper relative to its body. The blue circle represents the container on the body . . . . .	28
6.1	An overview of the subsystems implemented on the Spot robot . . . . .	31
6.2	The TST for the action <i>dock</i> which mean docking the robot to its dock-station . . .	33
6.3	The TST for the action <i>localise</i> which localise the robot . . . . .	33
6.4	The TST for the action <i>deliver-package</i> which delivers a package present in the robot's gripper . . . . .	33
6.5	The robot's perception of the AprilTags' poses before the robot has been localised. A tag number identifies each AprilTag, and the origins for both coordinate frames are denoted by <i>world</i> and <i>odom</i> . The red arrows represent the true pose of each AprilTag . . . . .	34

6.6	The robot's perception of the AprilTags' poses after the robot has been localised. An ID number identifies each AprilTag, and the origins for both coordinate frames are denoted by <i>world</i> and <i>odom</i> . The red arrows represent the true pose of each AprilTag . . . . .	34
6.7	A map created from the <i>Autowalk</i> feature where the rectangles with black and white patterns are the AprilTags. The number below each AprilTag denotes its tag number . . . . .	35
6.8	The Spot robot's localisation in comparison to GPS sensor readings. The rectangles represent the AprilTags, and the numbers under them are the tag numbers . . . . .	35
6.9	Difference between the GPS and the robot's localisation . . . . .	36
6.10	The Spot robot's localisation during the test scenario 1. The rectangles represent the AprilTags (see Figure 6.7 for a comparison), and the numbers under them are the tag numbers (527 is the AprilTag located at the dock station). The other numbers correspond to the starting point for each planner's actions in Table 6.1 . . . . .	37
6.11	The Spot robot's localisation during the test scenario 2. The rectangles represent the AprilTags (see Figure 6.7 for a comparison); the numbers adjacent to them are the tag numbers (527 is the AprilTag located at the dock station). The other numbers correspond to the starting point for each planner's action in Table 6.2 . . . . .	38
6.12	The Spot robot's localisation during the test scenario 3. The rectangles represent the AprilTags (see Figure 6.7 for a comparison); the numbers adjacent to them are the tag numbers (527 is the AprilTag located at the dock station). The other numbers correspond to the starting point for each planner's action in Table 6.3 . . . . .	39

# List of Tables

6.1	The actions generated by the planner for test scenario 1, along with their respective start- and end times during the execution . . . . .	38
6.2	The actions generated by the planner for test scenario 2, along with their respective start- and end times during the execution . . . . .	39
6.3	The actions generated by the planner for test scenario 3, along with their respective start- and end times during the execution . . . . .	40



# Abbreviations

**AIICS** Artificial Intelligence and Integrated Computer Systems.

**API** Application Programming Interface.

**BT** Behavior Tree.

**gRPC** Google Remote Procedure Call.

**IMU** Inertial Measurement Unit.

**LiDAR** Light Detection and Ranging.

**PDDL** Planning Domain Definition Language.

**PlanSys2** ROS2 Planning System.

**POPF** Partial Order Planning Forwards.

**ROS** Robot Operating System.

**RPC** Remote Procedure Call.

**SDK** Software Development Kit.

**STRIPS** STanford Research Institute Problem Solver.

**TAL** Temporal Action Logic.

**TALF** Temporal Action Logic Fixpoint.

**TFD** Temporal Fast Downward.

**ToF** Time-of-Flight.

**TST** Task Specification Tree.



# 1 Introduction

This chapter introduces the thesis by describing the problem from a general point of view, its underlying purpose, research questions and delimitations.

## 1.1 Motivation

Public safety organisations such as emergency and medical services operate differently today compared to the last decade due to technical advances. Ground, aerial, and underwater robotic systems with sophisticated commands and control systems are progressively being incorporated into the operations to help prevent and protect the general public from dangerous events. Unmanned aerial robotic systems such as drones have actively been used in hundreds of search and rescue operations [26].

As robotic systems become more autonomous and robust, they enable opportunities for multiple systems to collaborate in larger emergency scenarios [38]. For example, in the event of a natural disaster, drones can collaborate to identify and reach injured individuals, providing them with necessary supplies such as water, food, and medical kits. However, for the systems to be acceptable, practical and safe, the collaboration software must be based on a verifiable, principled and well-defined interaction foundation between the robotic systems and human operators.

The Artificial Intelligence and Integrated Computer Systems (AIICS) Division at the Department of Computer and Information Science, Linköping University has over the years developed a high-level software architecture for collaborative robotics that focuses on supporting collaboration between humans and heterogeneous robotic systems [1]. The developed framework, among its many functionalities, includes a delegation system that allows for defining and executing complex missions by teams of human and robotic agents. Recently, the division purchased a Boston Dynamics Spot robot equipped with a robotic arm (see Figure 1.1). This robot has navigation functionalities such as map building, motion planning, and obstacle avoidance. Moreover, it can operate on varying terrain including uneven or inaccessible areas, where they might be unsafe for humans and inconvenient for other robotic systems. Additionally, there are API (Application Programming Interface) services that enable developers to design customised controls and extend the robot's functionalities and tasks it can do. By integrating it into the architecture, the robotics systems will be able to collaborate with a broader range of tasks in emergency response scenarios.



Figure 1.1: The Boston Dynamics Spot robot acquired by the AIICS division

## 1.2 Aim

This thesis aims to investigate how the Boston Dynamics Spot robot can be integrated into the software framework for collaboration between humans and multiple robotic systems developed by the AIICS division. Moreover, how the robot can be programmed to accomplish complex behaviours, such as delivering a first-aid kit to an injured person. This will broaden the range of tasks that can be accomplished in scenarios associated with public safety domain.

## 1.3 Research Questions

The thesis aims to answer the following research questions:

1. How can the Boston Dynamics Spot robot be integrated into the collaborative robotics software framework developed at AIICS division?
2. How can an automated planner be used to achieve the following autonomous behaviours?
  - Navigate to a specific position, find a specific object (such as a first-aid kit), pick it up and move to another location to deliver that object.
  - Navigate to multiple positions and collect the object at each location.

## 1.4 Delimitations

The robot's navigation in this thesis is built upon localisation and mapping services from Boston Dynamics. This requires additional configuration to work as intended and will be further described in the thesis. The machine learning model used to detect objects such as first-aid kit is provided by the AIICS division.



## 2 Background

This chapter provides a comprehensive overview of the delegation system included in the software framework for collaboration between humans and multiple robotic systems, and the Task Specification Tree (TST) language used by it. In addition, this chapter also gives an introduction to the tools used in AIICS's delegation system and automated planner used in this study.

### 2.1 Task Specification Tree

TST, proposed in [12, 15], is a declarative representation for defining complex tasks for single and multi-agent systems. It is structured as a tree, with internal and external nodes representing control statements and elementary actions, respectively. Elementary actions are robotic platform-dependent and viewed as indivisible from an external perspective. However, they can be divided into composite actions from the robotic platform's point of view. For example, the action "move" from the TST node's perspective is indivisible, but the robot internally may have to first perform localisation, pathfinding, and other sub-actions.

#### Node types

The TST language offers several types of nodes and this section provides descriptions of them.

#### Control Nodes

Control nodes are nodes used to organise a set of actions. They include the following:

- **Sequence (S)**  
The Sequence node will execute its children in a sequence.
- **Concurrent (C)**  
The Concurrent node will execute its children concurrently.
- **Test-if (Test-if)**  
The Test-if node is a conditional branching node in the tree.

- **Select (Sel)**

The Select node tests each of its children until either one succeeds or all have been tested.

- **Loop (L)**

The Loop node will execute its sub-tree repeatedly until a termination condition becomes true.

- **Monitor (Monitor)**

The Monitor node uses one or more temporal logic formulas to be evaluated over a sequence of states and triggers sub-tasks in case the formulas become true. Temporal Logic is used for reasoning about time and temporal information [30] (compared to logic which is used for reasoning only), and representing them formally within a logical framework. This type of node is helpful for execution monitoring and failure recovery.

- **Try (Try)**

The Try node corresponds to the try-catch-throw mechanism supported by many programming languages. It helps to catch contingent problems and act on them.

### Interaction Nodes

Interaction nodes are used to handle interactions between an agent (a human or a robotic platform) and other agents. These include:

- **Do (Do)**

The Do node specifies a task for the human operator to perform.

- **Approval (Appr)**

The Approval node indicates that human approval is required before the execution can proceed.

- **Query (Query)**

The query node represents a query for other agents or human operators involved.

- **Goal (Goal)**

The Goal node contains a high-level goal to be achieved and can be expanded using an automated planner or a template. The expansion results in a sub-tree representing the generated plan, extending the original tree.

### Elementary Action Nodes

An elementary action node indicates that the action of this node is robotic platform dependent, and the implementation is platform specific.

### TST Example

Figure 2.1 shows an example of a TST in a collaborative mission allocated to different agents [1]: MO – Mission Operator, A - DJI Matrice 100, and B - Spot robot. The light blue circles are control nodes, and the green rectangles are elementary action nodes. In this example, there are two types of control nodes, sequence (S) and Test-if (test\_if) nodes.

In the example, the mission operator is responsible for a rescue mission, which has been expanded into sub-tasks and allocated to agents A and B. The tasks are to be executed in a sequence. First, agent A will scan an assigned geographical area to locate injured individuals and then, agent B will deliver a medicine kit to the identified person. Before agent A starts

executing the task, it first verifies whether it is already in the air. If not, it will take off and then proceed with the mission. However, if the agent is already in the air, it proceeds with the mission. This verification process is specified using a Test-if control node, which queries and then determines the next task based on whether the query result is true, false, or unknown.

Each node is associated with a time constraint, indicated by the red text specifying the minimum and maximum time duration for initiating and completing the task. However, it is also possible to add other types of constraints, like parameter constraints. For example, the elementary action "deliver\_medical" has parameter constraints that ensure the medical kit is delivered to a particular person.

Node constraints can constrain the parameters of the node's ancestors, express precedence relations and organise the relation between the nodes in the TST not captured by using control nodes. As a result, the constraints form a constraint network where the parameters of the node in a TST are constraint variables.

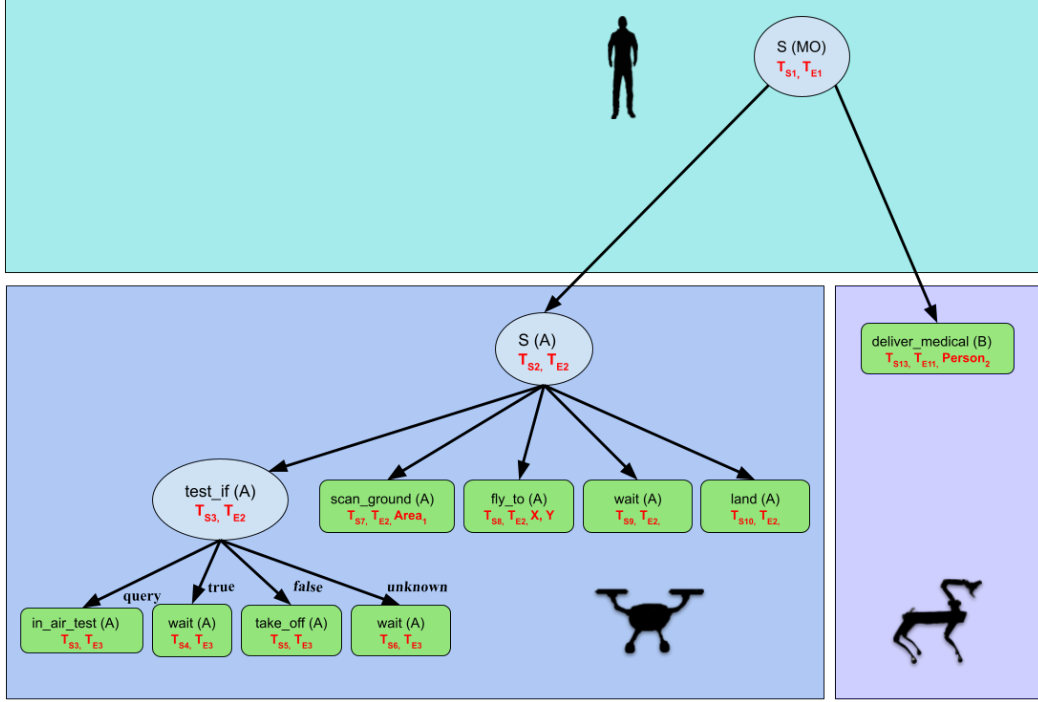


Figure 2.1: An example of a TST in a collaborative mission

### Formal Semantics

This section presents a formal task specification language that establishes the representation of TSTs. The formal semantics of temporal composite actions (actions that are composed of multiple sub-actions) representing missions are provided by Temporal Action Logic Fixpoint (TALF). TALF is an extended version of Temporal Action Logic (TAL) [10, 13], a highly expressive logic for defining and reasoning about actions and their effects. The language considers temporal aspects (involve aspects of time), concurrent execution, and incomplete knowledge about the environment, making it suitable for defining elementary actions. TAL has been proven to be extensible but cannot represent loops, recursion, and inductive recursion required for specifying composite actions that capture the high-level mission specification. For this reason, it is extended with fixpoint logic [2], increasing the expressivity but still allowing use of relatively efficient inference techniques. The syntax for temporal composite action (C-ACT) using TALF is defined as follows:

$$\begin{aligned}
\text{C-ACT} &:= [\tau, \tau'] \text{ with } \bar{x} \text{ do TASK where } \phi \\
\text{TASK} &:= [\tau, \tau'] \text{ ELEM-ACTION-TERM} \mid \\
&\quad [\tau, \tau'] \text{ COMP-ACTION-TERM} \mid \\
&\quad (\text{C-ACT}; \text{C-ACT}) \mid \\
&\quad (\text{C-ACT} \parallel \text{C-ACT}) \mid \\
&\quad \text{if } [\tau]\Psi \text{ then C-ACT else C-ACT} \mid \\
&\quad \text{while } [\tau]\Psi \text{ do C-ACT} \mid \\
&\quad \text{foreach } \bar{x} \text{ where } [\tau]\Psi \text{ do conc C-ACT}
\end{aligned}$$

where the time interval the task must be executed is denoted by  $[\tau, \tau']$ . The term “with  $[\bar{x}]$  do TASK where  $\phi$ ” denotes a composite action consisting of a task TASK that should be executed in a context characterised by a set of variables (possibly empty)  $\bar{x}$  and a group of constraints  $\Phi$ . The TASK can be an elementary action (ELEM-ACTION-TERM) or a combination of composite actions (COMP-ACTION-TERM) using constructs such as sequence ( $;$ ), concurrency ( $\parallel$ ), or conditionals, “while-do”, and a concurrent “foreach” operator. The expression  $[\tau]\Psi$  in the last three lines is a logical formula referring to facts at a single time-point  $\tau$  [11].

TALF provides the formal semantics for TSTs, and can directly be translated from or to composite actions in TALF. Any composite action within TALF, when combined with the additional axioms and circumscription policy, is linked to a collection of logical models. Since TALF is a temporal logic, each model represents a potential evolution of the world’s state over time, assuming the composite action is executed. Suppose the information given in the specification is correct. In that case, the trace from the execution of the corresponding TALF must correspond directly to a member of that set of models [11]. This provides formally grounded conclusions about the outcome if a particular TST is executed.

## 2.2 The Delegation System

Delegation generally refers to assigning a particular job or duty to another person. However, in the context of the delegation system included in the software framework for collaboration between humans and multiple robotic systems, the meaning of delegation extends beyond human-to-human interactions [11]. Delegation in this context can be viewed as a human operator assigning a mission to a robotic platform (agent) or an agent assigning a mission to another robotic or human agent. The mission can be complex, and certain parts may be delegated to multiple agents as needed. This delegation process can continue recursively, resulting in a network of actions, indivisible from an external view allocated by agents, called tasks.

Figure 2.2 shows an example scenario of a delegation process [14] in the software framework. The mission specification is created using a user interface, an automated planner, or both, which is then translated into a TST. The delegation framework then delegates the tasks to the agents involved.

