Enabling Semantically Grounded, Long Horizon Planning and Execution for Autonomous Agents

by

Lucian Covarrubias

S.B. in Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2023

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2025

Authored by:	Lucian Covarrubias The Department of Electrical Engineering and Computer Science Jan 17, 2025
Certified by:	Brian C. Williams Professor of Aeronautics and Astronautics, Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

Enabling Semantically Grounded, Long Horizon Planning and Execution for Autonomous Agents

by

Lucian Covarrubias

Submitted to the Department of Electrical Engineering and Computer Science
on Jan 17, 2025 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

Robots have been playing an ever increasing role in complex environments, often in coordination with teams of systems or humans. Autonomous systems of the future will need to be tightly grounded in the real world, drawing information directly from their environment to develop an understanding of the world. They will need to maintain a semantic understanding of their environment, including the kinds of objects they observe and their relationships to each other. At the same time, they must be able to reason over diverse constraints related to their tasks, such as time limits and resource usage. While there are existing approaches which enable robots to execute tasks with semantic goals, such as finding a certain type of object in a room, they often fail to consider the multitude fo task specific constraints which are vital to robust performance. On the other hand, planners which consider task specific constraints require a human to provide all information about the environment manually. These systems are too cumbersome to model complex tasks, requiring hours of manual effort which is prone to errors. This thesis presents an architecture for semantically grounded planning which leverages the strengths of constraint based planners while automating the environmental modeling step with an advanced semantic perception engine. By automating environmental modeling, we are able to create a system which executes complex semantically grounded tasks such as navigating to certain objects within a certain room, without major user input which is typically required of these systems.

Thesis supervisor: Brian C. Williams
Title: Professor of Aeronautics and Astronautics

Acknowledgments

This program has been an incredible opportunity to explore, reflect, and grow. Throughout the highs and lows, I am so thankful to the people who supported and encouraged me to make this journey a reality. This would not have been possible without them.

Firstly, I would like to thank my advisor, Brian Williams, for his engagement, intuition and unwavering support. From early ideations to the final touches on this thesis, Brian has always been there for me to support my vision. In the midst of everything that I had to manage, Brian was always a source of calm wisdom and passionate thinking. He taught me to always think about the end goal, while building up a strong base to get me there. I truly couldn't have done this without him, and will be forever thankful.

I'm also grateful to my lab mates Jake Olkin, Aneesh Jois, Shashank Swaminathan, Marlyse Reeves, Viraj Parimi, Ingrid Wu, and the MERS squad who were always there to help tackle issues, think through problems, and make me smile. The sense of comradery among us students made me feel welcome and inspired every day. My sanity was upheld by the wonderful people who make MERS what it is, and I will remember them long after I've moved on from MIT.

To my partner, Lizi, who has always been there for me. Whenever life threw me a curve ball or I was uncertain about the future, she always provided me with comfort, stability, and thoughtful advice. I will never forget how many times she stepped up to make our lives work together despite the roller coaster of my MEng, and I can't wait to see what we can build together in our future.

To my sister, Hayley, who has always been there for me in times of duress. Her pragmatic wisdom and confident advice has been my rock in the most uncertain of times. She has guided me from the day I was born into this world, and I'm certain that I would not have completed this program, much less attended MIT without her guiding me through life.

To the friends I've made throughout my time here, you are the ones who made MIT the most transformative experience of my life. From playful banter to deep philosophical discourse, I was able to explore who I truly was and what I believed in. I've never felt more at home than when I'm with you, and I will nurture these friendships for the rest of my life.

The community, encouragement, and inspiration I've experienced from you all has made me who I am today, and I will be forever grateful. Thank you.

The work in this thesis was supported by BP. I am grateful to them for giving me the opportunity to pursue this research.

Contents

Title page 1

Abstract 3

Acknowledgments 5

List of Figures 9

1 Introduction 11

2 Semantically Grounded Planning and Execution 15
- 2.1 Semantically Grounded Problems . . . 16
- 2.2 Long Horizon Planning . . . 17
- 2.3 Hierarchical Semantic SLAM . . . 18

3 Supporting Technology 19
- 3.1 Activity Planner . . . 19
  - 3.1.1 Representation Language . . . 22
- 3.2 Plan Execution . . . 25
- 3.3 Semantic Mapping . . . 27

4 Environment Translation 31
- 4.1 Command Verification . . . 32
- 4.2 Activity Planner Domain Specification . . . 32
  - 4.2.1 A Note on Room Classification . . . 35
- 4.3 Plan Executive Domain Specification . . . 35

5 Scaling to Large Maps 39

6 Experiments and Results 43
- 6.1 Demonstration . . . 43
  - 6.1.1 Office Environment Specification . . . 43
  - 6.1.2 Watering Plants . . . 44
- 6.2 Future Work . . . 46
  - 6.2.1 Initial Domain Specification . . . 46
  - 6.2.2 Object Representations in Kirk . . . 47

6.2.3 Room Classification 48
6.2.4 Planning From Goals via PDDL Planners 48
6.3 Conclusion 48
6.3.1 Semantic Mapping 49
6.3.2 Activity Planning 49
6.3.3 Plan Execution 49
6.3.4 Automatic Semantic Translation 50

A Appendix **51**

References **55**

List of Figures

3.1 A variety of different TPN structures. Top is two actions in sequence, bottom left is two actions that can be completed in parallel, bottom right is two actions with a non-deterministic choice to perform one or the other. 20
3.2 TPN with an open constraint. The original TPN(top) is updated by Kirk to contain a causal link defining the implied constraint of the TPN(bottom). . . 21
3.3 QSP describing a simple mission. Rover 1 is tasked with action A while Rover 2 completes action B in parallel. 22
3.4 Dynamic Scene Graph of Office Environment 28

4.1 Data Flow Block Diagram of Domain Specification Modules 31
4.2 2D map of the office environment before any mesh triangles have been filtered out. This map contains over 600,000 individual triangles. 37
4.3 2D map of the office environment after filtering. This map contains roughly 28,000 triangles. 37
4.4 Visualization of complete Magellan model. Obstacles are in black while interest regions are in green. 38

5.1 Simple example of corner cutting. While each time step (denoted with an 'x') is inside of the safe regions (blue), the actual trajectory crosses through the obstacle (gray). 41
5.2 Simple example which avoids corner cutting. The agent is forced to transition between safe regions (light blue) through the intersections (darker blue). This avoids any cases where the agent might accidentally pass through an obstacle. 42

6.1 Visualization of complete Magellan model for the plant watering task. Obstacles are black, safe regions are blue, and locations of various plants are green. The agent's start location is in pink. 45
6.2 Visualization of plant watering trajectories, simplified for clarity. Left is the trajectory when only watering one plant, right is the trajectory when watering two plants. Solid lines connect locations for each event in the plan, while dashed lines track the MPC trajectory with vehicle dynamics. 46

A.1 Visualization of full plant watering trajectory. 52
A.2 Visualization of safe regions generated by IRIS in the office environment. . . 53

Chapter 1

Introduction

Robots have been playing an ever increasing role in complex environments, often in coordination with teams of systems or humans. It is expected that in the future, robots will need to be highly autonomous and maintain an understanding of objects in the environment in order to complete tasks. This could be moving supplies to a certain room in a building, retrieving specific objects, or searching for survivors in emergency situations. We can define objectives which require a semantic understanding of the environment as a semantically grounded goal. For example, if we want an agent to bring us the chair next to the door, the agent needs to understand what a chair and door is, and what is means to be next to something. In order to understand a semantically grounded goal, an autonomous agent must be