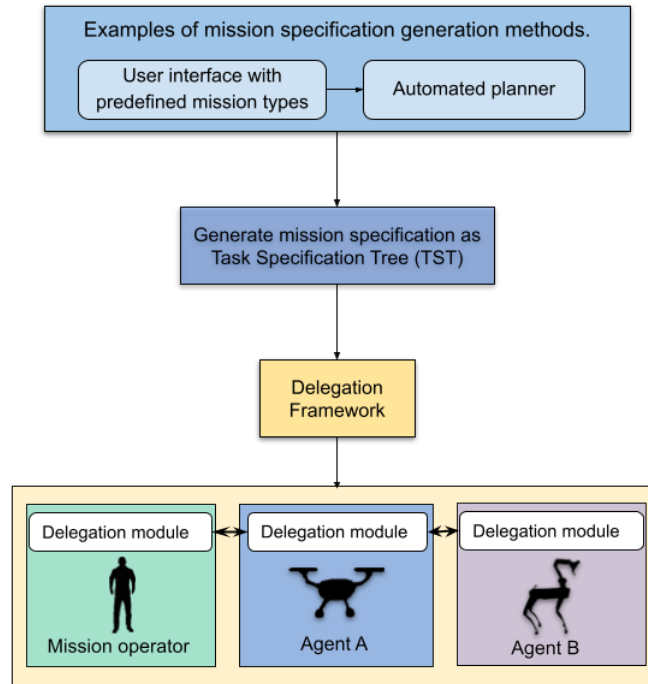


Figure 2.2: An overview of the delegation process

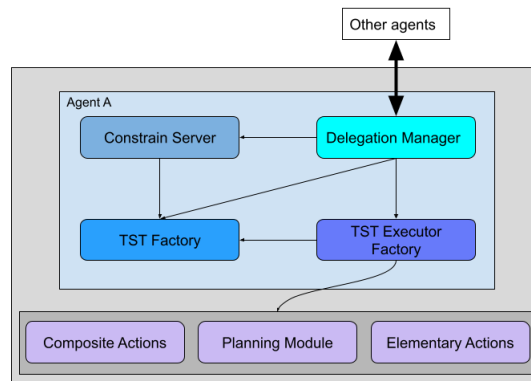


Figure 2.3: An illustration of a delegation module associated with each agent

In the delegation system, each agent in the collaborative team has a delegation module [1] consisting of four conceptual components: the Delegation Manager, the TST Factory, the TST Executor Factory, and the Constraint Server (see Figure 2.3). Each component handles different aspects of the delegation process and their respective roles can be summarized as follows:

- **TST Factory**

The TST Factory is responsible for generating TST nodes and TST sub-trees during the TST generation in the delegation process.

- **TST Executor Factory**

The Executor Factory manages the execution of executors associated with elementary or composite actions for a specific platform. These actions are publicly declared by the

platform and thus platform specific. Executors are executable procedural declarations defining how to execute a corresponding task associated with specific actions of one particular platform. In addition, TST Executor Factory also provides an interface to an automated planner for the robotic platform.

- **Delegation Manager**

The Delegation Manager manages interactions with other agents during the delegation process. Additionally, it handles the TST nodes generation and the execution of TSTs in the TST Factory and TST Executor Factory.

- **Constraint Server**

The Constraint Server handles constraints that TST nodes can have or inherit as the delegation process progresses. When an agent receives a task request from another agent, it can automatically set up a constraint problem and check for consistency, ensuring it can execute the task. The constraints can, for example, be temporal, resource-based, or associated with sensor capabilities.

## The Delegation Process

Given a high-level mission specification, the delegation framework is responsible for generating, specifying, and executing multi-agent plans. Suppose the mission is represented as a TST request by a human operator, it is modelled as a speech act of the form "Delegate (A, B, Task, Context)" [17]. Speech act is a concept that captures the idea that words are used for more than just presenting information; they can also be used to accomplish actions [45]. To put it simply, when a human speaks, we do not just deliver information but also conduct various acts with our words, such as requesting, asking questions, apologising and giving commands. For example, while eating, we might say, "The food is tasteless, can you pass me the salt". Here the information part corresponds to "The food is tasteless", and the acting part of the speech is "can you pass me salt". As for the delegation, the act is in the form of an agent ("A") requesting another agent ("B") to delegate ("Delegate") a task with the following information, task name ("Task") and a set of constraints for this task ("Context").

The speech act is implemented through an interaction protocol in two stages: the first phase and the second phase. During the first phase, tasks in the TST are allocated to agents who have the capability to perform them while satisfying all the task's constraints. In the second phase, the task allocations and constraint solution are presented to the operator, who can accept or reject them [11].

### First Phase

The delegation process is typically initiated by the same agent that provided the initial TST, which is usually the mission operator [11]. The process begins with the interaction protocol sending a "CALL-FOR-PROPOSAL" speech act to the mission operator, indicating that the root node of the initial TST, along with its specific constraints, is to be delegated. At this point, from the contractor's (i.e. mission operator) point of view, the process can be characterised using the DELEGATE-FIRST-PHASE procedure showed in Algorithm 1 [1]. The procedure begins by ensuring that an agent who wants to allocate the root node (task T) has the necessary capabilities to execute a task given the specified constraints (lines 2, 4, 5, and 6). If it does not, it responds with a REFUSE speech act. Otherwise, a delegation may be possible if the agent who wants the task can successfully delegate all of the root node's children. To find potential contractors for each child node, a REQUEST speech act for potential participants is broadcasted with a specification of the required capabilities (lines 10-11). An auction process is then initiated where each potential contractor bids for the task (line 12). Depending on the task, a bid can be based on the the robotic platform's need for assistance

or its sensors' capabilities. Bids are used to prioritise potential contractors, but backtracking may be needed if a decision is good for one part of the TST and has negative consequences for other regions of the tree. When a child node is provisionally delegated, the sub-tree may contain expanded nodes, and the nodes of the resulting tree are associated with constraints given by the contractor(s). The corresponding result (expanded tree and updated constraints) returned from a recursive delegation call are handled in lines 13-15. The agent provisionally commits to the delegated task before proposing a solution to the caller, the mission operator in this case (lines 16-17).

---

**Algorithm 1** Delegation process
 

---

```

1: procedure DELEGATE-FIRST-PHASE(task  $T$ , constraint set  $C$ )
2:   if basic capabilities for root( $T$ ) are missing then
3:     reply REFUSE
4:   Add constraints and parameters specified in root( $T$ ) to  $C$ 
5:   Add platform-specific constraints for root( $T$ ) to  $C$ 
6:   if  $C$  is inconsistent then
7:     reply REFUSE
8:   if root( $T$ ) is a leaf and this platform wants to expand it then
9:     Expand root( $T$ ), adding new children
10:  for every child  $c_i$  of root( $T$ ) corresponding to a subtree  $T_i$  do
11:    Broadcast a REQUEST to find  $P$  = potential contractors with capabilities for  $c_i$ 
12:    Perform auction for  $c_i$  among  $P$ , and sort  $P$  accordingly
13:    nondeterministically choose  $p \in P$ :
14:     $(T'_i, C) \leftarrow p.$ DELEGATE-FIRST-PHASE( $T_i, C$ )
15:    replace  $T_i$  with  $T'_i$  in  $T$ 
16:  Provisionally commit to the delegation
17:  reply PROPOSE( $T, C$ )

```

---

**Second Phase**

After a solution has been proposed to the caller, the caller can accept it or request a new alternative. If the solution is accepted, an ACCEPT speech act is sent to all relevant parties, including a specification of constraints for execution. A REJECT speech act is sent if the solution is not accepted.

## 2.3 ROS/ROS2

The delegation system described above is implemented using ROS/ROS2 (Robot Operating System). ROS is a distributed middleware framework for building complex robotic systems. It provides software libraries and tools that help users develop, deploy, and manage robot applications [47].

There are several ways in which ROS is used, such as hardware abstraction, device drivers, libraries, visualisers, message passing, and package management [48]. Moreover, due to its large user group, ROS also provides a community where developers can share helpful software. The original version of ROS was primarily focused on developing applications for a single robot, while ROS2 is designed to be more modular and flexible making it more applicable for multi-agent systems.

Spot ROS2 Driver<sup>1</sup> is an open-source ROS2 package for the Boston Dynamics Spot that has been used and further extended in this thesis. It contains essential ROS2 topics, services and actions for the Spot.

---

<sup>1</sup>[https://github.com/bdaiinstitute/spot\\_ros2](https://github.com/bdaiinstitute/spot_ros2)

## 2.4 Automated Planner

Achieving autonomy requires a system to use its knowledge of the world, including possible actions and outcomes, to determine the appropriate actions the system can take to achieve the desired goal. This process can be realised by using an automated planner. Most of the planners, including those that are used in this thesis, use a standardised Planning Domain Definition Language (PDDL). PDDL is a formal language based on first-order logic that defines planning domains and problems [41]. The domain specifies the general characteristics of a problem that are applicable universally and remain unchanged regardless of problems. It generally consists of object types, predicates, and actions. Object types are the types of objects that exist in the domain, such as people and places. Predicates, expressed using predicate logic, are the properties or relations between the objects, and actions are the operations that can change the state of the world.

The problem is defined by specifying initial states and the goal states using predicates and the objects in the problem. Initial states are the states of the world at the beginning of the planning problem, and the goals states are the desired state that the planner wants to achieve once the plan is executed.

### PlanSys2

ROS2 Planning System (PlanSys2)<sup>2</sup> is an open-source symbolic planning framework designed to support the development of complex planning problems. It is implemented in ROS2 and uses PDDL to define planning domains and problems. There are two planning algorithms available in PlanSys2 [39], Partial Order Planning Forwards (POPF) [6] (default planner) and Temporal Fast Downward (TFD) [27]. PlanSys2 transforms a plan generated by the planning algorithms into a Behavior Tree (BT), expressing the execution of the plan's actions. In addition to this, PlanSys2 also includes an algorithm for initiating calls to action performers, which are components that execute an action. This approach allows a multi-robot execution and specialised action performers to handle certain elements, such as debugging or a customised functionality for the action.

### Limitations

PlanSys2 has several limitations [39]. First, it only supports PDDL 2.1 [28]. Furthermore, as previously mentioned, it only has two plan solvers: POPF and TDF. It can be extended with other planners, but this requires the user to provide a plugin that contains how to call a specific planner and parse the generated plans. Finally, it does not take advantage of planners that offer an initial plan, and then improved plans can be generated when the initial plan is already running.

## 2.5 Vicon System

Vicon system is a high-precision 3D motion capture system that tracks reflective markers. It requires minimum three markers in a asymmetrical arrangement to determinate the orientation of a rigid body. Initially, the system was developed for gait analysis [3], but it is widely used in robotics. Studies of UAVs have, for example, used the Vicon system to determinate ground truth positioning, 3D reconstruction, and developing real-time controllers [18, 42, 44]. The Vicon system used in this study provides 3D poses (position along with orientation) and consists of 16 cameras (ten are of model T10 and six are T40s). The cameras are used in the configuration that covers a total area of approximately 10x10x5 meters.

---

<sup>2</sup><https://plansys2.github.io>

3

## Boston Dynamics Spot

The following chapter introduces the Spot robot developed by Boston Dynamics and used in this thesis.

### 3.1 General Description

The Spot robot (see Figure 3.1) is 1.10 meters long, 0.50 meters in width, and 0.84 meters in height (standing up). It can move at a maximum speed of 1.60 m/s and its typical run-time before needing to charge is 90 minutes [19]. The robot is equipped with five pairs of stereo cameras that provide black-and-white images and video [19] and a Light Detection and Ranging (LiDAR) sensor for measuring distances to nearby objects.

The particular robot acquired by the AIICS division is equipped with an arm that, at full extension, has a reach of approximately one meter [20] and has the following sensors in the gripper: a Time-of-Flight (ToF), an Inertial Measurement Unit (IMU), and a 4K RGB sensor. The ToF sensor is used for measuring distances to nearby objects, while the IMU is used for measuring linear and angular motion using a combination of accelerometers and gyroscopes. The RGB sensor is used for capturing images and video at a high resolution.

### 3.2 Communication with Spot

The Spot's API allows applications to control the robot, access sensor data, and create and integrate payloads. The API is based on a client-server model, where client applications communicate with the Spot robot services over a network connection. Client applications can run on tablets, laptops, cloud-based applications, or payloads connected to the robot. There are several methods of networking supported, the options are the following [24]:

- **Spot as a connected peer**

This method deploys the applications on computers physically connected to robot via the rear RJ-45 port, the DB-25 payload port, or the RJ-45 port on a Spot GXP payload. The method provides a reliable, high-rate communications link without infrastructure requirements.

- **Spot as a WiFi access point**

This method involves connecting to the robot's WiFi access point directly and requires



(a) The Spot robot docked to the dock-station and the *AprilTag* (a type of fiducial marker) in the background



(b) The robot, the first-aid kit (used as one of the objects in this thesis) and the *AprilTags* (a type of fiducial marker) in the background

Figure 3.1: Images of the Spot robot

no networking infrastructure. However, the method does require the application clients to be physically close to the robot.

- **Spot as a WiFi client**

This method requires that the robot is connected to the same WiFi network as the application clients. This option can extend the distance between the robot and the application clients, but the user must be aware of areas where the WiFi signal may be weak and the time it takes to switch between access points.

- **Spot via custom communications links**

This method is an option in cases where the aforementioned existing network options are insufficient. Custom communication links such as cellular modems (e.g. LTE/5G) and point to point radio links (e.g. Persistent Systems radio [25]) can then serve as a connection bridge between the robot and application clients.

### Spot API

Most of the Spot API uses Google Remote Procedure Call (gRPC), an open-source Remote Protocol Call framework [24]. This framework follows a client-server model [31], allowing communication between the client application and the robot's services through a request-response messaging technique. The messages used are Protocol Buffer messages with a format

independent of any programming language and include built-in serialisation and deserialisation. This provides a platform neutral and extensible method for encoding structured data in a way that is both forward and backwards-compatible.

For convenience of use, the Boston Dynamics provides a Python based SDK (Software Development Kit) abstracts the use of gRPC and Protocol Buffers. An example is the *AuthClient* class, which communicates with the *AuthService* server and has an authentication method called *auth*. This method requires input such as username and password and generates a Request message. The RPC is sent to the *AuthService*, which returns a Response indicating if the authentication was successful. An illustration of this is given in Figure 3.2.

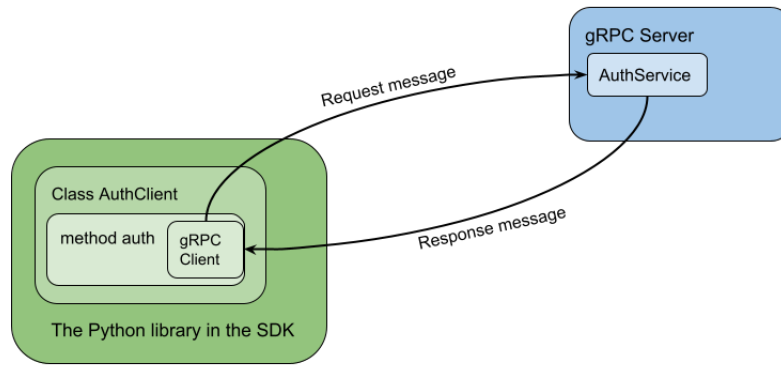


Figure 3.2: An illustration of how the SDK abstracts the use of gRPC

Although the abstraction simplifies most common use cases, there may be times when developers need to work with the protocol buffer directly. Therefore, the Boston Dynamics provides the documentation of message definitions and the data structures available on their web-page<sup>1</sup>.

### 3.3 Network Compute Bridge

The Spot API provides a *Network Compute Bridge* service server which allows developers to offload computationally demanding applications required by the robot to an external computer (e.g. another local computer or cloud server). These include, for example, inference processes based on deep neural-networks. In this study, the *Network Compute Bridge* is used to offload the extensive image processing computations required for detecting objects (e.g. first-aid kit) in images captured by the robot's cameras. Figure 3.3 shows the system diagram for how the *Network Compute Bridge* API is used. The diagram consists of an API Client (e.g. the tablet or a computer), the Spot robot, and the *Computer Server* (e.g. external computer that the computation server runs on). The *Network Compute Bridge* API helps coordinate the networking between different components (the robot and the *Computer Server*), capturing and rotating images and pre- and post-processing them before running them on the supervised machine learning model [23].

<sup>1</sup>[https://dev.bostondynamics.com/protos/bosdyn/api/proto\\_reference](https://dev.bostondynamics.com/protos/bosdyn/api/proto_reference)



Figure 3.3: An overview of the communication between the robot and the external computation server

Suppose that the API Client, for example, wants to request a detection of an object in an image by a machine learning classification model. In that case, the client application can send a request with a specific image source which the robot uses to capture the images, preprocess them and then sends them to the *Computer Server*. The result is then returned to the client via the robot after the detection process is finished.

### 3.4 GraphNav

The Spot robot uses a mapping, localisation, and autonomous traverse system called *GraphNav* [22]. It provides localisation, locomotion and other services<sup>2</sup> which developers can use. The services include for example, localising the robot, commanding the robot to navigate a route or recording a map. However, several of these services require the operator first to create a map, for example, using *Autowalk* available on the robot’s tablet or the *GraphNavRecordingServiceClient* for *GraphNav* recording services. In both methods the robot operator will first have to manually drive the robot along a path in an environment. This allows the robot to record and map the environment using fiducial markers which the robot can use to localise. A fiducial marker is a pattern or shape with known characteristics (see Figure 3.1) that can be recognised by a camera or sensor, allowing the system to determine its position and orientation relative to the fiducial. The recorded map can also be downloaded and uploaded to the robot for map processing services.

#### Map Structure

*GraphNav* uses a locally consistent graph to represent a map of the world. Nodes in the graph are known as *waypoints* and represent places in the world that the robot can navigate to. Each *waypoint* is associated with a snapshot of feature data from its location captured by the robot’s onboard sensors. The robot uses this data to track its position as it moves from one *waypoint* to another. For this reason, the robot might have difficulty navigating areas without many visual features. In such cases, fiducial markers whose features are easier to detect by the sensors can be used to help the robot through difficult areas.

An edge between two *waypoints* in a *GraphNav* graph includes information about how the robot moves. It contains both a transform describing the relative pose (position and orientation) of connected *waypoints* and parameters needed for controlling the movements between them. The parameters can, for example, indicate if the robot must use a stairs mode on this edge or if its speed should be limited to a specific value.

Boston Dynamics does not disclose what *GraphNav* is based on, but it is presumably based on a variant of a factor graph proposed in [9] considering its graphical model and the fact that it stores feature data from the robot’s onboard sensor in each *waypoint*. A factor graph is a probabilistic graphical model similar to a Bayesian network used to solve probabilistic inference problems. Namely, to deduce the probability distribution of one or several random

<sup>2</sup>[https://dev.bostondynamics.com/docs/concepts/autonomy/graphnav\\_service](https://dev.bostondynamics.com/docs/concepts/autonomy/graphnav_service)

variables taking into consideration the evidence (i.e. observed information). In robotics the random variables would correspond to the unknown state of the robot such as poses and the evidence would be the sensors' measurements. In other words, the objective is to estimate the most likely pose of the robot given the measurements from the sensors (i.e. posterior probability). Similar to the Bayesian network, the factor graph can also be used to specify a joint density of random variables, that is the probabilities of all combinations of the random variables. However, instead of just using the probability density function, it is also possible to use any factored function (i.e. a function that is expressed as a product of some factors). The factor graph uses a factor node to represent a factor function and is only connected to the variables it depends on. Furthermore, the evidence is treated as a fixed parameter for the factor nodes and is not explicitly represented like in a Bayesian graph. As a result, this approach enables faster inference computation.

Factor graphs have been applied to solve multiple problems related to robotic perception for example, real-time 3D-map representation [53], localisation, and mapping with computer vision [32].

### Anchoring and Anchoring Optimisation

*Anchoring*s is a concept in *GraphNav* that allows *waypoints* and fiducial markers on a *GraphNav* map to be mapped to any desired metrically consistent reference frame. Meaning for every *waypoint* and fiducial marker, there is a 3D transform describing its position relative to a reference frame. Transformations are mathematical functions that map a vector in n-dimensional space to another vector in the same space. They can represent various geometric operations, such as movement, rotation, scaling, and changes of coordinate systems [51]. In addition to this, it is also possible to get the robot's pose (position and orientation) relative to the reference frame. This allows the developer to use a *GraphNav* service called *NavigateToAnchor* to send the robot an approximate pose or just a position (x, y, z) in that specific frame. The *anchoring*s can also be optimised to improve the metric consistency of a map's *anchoring*. When the *NavigateToAnchor* service is called, the robot will navigate along the edges and waypoints in the map to the nearest waypoint to the given pose and then move straight toward the provided pose.

## 3.5 Coordinate Frames in the Spot World

The Spot robot uses 3D transformations to keep track of its pose (position and orientation) and the objects in its surroundings, and its perception of the world [21]. There are several coordinate frames that the robot uses to represent the objects, and the following frames are the fundamental ones that are useful in this thesis:

- **inertial frames**  
The inertial frames of the Spot are the *vision* frame and the *odom* frame. The origin and initial rotation of these frames are determined relative to the startup position of the robot. The *odom* frame estimates the robot's fixed location in the world based on its kinematics, while the *vision* frame analyses the surroundings and the robot's odometry to determine its location.
- **body frame**  
The *body* frame describes the robot body's position and orientation relative to its origin which is at the geometric centre of the hips, as shown in Figure 3.4.
- **seed frame**  
The SDK provides a map processing service that maps the *waypoints* and objects within the robot's surroundings (such as fiducial markers) to a desired global reference frame. The global reference frame is referred to as a *seed* frame in the Spot world.

- **world object frames**

The objects in the Spot robot surroundings can be described using a frame. Fiducial markers for example, are used regularly as visual markers to identify and track objects or locations. In the Spot robot's world, a fiducial is described using a *fiducial* frame and a *filtered fiducial* frame.

- **sensor frames**

Sensor frames are commonly used to describe detected objects' position and orientation relative to the sensor's origin.

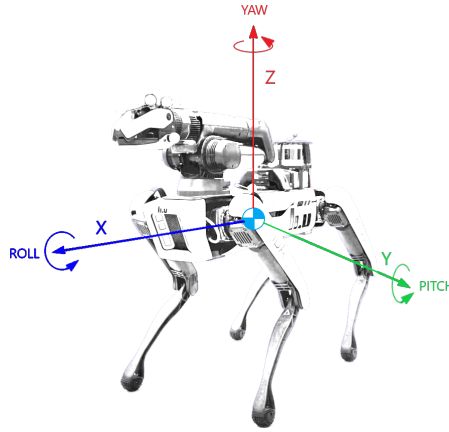


Figure 3.4: An illustration of Spot's body frame

Each frame can be described with respect to another frame using a 3D transformation. The transformation uses a vector  $[x, y, z]$  to describe the translation difference between their origin and a quaternion  $[w, x, y, z]$  to describe the change between two frames' coordinate axes' orientations.

The Spot robot's API uses a tree graph to represent relevant frames that can be uniquely described relative to another frame using a transformation. Different API services provide a snapshot of the transformation tree graph at a given timestamp, including relevant frames such as "*vision*", "*odom*", and "*body*" frames, as well frames for world objects.

### Transformations Between the Frames

Since frames are connected to each other through transformations, it is possible to transform geometric data from one frame to another. Suppose the transformation matrix for expressing frame B with respect to frame A ( $T_B^A$ ) and the transformation for expressing frame C with respect to frame B ( $T_C^B$ ) are known. Then the following matrix operation can be used to express a position in the frame C with respect to frame A ( $T_C^A$ ):

$$T_C^A = T_B^A \times T_C^B$$

A transformation can also be inverted which can be useful if a matching middle element of two transformation matrices is missing. That is if  $T_B^C$  (transformation matrix for expressing frame B with respect to C) is known instead of  $T_C^B$  in the equation above.

## 4 Theory and related work

This chapter contains theory and related work considered relevant to this study. The first part of the chapter presents the approach used for transforming between different coordinate frames. The second and third parts present related work using Task Specification Trees and other task specification languages. Finally, the last part presents papers related to automated planning.

### 4.1 Coordinate Frame Transformations

Since the robot uses multiple related coordinate frames expressing a pose (position and orientation) in a coordinate frame relative to another is sometimes more practical. Figure 4.1 shows an example of a scenario with more than one coordinate frame in the same problem. The reference coordinate frame (target frame) indicated with the superscript  $W$  is fixed at a known location in space and does not move over time. The fiducial coordinate frame (source frame) indicated with the superscript  $F$  is also fixed (it could also be other moving frames, such as the sensor frame attached to the robot). A transformation from fiducial frame (source) to the reference frame (target) can be described with a linear transformation as follows:

$$\vec{p}^W = \mathbf{R}\vec{p}^F + \vec{t} \quad (4.1)$$

where  $\vec{p}^W$  is the point expressed in the reference coordinate frame and  $\vec{p}^F$  is the point expressed in the fiducial coordinate frame,  $\mathbf{R}$  rotation matrix that is the relative rotation be-

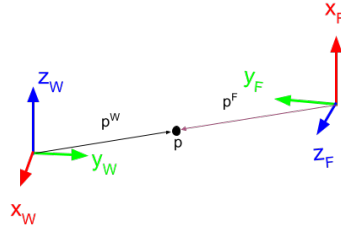


Figure 4.1: An example scenario with two coordinate frames, the reference coordinate frame (W) and the fiducial coordinate frame (F)

tween the coordinate frames (the transformation required to rotate points from the fiducial frame to the world frame), and  $\vec{t}$  is the translation difference between the origin of the frames [50]. Three methods can be used to describe the rotation: Euler angles, rotation matrix and quaternions.

### Euler Angles

Euler angles describe the orientation after a sequence of rotations around the coordinate axes. These rotations are called roll (a rotation around the x-axis), pitch (a rotation around the y-axis) and yaw (a rotation around the z-axis). For this reason, it is considered to be a more intuitive presentation than other methods. However, this method requires that the rotation order be specified since different sequences result in different orientations.

In other words, the orientation of a body after a 90 degrees rotation around the z-axis followed by a 90 degrees rotation around the x-axis does not have the same orientation as the same rotation applied but in the opposite order. It is also necessary to specify whether the rotations are relative to the original frame (static/extrinsic) or the frame after the previous rotation (relative/intrinsic) [50]. Euler angles are inconvenient to work with from a mathematical point of view since they suffer from the gimble lock effect [50] (when two axes become aligned, and most rotations fail to de-align), and the mapping from spatial orientations to Euler angles is discontinuous (small changes might cause a significant change in the required representation).

### Rotation Matrix

In this method, the rotation  $\theta$  is described by a 3x3 matrix and the orientation after a rotation of a point can be determined by pre-multiplying it with the rotation matrix [50]. The following rotation matrix represents a rotation about the three principal  $x, y, z$  axes:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (4.2)$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad (4.3)$$

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.4)$$

A rotation matrix can also represent a sequence of rotations as a product of the rotation matrix Eq. 4.2 - 4.4. Thus, converting from the Euler angle to a rotation matrix is straightforward. For example, rotations relative to the original frame (static rotation) with ZXY order, the rotation matrix can be retrieved by:

$$R_{intrinsic} = R_z(\theta)R_y(\alpha)R_x(\beta)$$

where  $\theta, \alpha, \beta$  are the rotations about the  $z$ -,  $y$ - and  $x$ -axis respectively.

### Quaternions

Quaternions expressed as  $q = (w, x, y, z)$  where

$$\begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \cos \frac{\Theta}{2} \\ v_x \sin \frac{1}{2} \Theta \\ v_y \sin \frac{1}{2} \Theta \\ v_z \sin \frac{1}{2} \Theta \end{pmatrix}$$

This representation can be interpreted as a rotation  $\Theta$  about an arbitrary unit vector ( $\vec{v} = (v_x, v_y, v_z)^T$  and  $\|\vec{v}\| = 1$ ) rather than predefined axes like Euler angles [50]. Quaternions are harder to interpret intuitively but can be computed efficiently and do not suffer from discontinuities associated with Euler angles.

## 4.2 Task Specification Tree and Automated Planner in Collaborative Robotic Systems

Several papers have used the Task Specification Tree (TST) and automated planning in collaborative multi-agent/robotic systems. Doherty et al. [12] have, in earlier work, proposed the TST as an abstract data structure for mixed-initiative problem-solving in collaborative multi-agent/robotic systems. The mixed initiative refers to the collaboration between a human and robots in the development, execution, and adjustment of a mission plan. This collaborative process enables the team to respond to unexpected events while maintaining the ability to adapt the level of autonomy as needed. The thought behind it was that "a task structure is a form of compromise between a compiled plan at one end of the spectrum and a plan generated through an automated planner at the other end of the spectrum" [12]. For this reason, the representation and meanings of a task should accommodate both. This has laid the foundation for incorporating automated planning into the delegation system proposed in the paper. The authors developed a prototype to evaluate the framework's effectiveness and practicality, which was applied to unmanned aircraft systems to demonstrate its real-world applicability. The authors concluded that further research is needed in this complex area, and their work serves as a basis for future studies.

Additional case studies that have used TSTs as the task specification language are, for example, [35, 14, 16]. In [35], the author examined the problem of allocating high-level tasks in mixed-initiative delegation for UAVs. As a result, a framework that combines the TSTs, a task allocation algorithm, and a dialogue management system to facilitate communication between human operators and UAVs was developed.

The framework was tested in several experiments involving a search and rescue mission. One particular test scenario's goal was to deliver boxes of food, water, and medical supplies to injured individuals in different places. In this scenario, there were five injured people, one ground operator in charge, and three UAVs. Some UAVs can carry multiple boxes simultaneously with a carrier, while others can only transport one box at a time. To accomplish the goal, the author used an automated task planner [34] first to create a plan and then the TST for the mission. This TST was then used together with the task allocation algorithm to delegate tasks to the UAVs involved.

The authors in [14] have used the TSTs in a collaborative framework for 3D mapping using UAVs. They describe the delegation process for the mission, which started with the operator specifying a high-level goal request represented as a TST without specifying execution details. The agents involved expanded the TST further, resulting in a more detailed execution specification. The goal request was, in the experiment, generated using a template. However, the authors mention that it is also possible to use a general automated task planner depending on the mission's type. Such missions are often domain-specific, and missions themselves can be used as a goal for the planning problem. Two different methods can be used to approach this. The first method involves generating a plan for the goal and converting it into a TST similar to [35]. In contrast, the second method adapts the planner to the delegation framework, allowing immediate delegation of new actions. One of the platforms in the experiment used the planner TFPOP [11, 33] for this reason. The authors tested the framework both in simulation and in the real-world experiments.

The paper [16] proposes a distributed hybrid deliberative/reactive architecture for unmanned aircraft systems. It describes how a particular type of node in a TST is linked to an

executor that specifies how a task should be carried out. Additionally, it was mentioned that a task planner was used to convert goal nodes in a TST into more detailed task specifications.

### 4.3 Other Task Specification Languages

In addition to TSTs, several other task specification languages have been developed and applied in collaborative multi-agent/robotic systems.

Behavior Trees (BTs) [8] are based on linear temporal logic, similar to TSTs, and the tree structure to represent and execute high-level behaviours. The trees are structured as hierarchical trees of nodes, where each node represents a specific behaviour or control logic. In contrast to BTs, TSTs are designed to represent tasks as a decomposition of sub-tasks. The TST architecture is focused on task specification and does not include explicit control logic. Therefore, a separate control architecture can determine how to execute sub-tasks.

The authors in [7] have presented a case study of extending a single robot BT to a multi-robot BT to control a group of mobile robots performing cooperative tasks. However, this extension presumes that all of the robots in the system have the same control logic, meaning that they are all executing the same BT, which may only be feasible or desirable in some situations.

Gracia et al. [29] have proposed a Domain Specific Language (DSL) for multiple robots called PROMISE. The authors aim to balance user-friendliness and well-defined formal semantics needed for planners, analysis tools, simulators, and other modules. In addition to this, it is platform-independent and allows users to specify high-level missions through a combination of executable tasks and operators, similar to TSTs. However, compared to TSTs, PROMISE focuses more on generating plans for ground service robots to achieve a high-level mission rather than collaboration between robotic agents and humans. The authors used PROMISE to define high-level missions in simulation and real-world experiments to evaluate the language's expressiveness. The authors have concluded there are several improvements that could be made, such as support for run-time changes in a mission specification or synchronisation among robots.

### 4.4 Automated Planning

Achieving autonomy requires a system to use its knowledge of the world, including potential actions and their outcomes, to determine the appropriate actions and timing to achieve the goal. This process can be realised by using a task planning technique, which involves organising a sequence of actions based on a given environment model to achieve specific goals. The PlanSys2 framework provides two temporal satisficing planners, Partial Order Planning Forwards (POPF) and Temporal Fast Downward (TFD), that can be used to accomplish this.

This section provides related work that focuses on the practical applications of these planners, providing an overview of the reasoning behind the selection of the planners and how to design a problem's domain in various contexts.

POPF, proposed by Coles et al. [6], is built on grounded forward state space search (starts in an initial state and applies applicable actions until the goal state is reached) and partial order causal link planning by delaying the following commitments: the ordering of the actions, timestamps, the values of numeric parameters, and managing sets of constraints as actions start and end. It has been applied to planning problems in various domains, such as generating a plan for an autonomous underwater vehicle for underwater installation [5, 4] and inspection and solving the power balancing problem in an electricity network [46]. Their decision to use POPF was mainly due to its ability to handle durative actions with continuous numeric effects and negative timed initial literals.

TDF was proposed by Eyerich et al. [27] and is built on a heuristic search in a forward temporal search space (i.e. time-stamped states). The default heuristic (a function used for

deciding the search approach) used in the search is inadmissible that is, it overestimates the cost of reaching the goal and domain-independent extended to cope with numeric variables and durative actions. The work in paper [37] has used TDF and dynamic system control to generate control input sequences for a system with linear dynamics and discretized inputs. The main reason for choosing TDF was that it could handle both durative actions and numeric variables that do not change monotonically since their implementations are pretty mathematically-computation-heavy. In addition, they have also concluded that the standard heuristic for the TDF has a few drawbacks in their type of problems, leading to poor guidance because the heuristic is domain-independent. The authors were however able to improve the search during the planning using their heuristic function instead.

A work comparing temporal planners, including TDF and POPF, is presented by Venturelli et al. [52], which studied the problem of optimising compilations of quantum circuits to specific quantum hardware. The process must consider constraints on the hardware and the duration of the circuit's execution, as quantum hardware degrades the performance of quantum algorithms over time. The result of overall performance, based on the ability to solve the highest number of problems, TDF was able to solve more problems. However, it spent much time on "processing axioms" and "invariant analysis". Moreover, when the authors consider the time required to execute the actions in the generated plan (makespan), the result has shown that POPF performance, in general, is similar to TDF.

## Symbolic Representation

Autonomous systems require a combination of high-level planning and low-level control to operate effectively. High-level planning involves reasoning about actions and goals in abstract descriptions represented in symbols. On the other hand, low-level control is responsible for executing the planned actions and often requires metric-valued parameters, such as coordinates of a waypoint.

In order to model a problem description in an autonomous system, the system must be able to interpret and encode the current state or metric-value parameters in a symbolic representation, such as Planning Domain Definition Language (PDDL). This process involves translating between real-world relationships (e.g. a package on the grass) and symbolic representations (e.g. the spot symbol represents the Spot robot). Moreover, one has to consider how to store the data associated with interpreting the relationships that will enable the translating process [4].

In a case study by Cashmore et al. [5], a Knowledge Base was included in their framework to translate PDDL actions to Robot Operating System (ROS) action messages, generate the PDDL problem file, populate the messages with actual data, and notify the planner if there is a change in the environment that may invalidate the plan. Creating a mapping mechanism between symbolic representations and metric values from perception is one challenge in developing such systems. While ontology development is a complex research area that can address this challenge [49, 36, 43], this thesis will focus on a simple mapping mechanism.



## 5 Method

The work presented in this thesis was divided into three phases: pre-study, implementation of necessary subsystems, and testing. In the pre-study phase, relevant materials were studied to define the problem and examine potential solutions. The implementation of subsystems phase was performed through iterations, beginning with defining the objective, exploring possible solutions, testing, and adjusting based on the results. Finally, the developed subsystems were tested in three different scenarios to ensure the robot could execute the desired complex tasks.

### 5.1 Pre-study

The pre-study step included the study of relevant material for this thesis:

- Spot robot's concepts and the SDK's structure including command issuing and available autonomy services.
- Examples of how the methods and classes in the SDK library can be used to achieve specific tasks.
- The pre-existing software framework for collaboration between humans and multiple robotic systems.

This step provides essential knowledge and understanding for conducting the implementations and evaluation.

### 5.2 The Implementation of Subsystems

This section describes how different behaviours of the Spot robot were realised and how the planning domain was modelled.

The communication with the robot in this thesis is done by connecting the robot and the application client to the same WiFi network (i.e. *Spot as a WiFi Client* see Section 3.2).

## The Spot Navigation

During the implementation phase, the first step was to create solution enabling the robot to move independently to a specific pose (position and orientation) within a fixed reference coordinate frame. As mentioned earlier, the robot's fixed location in the world is dependent on where it is powered on, which changes regularly. However, in this study it is desired that the robot should be able to power on anywhere and maintain a consistent coordinate system common among the Spot and other agents. This is not only important for reliable navigation, but also to enable the use of the Spot robot in collaborative missions. In order to accomplish this goal, the available services and features offered by the Boston Dynamics SDK library were integrated and used. Figure 5.1 provides an overview of the steps involved in this integration process. As shown in the figure, the initial step for the robot involves constructing a map of the surrounding environment. This is achieved via the *Autowalk* feature, which involved manually walking the robot through a designated area to generate the map. An example of the environment setup is shown in Figure 5.2.

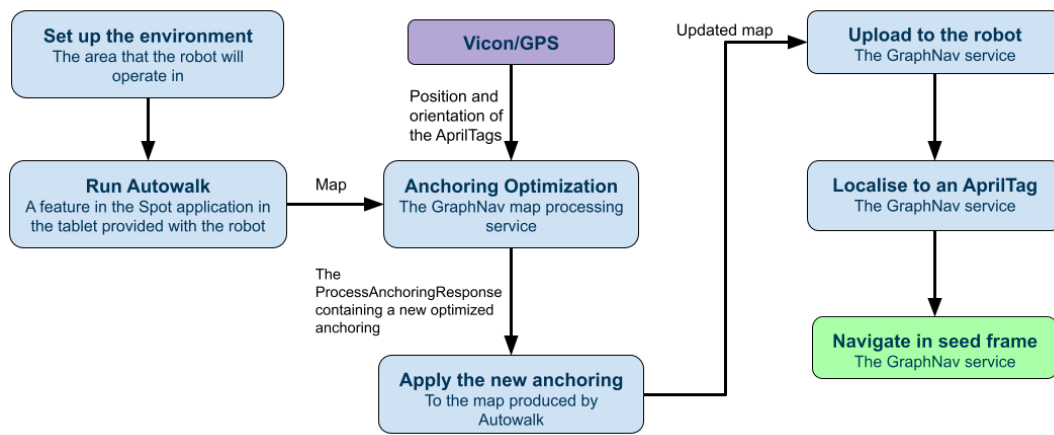


Figure 5.1: An overview of the implementation of the robot's navigation



Figure 5.2: An example of the environment setup

The map recorded during the *Autowalk* is then downloaded and used as one of the inputs to the *anchoring* optimisation provided by the *GraphNav* map processing service. Another input is the ground truth pose of the AprilTags (a type of fiducial markers that the robot recognised mentioned in Section 3.4) relative to the reference coordinate frame (e.g. GPS or Vicon system's coordinate frame), calculated by Eq. 4.1. Figure 5.3 shows a close-up image of the AprilTag mounted vertically against a surface, with its tag number upright and the setup of reflective orbs used by the Vicon system. The poses are used as *AnchoringHint(s)*, an initial guess of where the AprilTags are in the *seed* frame for the optimisation. A *ProcessAnchoringRequest* is sent to the *Map Processing Service* to find and improve metrically consistent *anchorings*. The request is responded with *anchorings* consisting of poses in the reference coordinate frame, which are used to update the map and upload it to the robot.

To navigate the robot, it must first initialise its localisation to the nearest AprilTag existing in the created map. It is worth to mention that there is always an AprilTag located at the robot's dock station (see Figure 3.1a). Once this is executed, the *NavigateToAnchor* command can be used to move the robot to an approximate pose in relation to the reference coordinate frame.

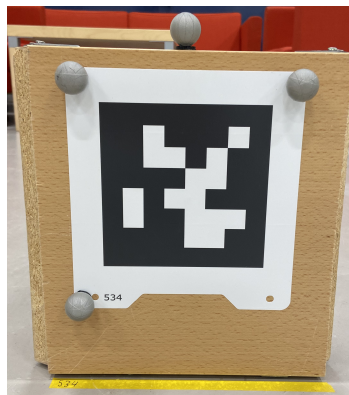


Figure 5.3: AprilTag setup

### The Spot Pick-up Behaviour

The *Network Compute Bridge* described in Section 3.3 was used to integrate the supervised machine-learning detectors (in this case a convolutional neural network), image capturing, and API calls from the client computer. An illustrated representation of the connection and communication between these components is shown in Figure 5.4. In order to identify the desired object, the client application first has to generate a protocol buffer message request as shown in Figure 5.4 (i.e. 1. *NetworkComputeRequest*). The request includes the following information: where to collect the images from, whether the images should be rotated before processing, the name of the machine learning model to be used, the minimum confidence level for classifications, and the server that will run the model. In this study, the request included the following:

- collect images from all of the image sources on the robot's body.
- the collected images should be rotated if needed (e.g. images from the front cameras).
- the minimum confidence for first-aid kit classified by the machine learning model is 0.8.

The machine learning classification model was running on a server using the same computer as the client application for all of the experiments.

The response from the *Network Compute Bridge* contains the detected entity's position in that respective frame, which can be transformed to the frame in which the command "move"

in that frame is possible (the *vision frame*). After the robot's move command is finished, the request to pick up the object in the image can be sent to the *manipulation API service* through a protocol buffer message. In this study, the request included the following information:

- the centre of the bounding box (pixel location) where the object was detected.
- collection of image and sensor transformations, including standard frames (vision, body and odom), all acquired at the same time.
- the frame name for the image's sensor source that detected the object and the camera model.
- the grasp parameters:
  - where in the gripper's palm to grasp the object is set to 0.6 (if set to 0, the object will be grasped against the most inner part of the gripper, and if set to 1, the object will be grasped at the tip of the gripper).
  - the optional constraint about the orientation was set to a top-down grasp.

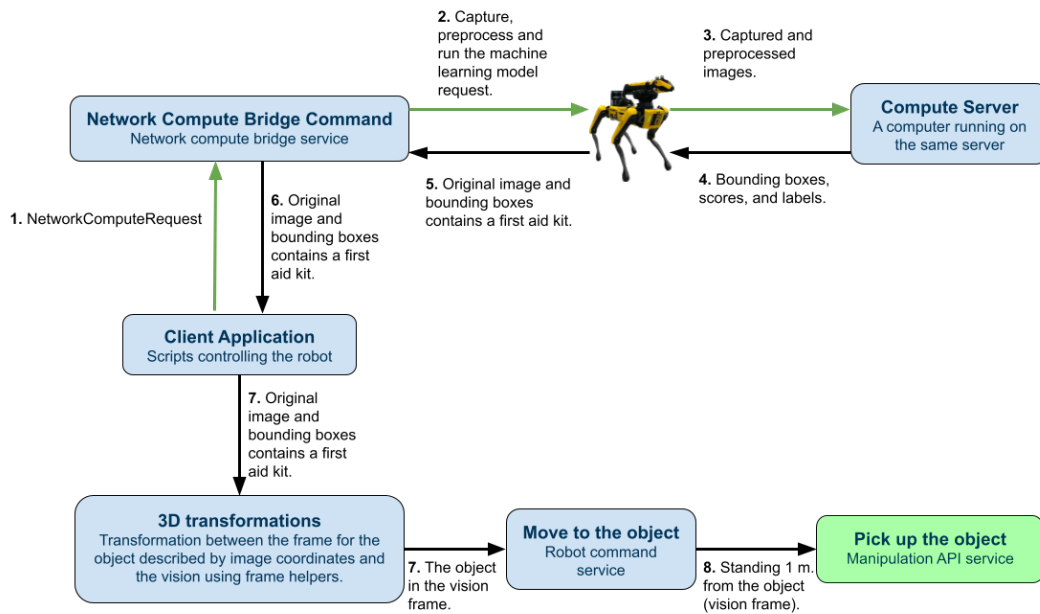


Figure 5.4: An overview outlining the steps in the implementation for finding and picking up an object, in this case a first-aid kit

### The Spot Deliver and Add to the Container Behaviours

Figure 5.5 provides an overview of how the delivering to a specific pose (position and orientation) in the fixed reference coordinate frame was implemented. In this part, a method provided in the Spot SDK Python library was used. This method abstracts the Remote Procedure Call to the API and therefore in this case only the desired pose in the robot's body frame has to be defined.

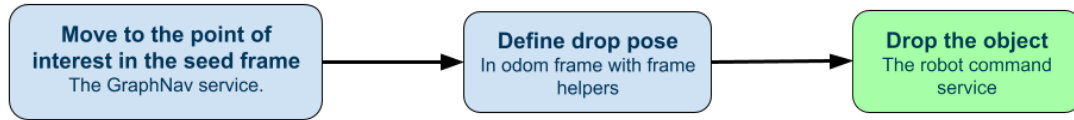


Figure 5.5: An overview of how the delivering behaviour was implemented

To mimic the ability to use a carrier which several of the robotic platforms have in one of the related work [35], an action that adds an object that has been picked up to the container on the robot's body was also added. Figure 5.6 shows an overview of how to command the arm to move along a defined trajectory. To command the arm, the *ArmCommand* request has to be sent to the *Robot command server* using a Protocol buffer message, and the request defined in this study contained the following:

- *ArmCartesianCommand* which contained a 3D pose trajectory (multiple poses for the arm relative to its body and the preferred time the arm should complete each pose).

An illustration of the command structure is shown in Figure 5.7. Furthermore, an example set of the *SE3TrajectoryPoint(s)* defined for action that moves a grasped object into the container mounted on the robot's body (a position and orientation of the gripper relative to the body) is shown in Figure 5.8.



Figure 5.6: An overview of commanding the arm to move along a trajectory process (e.g. to implement *pick-up* and store object in a container mounted on the Spot robot)

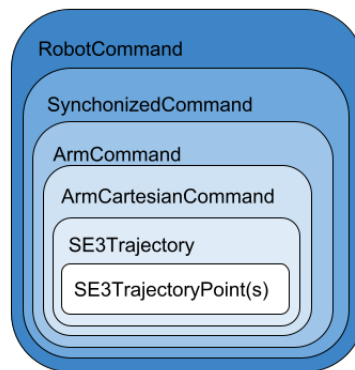


Figure 5.7: An illustration of the robot command request formation for commanding the arm to follow a trajectory

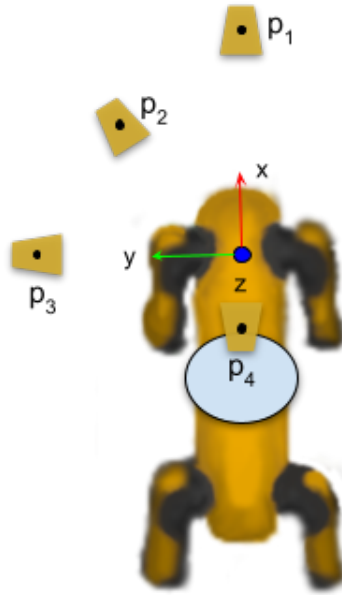


Figure 5.8: An illustration of the  $SE3TrajectoryPoint(s)$  defined in the action that adds a picked-up object to the container on the robot's body frame (z-axis point out from the page).  $p_1, p_2, p_3$ , and  $p_4$  represent positions, and the yellow triangles represent the orientation of the gripper relative to its body. The blue circle represents the container on the body

### Visualising Spot's Model

The Spot ROS2 Driver mentioned in Section 2.3 does not include necessary ROS topics to visualise the Spot's model correctly (i.e. the relation between the seed frame and the robot is not provided). Thus, to alleviate this problem the approach described in Section 3.5 was implemented. Given that the transformations for the body expressed in the seed frame ( $T_{body}^{seed}$ ) and the body expressed in the odom frame ( $T_{body}^{odom}$ ) are known, the seed expressed in the odom frame (i.e. odom frame is the robot's fixed location in the world see Section 3.5) ( $T_{seed}^{odom}$ ) is calculated. An example of the visualisation is shown in Figure 6.6.

### Modelling the Automated Planning Domain

The domain part of the Planning Domain Definition Language was defined using the following expressivity requirements: STanford Research Institute Problem Solver (STRIPS), typing, and durative-actions. Planning in general aims to determine a plan consisting of actions that will achieve the desired goals. Thus, for modelling the domain not all aspects of the world or all of the actions available (or actions that are forced by context) in the robot were defined. The domain was defined to include only the necessary information for the planner to compare different solutions. The aim was to avoid adding unnecessary details that would unnecessarily expand the search space, leading to an increase in computational time and resource requirements.

Actions defined in the domain correspond to commands a client application can issue to the robot. The commands are considered platform-specific and consist of the following:

- *stand-up*
- *power-motor*
- *deliver-package*
- *add-to-bucket*
- *pick-up*
- *sit*
- *walk*
- *localise*
- *dock*
- *undock*

where their meaning is mostly self-explanatory except *add-to-bucket*. This action commands the robot to move its arm toward a container attached on its body and drop the package as explained earlier.

Initially, the duration of each action in the domain was set to be approximately close to the actual time it took to execute. The reason why it was considered "approximately" was because some actions depend on varying factors, such as the map size during the execution of the *localise* action. However, this approach was later changed based on the results obtained from testing this particular domain with the planner across multiple problems. After the testing, some adjustments were made to the duration assigned to each action.

The problem part of the PDDL was defined using service calls provided by PlanSys2. The planner used to solve the problem was Partial Order Planning Forwards (POPF) (with best-first search as the search algorithm) since, from the related works, both POPF and Temporal Fast Downward (TFD) are comparable. This reasoning was, however, under the assumption that it is unimportant to use a customised search heuristic similar to [37] or the need for the planner to be domain-independent. Since POPF is already a standard, it was chosen to be used in the study.

### Task Specification Tree Executor

All the actions mentioned above were implemented in ROS2, enabling the TST Executors to perform these behaviours via the ROS service/action calls. In addition, a TST template was defined for every action with the aim of separating all low-level behaviours (basic actions) as much as possible.

## 5.3 Experimental Scenarios

The implementation was tested multiple times indoors and outdoors throughout the implementation phase to ensure its proper functionality. This section explicitly highlights the testing procedures that involved more complex scenarios outdoors. In the first scenario, the objective was to test if the robot could perform the *pick-up* and *delivery* behaviour. This involved moving to a specific location, picking up a package, and delivering it to a different location. The initial state of the robot in this scenario was: its power was on, it was localised and docked at the dock station (see Figure 3.1a).

In the second and the third scenarios, the objective was to mimic the UAV's capability to use a carrier to transport multiple packages at the same time, as mentioned in Section 4.2. Thus, given that there are multiple packages within an area and their locations are known, the robot is expected to walk to each location, retrieve the first-aid kit and add it to the container on the body. In this case, the goals of the PDDL problem were that each first-aid kit existing in the problem should be in the container while one should be carried and delivered to a specified location.

The main differences between the second and third scenarios were the number of packages involved and the initial state. First, in the second scenario, the number of packages to add to the container was lower. Second, in the second scenario, the robot's initial state differed from its state in the first and third scenarios in a way that it was standing and not

docked. Apart from that, the initial state was the same as in the first scenario. An example of the problem's definition in PDDL is shown in Listing 5.1. Line 3-6 defined objects in the problem, that is locations, packages and the Spot robot itself. Lines 9-19 defined the initial state of the domain that is the state of the robot (power state, localisation state and etc). Lastly, lines 23-26 specified the goals.

```

1  ( define ( problem problem_1 )
2  ( :domain spot )
3  ( :objects
4      loc1 loc2 loc3 loc5 loc0 - pose
5      spot0 - spot
6      pkg0 pkg1 - package
7  )
8  ( :init
9      ( power-on spot0 )
10     ( motor-on spot0 )
11     ( is-localized spot0 )
12     ( standing spot0 )
13     ( undocked spot0 )
14     ( at spot0 loc0 )
15     ( visited spot0 loc0 )
16     ( at pkg0 loc0 )
17     ( not-delivered pkg0 )
18     ( at pkg1 loc3 )
19     ( not-delivered pkg1 )
20 )
21 ( :goal
22     ( and
23         ( pkg-delivered loc1 )
24         ( in-bucket pkg1 )
25         ( docked spot0 loc5)
26     ) )
27 )

```

Listing 5.1: An example of the planning problem used in one of the test scenarios

In the test scenarios, there were three AprilTags, and the pose (position and orientation) of the AprilTags used as input to the *anchoring optimization* was provided by a portable GPS device. This device could transform the GPS coordinates to the local coordinates, which were then used as *AnchoringHint*(s).

## 6 Results

This chapter presents an overview of the subsystem and the results of the experimental evaluation of the implemented subsystem described in Sections 5.2 to 5.3.

### 6.1 Overview of the Subsystem Implementation

Figure 6.1 shows an overview of the system that enables the Spot robot to have the desired behaviours.

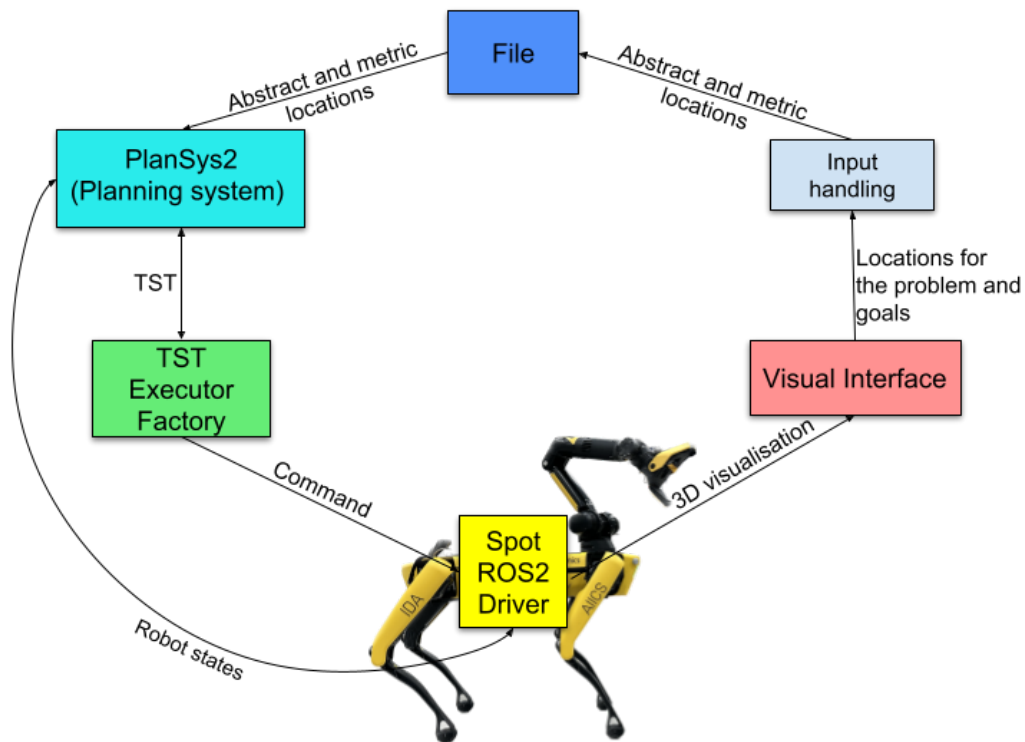


Figure 6.1: An overview of the subsystems implemented on the Spot robot

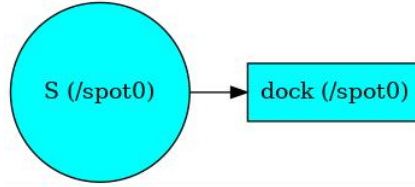
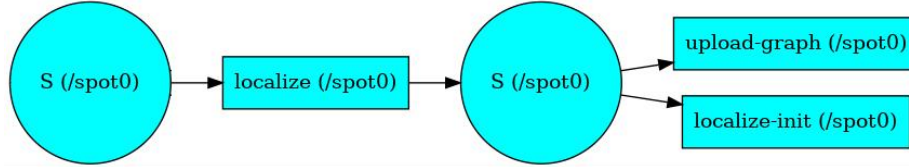
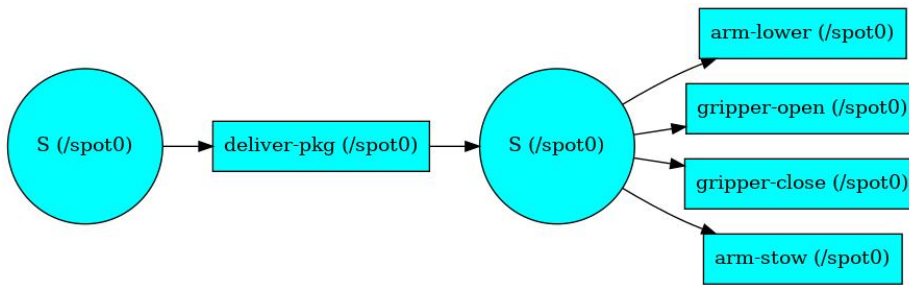
Starting from the yellow rectangle in Figure 6.1, it represents the Robot Operating System (ROS) driver used to control the Spot via ROS service calls and actions. The ROS driver also provides the robot's body pose in different coordinate frames and its perception, which are used to view the robot model in a visual interface (the red rectangle). With the interface, it is possible to select point which is then handled by a simple "Input handling" node. These points are later saved to a file as key-value pairs in a JSON file, and they are used by the planning system (the blue rectangle) when creating and executing a PDDL problem. The keys represent the names of locations (used when creating the planning problem), and the values represent the locations' coordinates (used when executing the plan generated from that problem). Additionally, the JSON file includes keys use to specify the planning goals. Three specific key values have been implemented. The *deliver* key indicates that a package should be delivered to a location. The *pkg* key specifies locations of packages. The *final* key denotes the final location at which the robot should finish its mission. An example of JSON file is shown in Listing 6.1.

```

1  {
2    "deliver": {
3      "loc3": {
4        "latitude": 58.39538734146802,
5        "longitude": 15.572315968249272,
6        "altitude": 104.4
7      }
8    },
9    "pkg": {
10     "loc1": {
11       "latitude": 58.3954196084843,
12       "longitude": 15.572322923562014,
13       "altitude": 104.4
14     },
15     "loc2": {
16       "latitude": 58.39540546868325,
17       "longitude": 15.572298592839863,
18       "altitude": 104.4
19     }
20   },
21   "final": {
22     "loc4": {
23       "latitude": 58.395404056429975,
24       "longitude": 15.572376404393726,
25       "altitude": 104.4
26     }
27   }
28 }
```

Listing 6.1: An example JSON file containing key-value pairs that represent the type of a location, its name, and its corresponding coordinate. Specifically, the key "deliver" corresponds to the delivery location (i.e. "loc3"), "pkg" indicates locations of first-aid kits (i.e. "loc1" and "loc2"), and "final" corresponds to the location where the robot should be at the end of the mission ( i.e. "loc4")

As shown in Figure 6.1, the planning system (Plansys2) also retrieves the robot states from the ROS driver and they are used to define the initial states in the planning problem. These states include the robot's current motor power state, body position state (sitting or standing) and localisation state (localised to a map). During the plan execution, a Task Specification Tree (TST) is generated for each action and sent to the TST Executor Factory. The system then makes ROS calls to the Spot ROS2 driver (see Section 2.3) to command the robot.

Figure 6.2: The TST for the action *dock* which mean docking the robot to its dock-stationFigure 6.3: The TST for the action *localise* which localise the robotFigure 6.4: The TST for the action *deliver-package* which delivers a package present in the robot's gripper

The TST for the following actions were modelled as a sequence node (S) followed by the action: *stand-up*, *add-to-bucket*, *power-motor*, *pick-up*, *sit*, *walk*, *dock* and *undock*. An example of the TSTs is shown in Figure 6.2. *Localise* and *deliver-package* actions are designed as expandable actions, as shown in Figures 6.3 and 6.4.

### Localisation

Figure 6.5 and Figure 6.6 show the robot's localisation before and after applying the *anchoring optimization* when the reference frame used was provided by the Vicon tracking system indoor (captured from the visual interface). When testing outdoors (see Section 5.3), the reference frame used was the GPS coordinate frame (transformed to the local coordinate frame) and the map created from the *Autowalk* prior to testing the scenarios is shown in Figure 6.7. Figure 6.8 shows a comparison between the robot's localisation and the GPS coordinates recorded during a walk outdoors, while Figure 6.9 presents the differences between them. It is worth noting that the recording started outside the building where the robot's dock station and its start location for all of the scenarios are located. This is because the GPS coordinate was inaccessible from the inside of the building, but the recording ended inside at the dock station.

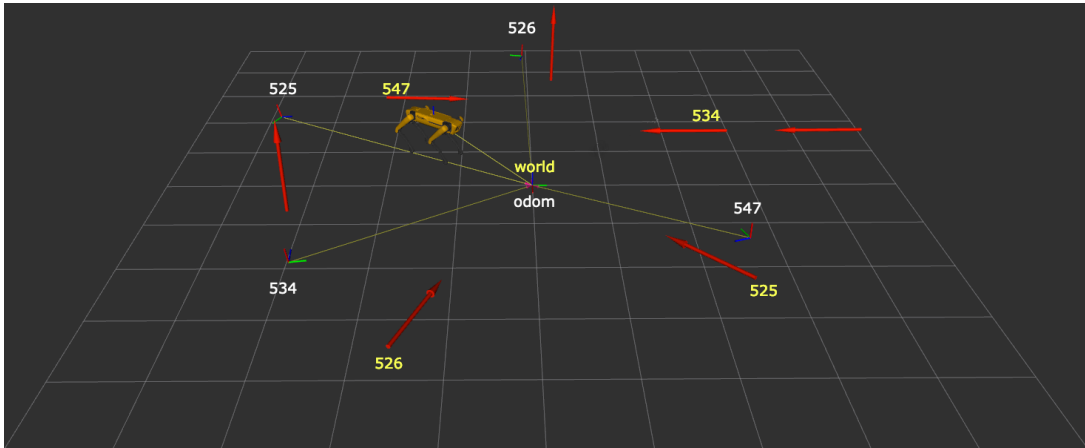


Figure 6.5: The robot's perception of the AprilTags' poses before the robot has been localised. A tag number identifies each AprilTag, and the origins for both coordinate frames are denoted by *world* and *odom*. The red arrows represent the true pose of each AprilTag

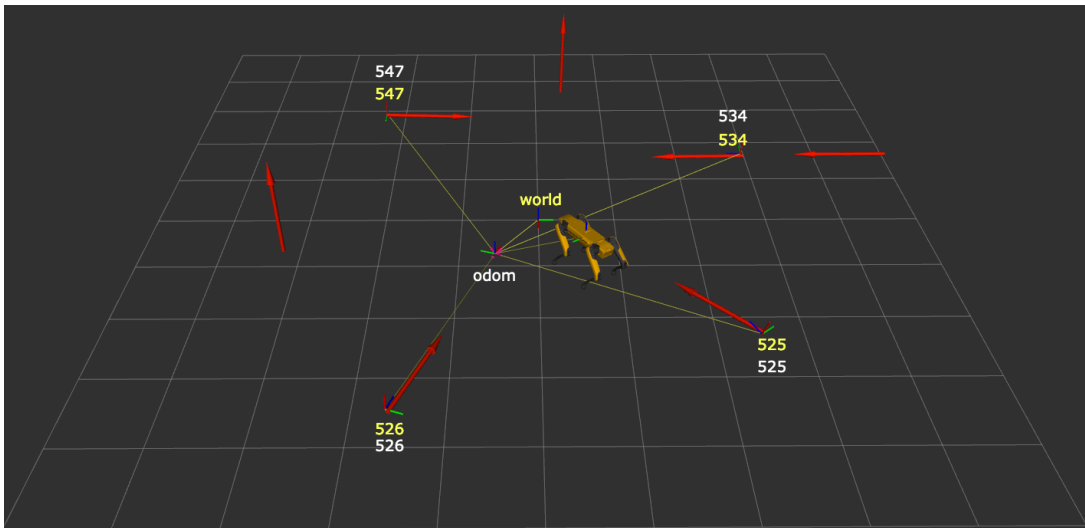


Figure 6.6: The robot's perception of the AprilTags' poses after the robot has been localised. An ID number identifies each AprilTag, and the origins for both coordinate frames are denoted by *world* and *odom*. The red arrows represent the true pose of each AprilTag

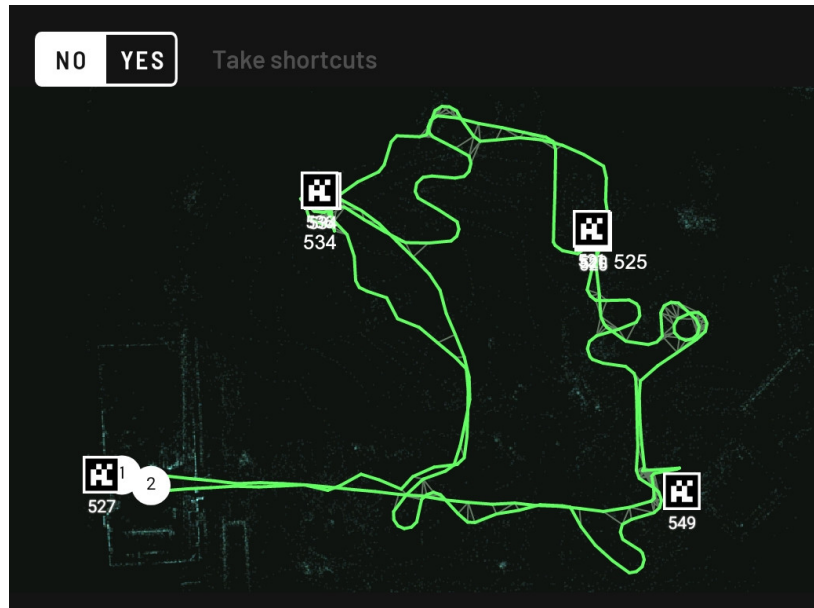


Figure 6.7: A map created from the *Autowalk* feature where the rectangles with black and white patterns are the AprilTags. The number below each AprilTag denotes its tag number

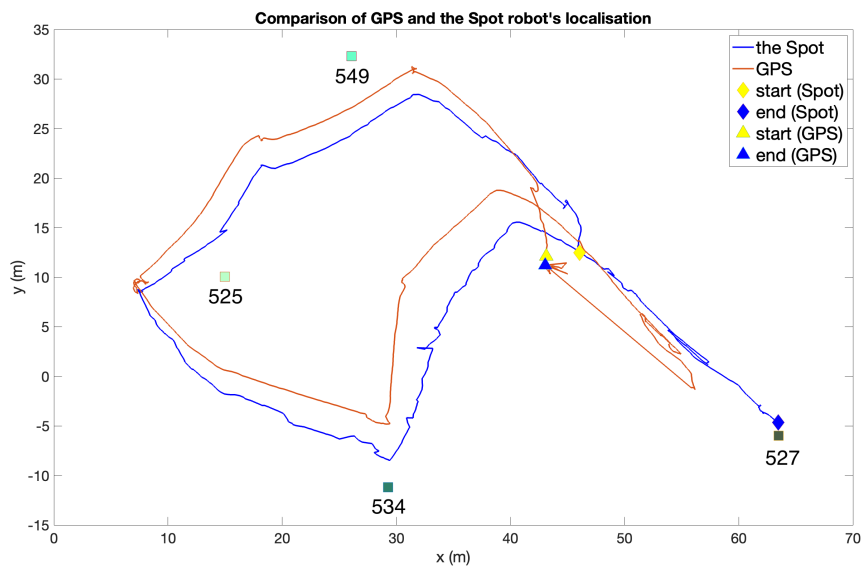


Figure 6.8: The Spot robot's localisation in comparison to GPS sensor readings. The rectangles represent the AprilTags, and the numbers under them are the tag numbers

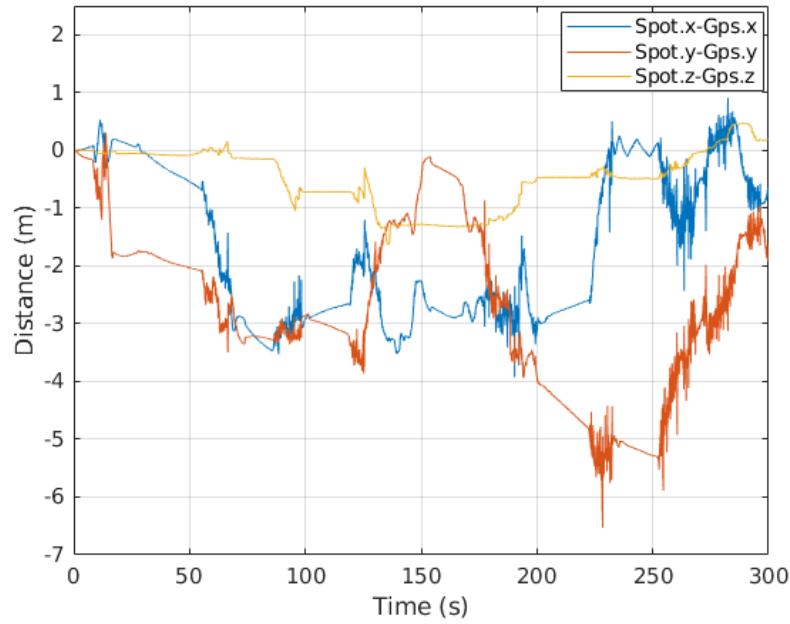


Figure 6.9: Difference between the GPS and the robot's localisation

### Modelling of the Planning Domain

Listing 6.3 depicts a plan generated for the problem in Listing 6.2 using the initial domain definition (see Appendix A.1), where the task durations were initially set to approximate the actual execution times. Examples of the plans generated using the final domain definition (see Appendix A.2) are presented later in Section 6.2.

```

1  ( define ( problem problem_1 )
2  ( :domain spot )
3  ( :objects
4      loc0 loc3 loc5 loc6 - pose
5      spot0 - spot
6      pkg0 pkg1 - package
7  )
8  ( :init
9      ( power-on spot0 )
10     ( sitting spot0 )
11     ( at spot0 loc0 )
12     ( visited spot0 loc0 )
13     ( not-delivered pkg0 )
14     ( not-delivered pkg1 )
15     ( at pkg0 loc5 )
16     ( at pkg1 loc6 )
17 )
18 ( :goal
19     ( and
20         ( pkg-delivered loc0 )
21         ( pkg-delivered loc3 )
22         ( sitting spo0 )
23     )
24 )

```

Listing 6.2: One of the problems tested defined in PDDL

```

1   Plan
2   ( power-motor spot0 )
3   ( localise spot0 )
4   ( stand-up spot0 )
5   ( walk spot0 loc0 loc5 )
6   ( pick-up spot0 pkg0 loc5 )
7   ( walk spot0 loc5 loc0 )
8   ( deliver-pkg spot0 pkg0 loc0 )
9   ( walk spot0 loc0 loc3 )
10  ( walk spot0 loc3 loc6 )
11  ( pick-up spot0 pkg1 loc6 )
12  ( walk spot0 loc6 loc3 )
13  ( deliver-pkg spot0 pkg1 loc3 )
14  ( sit spot0 )

```

Listing 6.3: The plan generated for the problem in Listing 6.2

## 6.2 Experimental Evaluations

This section describes the results retrieved in the testing scenarios mentioned in Section 5.3. Videos showing the result of each scenario are also available online on OneDrive.

In the first scenario, the robot started at the position marked by the yellow triangle (see Figure 6.10) and walked to the location marked by the green circle to pick up the package. After that it moved to the position marked by the red star to deliver the package. Finally, it returned to the starting point and docked. The actions executed during this test scenario are presented in Table 6.1.

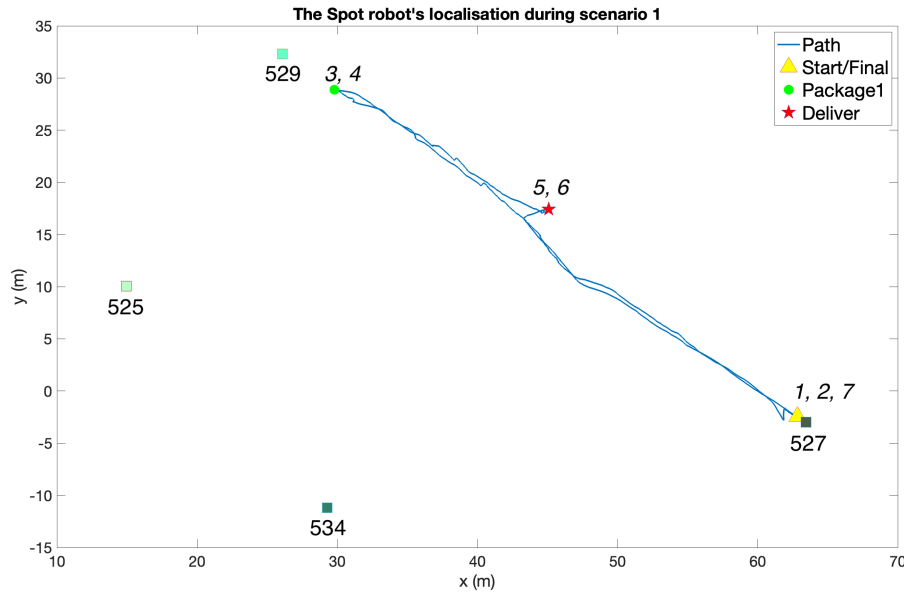


Figure 6.10: The Spot robot's localisation during the test scenario 1. The rectangles represent the AprilTags (see Figure 6.7 for a comparison), and the numbers under them are the tag numbers (527 is the AprilTag located at the dock station). The other numbers correspond to the starting point for each planner's actions in Table 6.1

	action	start time (s)	end time (s)
1	<i>undock</i>	0	7.2901
2	<i>walk</i>	8.4716	63.5730
3	<i>pick-up</i>	63.095	82.279
4	<i>walk</i>	82.7	110.99
5	<i>deliver-pkg</i>	111.59	129.58
6	<i>walk</i>	129.97	166.78
7	<i>dock</i>	167.19	186.79

Table 6.1: The actions generated by the planner for test scenario 1, along with their respective start- and end times during the execution

In the second scenario, the robot started at the position marked with yellow triangle and walked to the location marked by the green circle to pick up the first package (see Figure 6.11). Then, it placed the package in the container mounted on its body, and moved to the position marked by the purple circle for the second package. It picked it up and then moved to the location marked by the red star to deliver the packages. Finally, the robot returned to its final point, marked by the blue triangle in the figure, and docked. The actions executed during this test scenario are presented in Table 6.2.

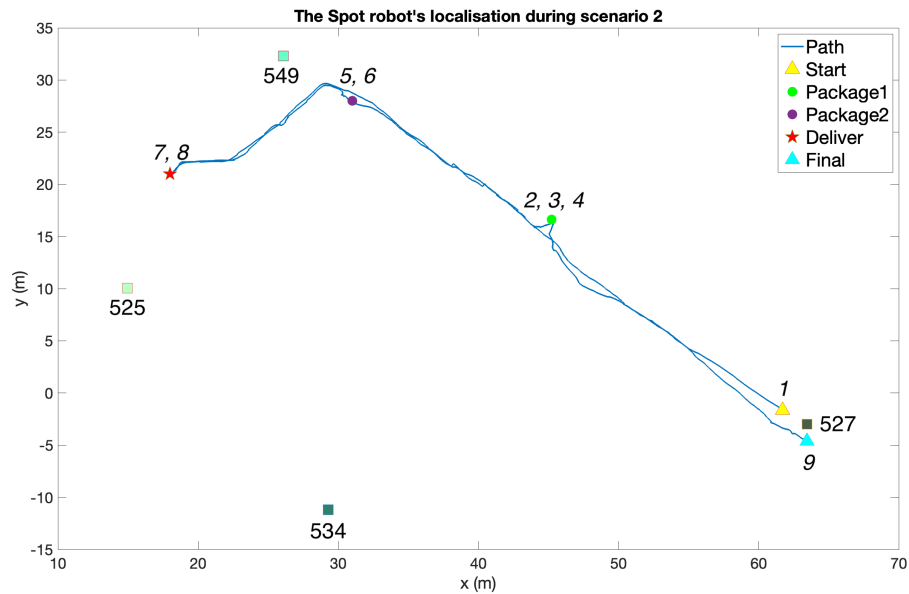


Figure 6.11: The Spot robot's localisation during the test scenario 2. The rectangles represent the AprilTags (see Figure 6.7 for a comparison); the numbers adjacent to them are the tag numbers (527 is the AprilTag located at the dock station). The other numbers correspond to the starting point for each planner's action in Table 6.2

	action	start time (s)	end time (s)
1	<i>walk</i>	0	35.095
2	<i>pick-up</i>	35.592	51
3	<i>add-to-bucket</i>	51.401	61.3
4	<i>walk</i>	61.803	88.303
5	<i>pick-up</i>	88.286	104.4
6	<i>walk</i>	104.78	129.25
7	<i>deliver-pkg</i>	129.48	146.4
8	<i>walk</i>	146.79	214.3
9	<i>dock</i>	214.68	232.35

Table 6.2: The actions generated by the planner for test scenario 2, along with their respective start- and end times during the execution

In the third scenario, the robot began at the location marked by the yellow triangle and walked to the position marked by the green circle to retrieve the first package (see Figure 6.12). It then placed the package in the container mounted on its body, and proceeded to the locations marked by the orange- and pink circles to do the same. After that, the robot moved to the location marked by the purple circle and started to pick up the last package; however, the robot could not pick it up, so it stowed its arm and moved on to the next task. The robot then headed towards the delivery location (marked by the red star), and once it arrived, it executed the delivery command even though it did not have any packages in its gripper. The actions executed during this test scenario are presented in Table 6.3.

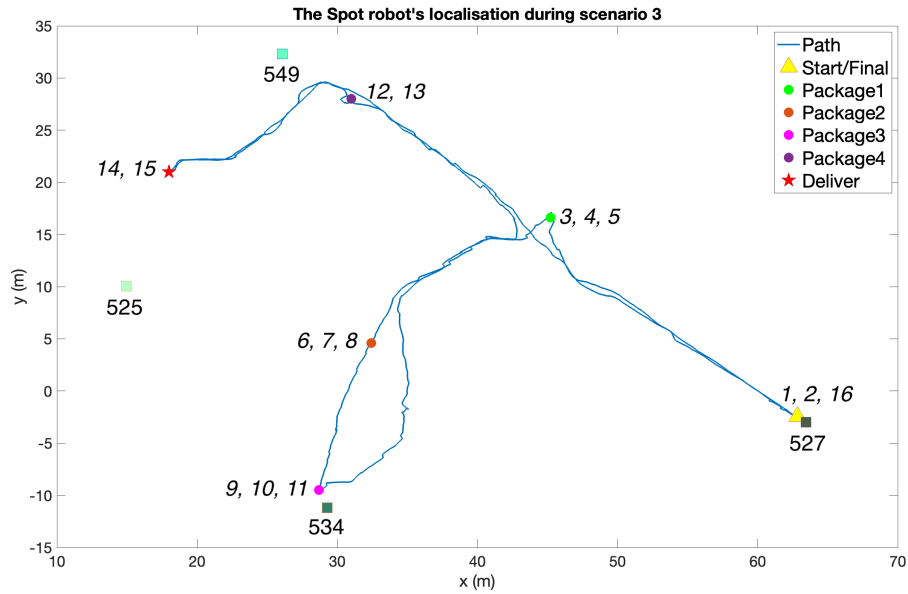


Figure 6.12: The Spot robot's localisation during the test scenario 3. The rectangles represent the AprilTags (see Figure 6.7 for a comparison); the numbers adjacent to them are the tag numbers (527 is the AprilTag located at the dock station). The other numbers correspond to the starting point for each planner's action in Table 6.3

	<b>action</b>	<b>start time (s)</b>	<b>end time (s)</b>
1	<i>undock</i>	0	7.21
2	<i>walk</i>	7.6051	43.838
3	<i>pick-up</i>	43.805	60.917
4	<i>add-to-bucket</i>	61.293	71.315
5	<i>walk</i>	71.7	105.62
6	<i>pick-up</i>	106.12	124.21
7	<i>add-to-bucket</i>	124.6	134.1
8	<i>walk</i>	134.5	149.72
9	<i>pick-up</i>	150.19	169.84
10	<i>add-to-bucket</i>	169.8	179.89
11	<i>walk</i>	179.8	245.61
12	<i>pick-up</i>	246	266.6
13	<i>walk</i>	267	292.1
14	<i>deliver-pkg</i>	292.7	310.71
15	<i>walk</i>	311.09	380.9
16	<i>dock</i>	381.3	400

Table 6.3: The actions generated by the planner for test scenario 3, along with their respective start- and end times during the execution



## 7 Discussion

This chapter discusses the results of the study and the methodology used. It also analyses potential opportunities for improvement and the ethical and societal aspects of the work.

### 7.1 Results

This section compares the obtained results to related works and discusses the notable outcomes.

#### Overview of the Implementation

In contrast to the related work [14], which used the task planner adapted to the AIICS division framework to create a plan and generate a Task Specification Tree (TST), this study makes a TST for each action during plan execution. If an action can be expanded, the TST Executor Factory will expand it using a pre-defined template (e.g. *localise* action, which expands to a sequence of uploading a graph and initialising localisation to a fiducial). This follows from the PlanSys2 planning system structure since it first translates the plan into a Behaviour Tree with the leaves representing actions of the plan, which then executes the low-level control modes. The planner POPF used in this study occasionally generated plans that were not optimal, as shown in Listing 6.3. In lines 9-10, instead of walking directly from *loc0* to *loc6*, the plan included a detour. The result is understandable since it is not an optimal planner meaning it is not guaranteed it will always return an optimal plan with respect to some cost function. Setting the search parameters of the planner search to the best-first search algorithm and modify the duration of the actions resulted in more reasonable plans.

#### Localisation

As shown in Figure 6.9, there are some differences between the recorded GPS coordinates and the robot localisation. These differences fluctuate and are especially large on the y-axis, reaching magnitudes up to 6 meters. However, it is noteworthy that they converge to 0 meters within some intervals, and there are some possible explanations for this.

First, the robot uses LiDAR to map its surroundings by extracting the environment's features and saving that information in the waypoints (see Section 3.4). When it is close to, for

example, building structures or trees, it gains higher confidence in its whereabouts. Contrarily, it becomes less certain when it moves to an open area with limited distinguishable features.

Second, during these intervals, the robot was close to the AprilTag (see Figure 6.8, the difference close to the AprilTags 525 and 534), where it could retrieve the position, orientation and relative distance with higher certainty.

Finally, it also has to be taken into consideration that the GPS sensor was placed in a different position than the origin of the robot's body frame. Consequently, this will affect the measurement, but the differences should be constant. Additional experiments are required before any conclusion can be made.

## Experimental Evaluations

The results obtained in the test scenarios are overall good. The actions generated by the planner are appropriate from the human's point of view. However, the duration of the actions during the execution (shown in the Tables 6.1 - 6.3) differs to some extent from the duration defined in the planning domain (see Appendix A.2). This is expected because the duration for the planner action *walk-to* is always the same even though the distances between the locations are different. Unlike one of the related work [4], which took the distance between the locations and the reorientation time (i.e. the duration it takes for the AUV to turn to the correct orientation) into consideration to make the plan capable of handling concurrent activities. Considering that there is only one Spot robot in the domain, the duration of the action does not have a significant effect on the plan execution as long as the generated plan is reasonable. However, if multiple robots are involved, then the duration of an action will have a significant effect as the states of the world change according to the duration. For example, consider a scenario where the plan is: robot *A* delivers a carrier of packages to location *loc1*, and then robot *B* retrieves a package from the carrier at *loc1* and delivers it to *loc2*. If the duration specified in the planner action is inaccurate (i.e. the execution takes longer time), then robot *B* will start to execute its action while there is no carrier at *loc1* and naturally, the plan fails.

As previously mentioned in Section 3.4 and according to the results (see Figures 6.10 - 6.12) compared to the map in Figure 6.7, it is shown that the robot follows the same path created during the initial *Autowalk*. In other words, it did not take paths that the operator might consider better (e.g. shorter distance). To ensure that the robot takes the preferred (e.g. shorter distance) paths, the operator should walk it on as many paths as possible during the *Autowalk* recording. However, as shown in Figure 6.7, there is a possibility to set the option "Take shortcuts" to true, which was not tested in this thesis, and could potentially result in the robot taking a different path that was not created in *Autowalk*. Furthermore, relating to the robot's locomotion, when the robot is commanded to move to a specific pose (position and orientation), it will first move to the the closest *waypoint* and then proceed straight to the pose. This could explain why the robot occasionally moves strangely, such as moving backwards, rotating 360 degrees just before the destination and heading towards it. Another possible explanation for this behaviour could be related to the information about how the robot should move saved in an edge between two waypoints (see Section 3.4). The edge from the *Autowalk* may specify that the robot should move in that particular manner; consequently, when combining this with the instruction that it should move straight to the nearest *waypoint*, results in a 360-degree rotation. Further experimentation is needed to draw any concrete conclusions.

Many times during the testing scenario outdoors, when using the top-down grasp method (meaning adding constraints to the *PickObjectInImage* request that it should grasp top-down), the feedback from the arm command was "the robot failed to pick up due to no motion plan can be found". Motion plans are generated by the robot and contain commands that solve the problem of how to move the arm from the start position to the specified pose. Once the

constraints were removed, the problem of being unable to find a motion plan disappeared, but there were instead other internal control level factors that led to failures when picking up an object. This happened in test scenario 3 (see Section 6.2), and for this reason, the robot failed to pick up the last package. In this case, it would be reasonable to retry the *pick-up* command again, and this behaviour was added at one point during the tests. However, the result of the second attempt resulted in the same behaviour as in the first attempt. This could depend on the fact that the robot still stands in the exact same pose as during the first attempt, resulting in the same inadequate motion plan for the arm that causes the *pick-up* action to fail. To solve this, one possible approach would be to command the robot to move to a new pose nearby and try again, in case the robot fails to pick up the object. More tests are needed to determine the possible reasons for this behaviour.

## 7.2 Method

This section discusses the applied method and what could have been done differently.

### TST Structure

Most of the TST defined do not utilise the full potential of the TST node types available. For example, the current definition of the TST template for the planner action *pick-up*, handles the *pick-up* failure directly on the low-level/control level. If the action fails, it puts the arm back to its initial position instead of the carry position. This could however, be defined differently using the TST "Test-if" node where the query would be *pick\_up\_test* corresponding to the *in\_air\_test* in the example shown in Figure 2.1. If the *pick\_up\_test* succeeds or is unknown, the robot should wait, and if the *pick\_up\_test* fails, the robot should try again. This could be done by executing a sequence of *move-to* (i.e. to a new pose nearby) and then *pick-up*, which can also be represented as a TST that is a "Sequence" node with *move-to* and *pick-up* as its child nodes. In this way, it would be easier to monitor the execution and detect at which step the execution has failed. Generally, handling these kinds of problems is easier at the level where the robot is controlled (i.e. where the Spot's API service calls are made). Consequently, implementing the extended TST that uses additional nodes, while it offers more flexibility, is not trivial. It requires, for example, finding a way to check and access the states/action-result at the API level. In future work, one should consider extensions to the current implementation to make it modular and consistent with the ROS driver node and allow more complex/flexible TSTs.

Similarly, the planner action *localise* which expands to upload the optimised graph to the robot and then initialise the localisation (see Figure 6.3) could also have used a "Test-if" node. In case the robot is in the sitting position, it most of the time fails to recognise an AprilTag and therefore fails to localise. This could be improved by checking if the localise to an AprilTag failed; if true, stand up and execute the action again.

If the TSTs were defined using the abovementioned approach, the resulting TSTs would be even more correlated to the system developed by the AIICS division. But eventually, the results are the same during the execution. The TSTs provide a foundation for future improvement since it is already known what necessary methods/API requests are required for each action, making it easy to isolate them.

### Pick-up Command

Based on the test results, it is noticeable that the environment plays a significant role in the *pick-up* command execution, particularly in bright sunlight. This is understandable because the robot relies on a pixel location in the image to determine the target object. Although this study only tested one manipulation API, other APIs and constraints could improve the results. The chosen method was based on an example from Boston Dynamics and was selected

early in development. One could approach this by investigating how different methods work and selecting the most appropriate one.

### Replicability, Reliability, and Validity

The developed planning domain (see Appendix A.2) does not consider the distances between different locations. During empirical evaluation of the test scenarios the symbolic locations were mapped manually in a way that the robot will visit the closest location first. Thus, this resulted in it moving to the closest location first and then to the next (in an optimal way).

Furthermore, as mentioned earlier, how the robot moves during the test and its path depends on how the operator walked it during the *Autowalk* (generated a map). The denser the map is (many waypoints connected), the more path alternatives the robot can choose between and vice versa. These factors may affect the repeatability of the study in this thesis.

The study did not investigate the effect of the number of AprilTags on the anchoring outcome. Because there is a ground truth available, it was possible to compare it to the robot's measurement directly. It showed that more than two AprilTags were needed for the optimisation API to determine the proper anchoring transformation. More experiments could have been conducted to gather additional data, increasing the results' reliability and validity.

### Source Criticism

The references used in this thesis are primarily from scientific and peer-reviewed databases, such as IEEE Xplore and SpringerLink. These references were mainly sourced using Google Scholar with relevant search words such as "Task Specification Tree" and "ROS planning". In addition, a recursive search approach was used, which involved following the cited-by works in the reference and exploring the works of the authors. However, there were some elements of the thesis, such as the *GraphNav* and methodologies used to implement the Spot's functionalities that the most relevant information can not be found in scientific studies. For *GraphNav*, the reason for this was that Boston Dynamics does not disclose specifically what it is based on. The scientific paper used for this was the work that has the closest characterisation to *GraphNav*'s description given by Boston Dynamics. Similarly, the methodologies used to implement the functionalities (e.g. *PickObjectInImage* API service call and moving the arm along a predefined trajectory) were based on documentation for the robot from Boston Dynamics, which did not disclose the scientific theory behind it.

## 7.3 The Work in a Wider Context

The purpose of this study is to examine the potential use of robots for rescue operations in the future. However, a robot is just a machine, and in certain situations, injured individuals may require more than just basic necessities like first-aid, food, or medicine. For instance, they may need emotional support, which robots cannot provide, especially for children who may be traumatised after a disaster. Empathy is a human trait that robots do not possess. Even 1177, a Swedish online healthcare service developed by governmental organisations, advises against leaving a person in shock [40] (assuming that the robot delivers the package and then moves on to another person) or offering food to someone in shock. However, the robot has a speaker that the rescue personnel or medical professionals can use to communicate and comfort the injured person to some extent.

The robot is commercially available, and it is commonly accepted that the internal implementation of the product to be kept confidential or undisclosed to the customer. Thus, in untested surroundings or due to users' assumptions, it may function unexpectedly (similar to the pick up behaviour mentioned earlier). This is particularly important to consider during rescue operations, which can be stressful and difficult.



## 8 Conclusion

The purpose of this thesis is to investigate how the Boston Dynamics Spot robot can be integrated into the software framework for collaboration between humans and multiple robotic systems developed by the Artificial Intelligence and Integrated Computer Systems (AIICS) division. Additionally, it examines how the Spot robot can be designed to be deliberative, meaning it can reason and determine which actions it should take to achieve its desired goals, considering its capabilities and current state. The primary desired goal here is to deliver a first-aid kit from one specific point to another. This could be proven helpful in a rescue operation assisting the public safety organisations in the future.

To determine the achievement of the aim, the system was tested in various scenarios and demonstrated the ability to plan and accomplish the intended goal. However, there were occasions when achieving the goal proved challenging due to various limiting factors. As part of this, the following questions were answered:

**1. How can the Boston Dynamics Spot robot be integrated into the collaborative robotics software framework developed at AIICS division?**

This was achieved by extending the pre-existing architecture (from the AIICS division), open source project and using available API services provided by Boston Dynamics. The pre-existing architecture provides a foundation for integrating a ROS2 automated planner into the delegation system. This was done by sending requests containing a TST to the Executor Factory for each action of the plan. The Spot ROS2 open-source project allowed for a modular development approach, while the API services enabled control over the robot.

**2. How can an automated planner be used to achieve the following autonomous behaviours?**

- Navigate to a specific position, find a specific object (such as a first-aid kit), pick it up and move to another location to deliver that object.
- Navigate to multiple positions and collect the object at each location.

In order to achieve desired goals, a plan consisting of actions was generated using an automated planner. The planner was provided with the domain definition and problem specification. The domain defines actions the robot can do, and predicates represent

---

properties of the robot or objects of interest that can be true or false. A problem is defined as objects that exist in the world relevant to the planning (locations, packages, the robot itself), the initial state of the robot and the goal specification. The goal specification includes which locations must have a package delivered to and which packages must be in the container mounted on robot's body.

To conclude, this thesis provides a groundwork for developing the Boston Dynamics Spot robot as a reasoning agent. During experimental scenarios it was able to operate independently and plan in a given environment to achieve its designed goals. Its actions are autonomous and flexible, meaning it can adapt to changing circumstances and make decisions based on its own internal state.

## Future Work

As previously stated, this thesis serves as a foundational basis. There is still room for improvement in order to enhance the robot's reliability.

This work consists of many components, integration into the software framework developed by the AIICS division, planning and control of the robot. Therefore the possibilities for improvements are plenty. The following are future work suggestions that can be made for each component.

The Task Specification Tree (TST) for some of the robot's actions can be detailed further, making them more consistent with the software framework developed by the AIICS division. An example is the action *pick-up*, on the control level the action consists of finding the desired target, walking to it and then picking it up. These certainly could have been separated into different TST nodes. However, the API services used in the action *pick-up* need to use the image in which the desired target was detected in and therefore requires more time to solve how to compose them in a TST in the most appropriate way. If this is achieved, in the future, with multiple robots involved, each could have different roles, such as locating desired targets, saving images of the target, and sharing information for *pick-up* and *delivery* tasks.

Currently, in the planning stage, it is assumed that the agent's actions will always have the intended outcome. This oversimplifies the reality, as it does not account for the possibility of the agent failing to find or pick up the desired object. A possible solution is to add a replanning feature to the planning module in case an action fails.

Finally, for the control part, the robot now uses images from the body's cameras. However, it is also equipped with a higher-resolution camera on the gripper. One possible enhancement could be to use this camera to search for the object, which could also solve the problem encountered in the *pick-up* action.

# A Domain Definitions

This appendix contains the initial and the final domains definitions for the planner used in this study.

## A.1 Initial Domain

```
1 ;; Domain definition for the Spot robot
2 (define (domain spot)
3   (:requirements :strips :typing :durative-actions)
4   (:types
5     pose
6     spot package - object
7   )
8
9   (:predicates
10    (is-localized ?spot - spot)
11    (standing ?spot - spot)
12    (at ?obj - object ?pos - pose)
13    (power-on ?spot - spot)
14    (motor-on ?spot - spot)
15    (sitting ?spot - spot)
16    (pkg-delivered ?pos - pose)
17    (visited ?spot - spot ?pos - pose)
18    (carrying ?pkg - package)
19    (not-delivered ?pkg - package)
20    (in-bucket ?pkg - package)
21    (picked-up-at ?pkg - package ?pos - pose)
22    (docked ?spot - spot ?loc - pose)
23    (undocked ?spot - spot)
24  )
25 )
26
27 ;; Power the robot motor
28 (:durative-action power-motor
29   :parameters (?spot - spot)
```

```

30      :duration (= ?duration 10)
31      :condition (and
32          (over all (power-on ?spot))
33          (over all (sitting ?spot))
34      )
35      :effect (and
36          (at end (motor-on ?spot))
37      )
38  )
39
40  ;; Dock the Spot ?spot to the dock-station at location ?loc
41  (:durative-action dock
42      :parameters (?spot - spot ?loc - pose)
43      :duration (= ?duration 10)
44      :condition (and
45          (over all (motor-on ?spot))
46          (at start (standing ?spot))
47          (at start (undocked ?spot))
48          (over all (at ?spot ?loc))
49      )
50      :effect (and
51          (at end (docked ?spot ?loc))
52          (at end (not (undocked ?spot)))
53          (at end (sitting ?spot))
54          (at end (not (standing ?spot)))
55          (at end (at ?spot ?loc))
56      )
57  )
58
59  ;; Undock the Spot ?spot from the dock-station at location ?loc
60  (:durative-action undock
61      :parameters (?spot - spot ?loc - pose)
62      :duration (= ?duration 20)
63      :condition (and
64          (over all (motor-on ?spot))
65          (at start (docked ?spot ?loc))
66          (at start (sitting ?spot))
67      )
68      :effect (and
69          (at end (not (docked ?spot ?loc)))
70          (at end (undocked ?spot))
71          (at end (standing ?spot))
72          (at end (not (sitting ?spot)))
73      )
74  )
75
76  ;; Deliver the package ?pkg at the location ?pos
77  (:durative-action deliver-pkg
78      :parameters (?spot - spot ?pkg - package ?pos - pose)
79      :duration (= ?duration 25)
80      :condition (and
81          (at start (carrying ?pkg))
82          (over all (at ?spot ?pos))
83          (over all (motor-on ?spot))
84          (over all (standing ?spot))
85          (over all (not-delivered ?pkg))
86          (over all (undocked ?spot))

```

```

87     )
88     :effect (and
89         (at end (at ?pkg ?pos))
90         (at end (pkg-delivered ?pos))
91         (at end (not (carrying ?pkg)))
92         (at end (not (not-delivered ?pkg)))
93     )
94 )
95
96 ;; Add the package ?pkg at the location ?pos to the container
97 ;; on the Spot ?spot
98 (:durative-action add-to-bucket
99     :parameters (?spot - spot ?pkg - package ?pos - pose)
100    :duration (= ?duration 15)
101    :condition (and
102        (at start (carrying ?pkg))
103        (over all (motor-on ?spot))
104        (over all (standing ?spot))
105        (over all (not-delivered ?pkg))
106        (over all (at ?spot ?pos))
107        (over all (picked-up-at ?pkg ?pos))
108        (over all (undocked ?spot))
109    )
110    :effect (and
111        (at end (not (carrying ?pkg)))
112        (at end (in-bucket ?pkg))
113        (at end (not (not-delivered ?pkg)))
114    )
115 )
116
117 ;; Pick-up the package ?pkg at the location ?pose
118 (:durative-action pick-up
119     :parameters (?spot - spot ?pkg - package ?pos - pose)
120    :duration (= ?duration 15)
121    :condition (and
122        (over all (at ?spot ?pos))
123        (at start (at ?pkg ?pos))
124        (over all (standing ?spot))
125        (over all (motor-on ?spot))
126        (over all (undocked ?spot))
127    )
128    :effect (and
129        (at start (not (at ?pkg ?pos)))
130        (at end (carrying ?pkg))
131        (at end (picked-up-at ?pkg ?pos))
132    )
133 )
134
135 ;; Stand the Spot ?spot up
136 (:durative-action stand-up
137     :parameters (?spot - spot)
138    :duration (= ?duration 5)
139    :condition (and
140        (over all (motor-on ?spot))
141        (at start (sitting ?spot))
142        (over all (undocked ?spot))
143    )

```

```

144         :effect (and
145             (at end (standing ?spot))
146             (at start (not (sitting ?spot)))
147         )
148     )
149
150 ;; Sit the Spot ?spot down
151 (:durative-action sit
152     :parameters (?spot - spot)
153     :duration (= ?duration 5)
154     :condition (and
155         (over all (motor-on ?spot))
156         (at start (standing ?spot))
157         (over all (undocked ?spot))
158     )
159     :effect (and
160         (at start (not (standing ?spot)))
161         (at end (sitting ?spot))
162     )
163 )
164
165
166 ;; Walk from ?from location to ?to location
167 (:durative-action walk
168     :parameters (?spot - spot ?from - pose ?to - pose)
169     :duration (= ?duration 60)
170     :condition (and
171         (over all (motor-on ?spot))
172         (over all (is-localized ?spot))
173         (at start (standing ?spot))
174         (at start (at ?spot ?from))
175         (over all (undocked ?spot))
176     )
177     :effect (and
178         (at start (not (standing ?spot)))
179         (at start (not (at ?spot ?from)))
180         (at end (standing ?spot))
181         (at end (at ?spot ?to))
182         (at end (visited ?spot ?to))
183     )
184 )
185
186 ;; Localize the Spot ?spot
187 (:durative-action localize
188     :parameters (?spot - spot)
189     :duration (= ?duration 5)
190     :condition (and
191         (over all (motor-on ?spot))
192         (over all (standing ?spot))
193     )
194     :effect (and
195         (at end (and (is-localized ?spot)))
196     )
197 )
198
199 )

```

## A.2 Final Domain

```

1 ;; Domain definition for the Spot robot
2 (define (domain spot)
3   (:requirements :strips :typing :durative-actions)
4   (:types
5     pose
6     spot package - object
7   )
8
9   (:predicates
10    (is-localized ?spot - spot)
11    (standing ?spot - spot)
12    (at ?obj - object ?pos - pose)
13    (power-on ?spot - spot)
14    (motor-on ?spot - spot)
15    (sitting ?spot - spot)
16    (pkg-delivered ?pos - pose)
17    (visited ?spot - spot ?pos - pose)
18    (carrying ?pkg - package)
19    (not-delivered ?pkg - package)
20    (in-bucket ?pkg - package)
21    (picked-up-at ?pkg - package ?pos - pose)
22    (docked ?spot - spot ?loc - pose)
23    (undocked ?spot - spot)
24  )
25 )
26
27 ;; Power the robot motor
28 (:durative-action power-motor
29   :parameters (?spot - spot)
30   :duration (= ?duration 10)
31   :condition (and
32     (over all (power-on ?spot))
33     (over all (sitting ?spot))
34   )
35   :effect (and
36     (at end (motor-on ?spot))
37   )
38 )
39
40 ;; Dock the Spot ?spot to the dock-station at location ?loc
41 (:durative-action dock
42   :parameters (?spot - spot ?loc - pose)
43   :duration (= ?duration 10)
44   :condition (and
45     (over all (motor-on ?spot))
46     (at start (standing ?spot))
47     (at start (undocked ?spot))
48     (over all (at ?spot ?loc))
49   )
50   :effect (and
51     (at end (docked ?spot ?loc))
52     (at end (not (undocked ?spot)))
53     (at end (sitting ?spot))
54     (at end (not (standing ?spot)))
55     (at end (at ?spot ?loc))

```

```

56     )
57   )
58
59   ;; Undock the Spot ?spot from the dock-station at location ?loc
60   (:durative-action undock
61     :parameters (?spot - spot ?loc - pose)
62     :duration (= ?duration 20)
63     :condition (and
64       (over all (motor-on ?spot))
65       (at start (docked ?spot ?loc))
66       (at start (sitting ?spot))
67     )
68     :effect (and
69       (at end (not (docked ?spot ?loc)))
70       (at end (undocked ?spot))
71       (at end (standing ?spot))
72       (at end (not (sitting ?spot)))
73     )
74   )
75
76   ;; Deliver the package ?pkg at the location ?pos
77   (:durative-action deliver-pkg
78     :parameters (?spot - spot ?pkg - package ?pos - pose)
79     :duration (= ?duration 1)
80     :condition (and
81       (at start (carrying ?pkg))
82       (over all (at ?spot ?pos))
83       (over all (motor-on ?spot))
84       (over all (standing ?spot))
85       (over all (not-delivered ?pkg))
86       (over all (undocked ?spot))
87     )
88     :effect (and
89       (at end (at ?pkg ?pos))
90       (at end (pkg-delivered ?pos))
91       (at end (not (carrying ?pkg)))
92       (at end (not (not-delivered ?pkg)))
93     )
94   )
95
96   ;; Add the package ?pkg at the location ?pos to the container
97   ;; on the Spot ?spot
98   (:durative-action add-to-bucket
99     :parameters (?spot - spot ?pkg - package ?pos - pose)
100    :duration (= ?duration 1)
101    :condition (and
102      (at start (carrying ?pkg))
103      (over all (motor-on ?spot))
104      (over all (standing ?spot))
105      (over all (not-delivered ?pkg))
106      (over all (at ?spot ?pos))
107      (over all (picked-up-at ?pkg ?pos))
108      (over all (undocked ?spot))
109    )
110    :effect (and
111      (at end (not (carrying ?pkg)))
112      (at end (in-bucket ?pkg))

```

```

113         (at end (not (not-delivered ?pkg)))
114     )
115 )
116
117 ;; Pick-up the package ?pkg at the location ?pose
118 (:durative-action pick-up
119   :parameters (?spot - spot ?pkg - package ?pos - pose)
120   :duration (= ?duration 15)
121   :condition (and
122     (over all (at ?spot ?pos))
123     (at start (at ?pkg ?pos))
124     (over all (standing ?spot))
125     (over all (motor-on ?spot))
126     (over all (undocked ?spot))
127   )
128   :effect (and
129     (at start (not (at ?pkg ?pos)))
130     (at end (carrying ?pkg))
131     (at end (picked-up-at ?pkg ?pos))
132   )
133 )
134
135 ;; Stand the Spot ?spot up
136 (:durative-action stand-up
137   :parameters (?spot - spot)
138   :duration (= ?duration 5)
139   :condition (and
140     (over all (motor-on ?spot))
141     (at start (sitting ?spot))
142     (over all (undocked ?spot))
143   )
144   :effect (and
145     (at end (standing ?spot))
146     (at start (not (sitting ?spot)))
147   )
148 )
149
150 ;; Sit the Spot ?spot down
151 (:durative-action sit
152   :parameters (?spot - spot)
153   :duration (= ?duration 5)
154   :condition (and
155     (over all (motor-on ?spot))
156     (at start (standing ?spot))
157     (over all (undocked ?spot))
158   )
159   :effect (and
160     (at start (not (standing ?spot)))
161     (at end (sitting ?spot))
162   )
163 )
164 )
165
166
167 ;; Walk from ?from location to ?to location
168 (:durative-action walk
169   :parameters (?spot - spot ?from - pose ?to - pose)

```

```

170     :duration (= ?duration 100)
171     :condition (and
172         (over all (motor-on ?spot))
173         (over all (is-localized ?spot))
174         (at start (standing ?spot))
175         (at start (at ?spot ?from))
176         (over all (undocked ?spot))
177     )
178     :effect (and
179         (at start (not (standing ?spot)))
180         (at start (not (at ?spot ?from)))
181         (at end (standing ?spot))
182         (at end (at ?spot ?to))
183         (at end (visited ?spot ?to))
184     )
185 )
186
187 ;; Localize the Spot ?spot
188 (:durative-action localize
189     :parameters (?spot - spot)
190     :duration (= ?duration 5)
191     :condition (and
192         (over all (motor-on ?spot))
193         (over all (standing ?spot))
194     )
195     :effect (and
196         (at end (and (is-localized ?spot)))
197     )
198 )
199
200 )

```



## Bibliography

- [1] Olov Andersson, Patrick Doherty, Mårten Lager, Jens-Olof Lindh, Linnea Persson, Elin A Topp, Jesper Tordenlid, and Bo Wahlberg. “WARA-PS: a research arena for public safety demonstrations and autonomous collaborative rescue robotics experimentation”. In: *Autonomous Intelligent Systems 1* (2021), pp. 1–31.
- [2] André Arnold and Damian Niwiński. “Complete lattices and fixed-point theorems”. In: *Rudiments of  $\mu$ -calculus* 146 (2001), pp. 1–39.
- [3] Aurelio Cappozzo, Ugo Della Croce, Alberto Leardini, and Lorenzo Chiari. “Human movement analysis using stereophotogrammetry: Part 1: theoretical background”. In: *Gait & posture* 21.2 (2005), pp. 186–196.
- [4] Michael Cashmore, Maria Fox, Tom Larkworthy, Derek Long, and Daniele Magazzeni. “AUV mission control via temporal planning”. In: *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, pp. 6535–6541.
- [5] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. “Rosplan: Planning in the robot operating system”. In: *Proceedings of the international conference on automated planning and scheduling*. Vol. 25. 2015, pp. 333–341.
- [6] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. “Forward-chaining partial-order planning”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 20. 2010, pp. 42–49.
- [7] Michele Colledanchise, Alejandro Marzinotto, Dimos V Dimarogonas, and Petter Ögren. “The advantages of using behavior trees in multi-robot systems”. In: *Proceedings of ISR 2016: 47th International Symposium on Robotics*. VDE. 2016, pp. 1–8.
- [8] Michele Colledanchise and Petter Ögren. “How behavior trees modularize robustness and safety in hybrid systems”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 1482–1488.
- [9] Frank Dellaert, Michael Kaess, et al. “Factor graphs for robot perception”. In: *Foundations and Trends® in Robotics* 6.1-2 (2017), pp. 1–139.
- [10] Patrick Doherty, Joakim Gustafsson, Lars Karlsson, and Jonas Kvarnström. *Tal: Temporal action logic language specification and tutorial*. Linköping University Electronic Press, 1998.

- [11] Patrick Doherty, Fredrik Heintz, and Jonas Kvarnström. "High-level mission specification and planning for collaborative unmanned aircraft systems using delegation". In: *Unmanned Systems* 1.01 (2013), pp. 75–119.
- [12] Patrick Doherty, Fredrik Heintz, and David Landén. "A distributed task specification language for mixed-initiative delegation". In: *Principles and Practice of Multi-Agent Systems: 13th International Conference, PRIMA 2010, Kolkata, India, November 12-15, 2010, Revised Selected Papers* 13. Springer. 2012, pp. 42–57.
- [13] Patrick Doherty and Jonas Kvarnström. "Temporal action logics". In: *Foundations of Artificial Intelligence* 3 (2008), pp. 709–757.
- [14] Patrick Doherty, Jonas Kvarnström, Piotr Rudol, Marius Wzorek, Gianpaolo Conte, Cyrille Berger, Timo Hinzmänn, and Thomas Stastny. "A collaborative framework for 3d mapping using unmanned aerial vehicles". In: *International Conference on Principles and Practice of Multi-Agent Systems*. Springer. 2016, pp. 110–130.
- [15] Patrick Doherty, Jonas Kvarnström, and Andrzej Szalas. "Temporal composite actions with constraints". In: *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*. 2012.
- [16] Patrick Doherty, Jonas Kvarnström, Mariusz Wzorek, Piotr Rudol, Fredrik Heintz, and Gianpaolo Conte. "HDRC3-a distributed hybrid deliberative/reactive architecture for unmanned aircraft systems". In: *Handbook of Unmanned Aerial Vehicles* (2014), pp. 849–952.
- [17] Patrick Doherty and John-Jules Ch Meyer. "On the logic of delegation-relating theory and practice". In: (2012).
- [18] Guillaume Ducard and Raffaello D’Andrea. "Autonomous quadrotor flight using a vision system and accommodating frames misalignment". In: *2009 IEEE International Symposium on Industrial Embedded Systems*. IEEE. 2009, pp. 261–264.
- [19] Boston Dynamics. *About Spot*. 2023. URL: [https://dev.bostondynamics.com/docs/concepts/about\\_spot](https://dev.bostondynamics.com/docs/concepts/about_spot). (accessed: 03.03.2023).
- [20] Boston Dynamics. *Arm and gripper specifications*. 2023. URL: [https://dev.bostondynamics.com/docs/concepts/arm/arm\\_specification](https://dev.bostondynamics.com/docs/concepts/arm/arm_specification). (accessed: 03.03.2023).
- [21] Boston Dynamics. *Geometry and Frames*. 2023. URL: [https://dev.bostondynamics.com/docs/concepts/geometry\\_and\\_frames](https://dev.bostondynamics.com/docs/concepts/geometry_and_frames). (accessed: 03.03.2023).
- [22] Boston Dynamics. *GraphNav Map Structure*. 2023. URL: [https://dev.bostondynamics.com/docs/concepts/autonomy/graphnav\\_map\\_structure](https://dev.bostondynamics.com/docs/concepts/autonomy/graphnav_map_structure). (accessed: 03.03.2023).
- [23] Boston Dynamics. *Machine Learning Bridge and External Compute*. 2023. URL: [https://dev.bostondynamics.com/docs/concepts/network\\_compute\\_bridge.html?highlight=network\\_compute\\_bridge](https://dev.bostondynamics.com/docs/concepts/network_compute_bridge.html?highlight=network_compute_bridge). (accessed: 12.04.2023).
- [24] Boston Dynamics. *Networking*. 2023. URL: <https://dev.bostondynamics.com/docs/concepts/networking>. (accessed: 03.03.2023).
- [25] Boston Dynamics. *Persistent Systems Radio*. (accessed: 23.05.2023). 2023. URL: <https://www.bostondynamics.com/sites/default/files/inline-files/persistent-systems-radio-kit.pdf>.
- [26] DJI Enterorise. *Drone Rescues Around The World*. URL: <https://enterprise.dji.com/drone-rescue-map/>. (accessed: 18.01.2023).

- [27] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger. “Using the context-enhanced additive heuristic for temporal and numeric planning”. In: *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*. Springer, 2012, pp. 49–64.
- [28] Maria Fox and Derek Long. “PDDL2.1: An extension to PDDL for expressing temporal planning domains”. In: *Journal of artificial intelligence research* 20 (2003), pp. 61–124.
- [29] Sergio Garcia, Patrizio Pelliccione, Claudio Menghi, Thorsten Berger, and Tomas Bures. “Promise: high-level mission specification for multiple robots”. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*. 2020, pp. 5–8.
- [30] Valentin Goranko and Antje Rumberg. “Temporal logic”. In: *Stanford Encyclopedia of Philosophy* (2020).
- [31] Google Group. *Overview*. 2023. URL: <https://protobuf.dev/overview/>. (accessed: 28.04.2023).
- [32] Ayoung Kim and Ryan M Eustice. “Active visual SLAM for robotic area coverage: Theory and experiment”. In: *The International Journal of Robotics Research* 34.4-5 (2015), pp. 457–475.
- [33] Jonas Kvarnström. “Planning for loosely coupled agents using partial order forward-chaining”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 21. 2011, pp. 138–145.
- [34] Jonas Kvarnström and Patrick Doherty. “Automated planning for collaborative UAV systems”. In: *2010 11th International Conference on Control Automation Robotics & Vision*. IEEE. 2010, pp. 1078–1085.
- [35] David Landén. “Complex Task allocation for delegation: From theory to practice”. PhD thesis. Linköping University Electronic Press, 2011.
- [36] David M Lane, Francesco Maurelli, Petar Kormushev, Marc Carreras, Maria Fox, and Konstantinos Kyriakopoulos. “PANDORA-persistent autonomy through learning, adaptation, observation and replanning”. In: *IFAC-PapersOnLine* 48.2 (2015), pp. 238–243.
- [37] Johannes Löhr, Patrick Eyerich, Thomas Keller, and Bernhard Nebel. “A planning based framework for controlling hybrid systems”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 22. 2012, pp. 164–171.
- [38] Lorenzo Marconi, Claudio Melchiorri, Michael Beetz, Dejan Pangercic, Roland Siegwart, Stefan Leutenegger, Raffaella Carloni, Stefano Stramigioli, Herman Bruyninckx, Patrick Doherty, et al. “The SHERPA project: Smart collaboration between humans and ground-aerial robots for improving rescuing activities in alpine environments”. In: *2012 IEEE international symposium on safety, security, and rescue robotics (SSRR)*. IEEE. 2012, pp. 1–4.
- [39] Francisco Martín, Jonatan Ginés, Francisco J. Rodríguez, and Vicente Matellán. “Plan-Sys2: A Planning System Framework for ROS2”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - October 1, 2021*. IEEE, 2021.
- [40] Ernesto Martinez. *Chock (in Swedish)*. 2023. URL: <https://www.1177.se/Ostergotland/olyckor--skador/akuta-rad---forsta-hjalpen/chock/>. (accessed: 13.05.2023).
- [41] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. “PDDL-the planning domain definition language.(1998)”. In: *Cited on* (1998), p. 19.

- [42] Daniel Mellinger, Nathan Michael, and Vijay Kumar. "Trajectory generation and control for precise aggressive maneuvers with quadrotors". In: *The International Journal of Robotics Research* 31.5 (2012), pp. 664–674.
- [43] Daniel Mitchell, Jamie Blanche, Osama Zaki, Joshua Roe, Leo Kong, Samuel Harper, Valentin Robu, Theodore Lim, and David Flynn. "Symbiotic system of systems design for safe and resilient autonomous robotics in offshore wind farms". In: *IEEE Access* 9 (2021), pp. 141421–141452.
- [44] Elias Mueggler, Basil Huber, and Davide Scaramuzza. "Event-based, 6-DOF pose tracking for high-speed maneuvers". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 2761–2768.
- [45] Richard Nordquist. *Speech Act Theory*. 2023. URL: <https://www.thoughtco.com/speech-act-theory-1691986>. (accessed: 25.05.2023).
- [46] Chiara Piacentini, Varvara Alimisis, Maria Fox, and Derek Long. "Combining a temporal planner with an external solver for the power balancing problem in an electricity network". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 23. 2013, pp. 398–406.
- [47] Open Robotics. *ROS2 Documentation*. 2023. URL: <http://docs.ros.org/en/humble/index.html>. (accessed: 04.03.2023).
- [48] ROS. *ROS Documentation*. 2023. URL: <https://wiki.ros.org>. (accessed: 04.03.2023).
- [49] Moritz Tenorth and Michael Beetz. "KnowRob: A knowledge processing infrastructure for cognition-enabled robots". In: *The International Journal of Robotics Research* 32.5 (2013), pp. 566–590.
- [50] The Intelligent Motion Lab at University of Illinois. *3D Rotations*. 2023. URL: <http://motion.cs.illinois.edu/RoboticSystems/3DRotations.html>. (accessed: 20.04.2023).
- [51] The Intelligent Motion Lab at University of Illinois. *Transformations*. 2023. URL: <http://motion.cs.illinois.edu/RoboticSystems/CoordinateTransformations.html>. (accessed: 03.03.2023).
- [52] Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. "Compiling quantum circuits to realistic hardware architectures using temporal planners". In: *Quantum Science and Technology* 3.2 (2018).
- [53] Thomas Whelan, Michael Kaess, John J Leonard, and John McDonald. "Deformation-based loop closure for large scale dense RGB-D SLAM". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 548–555.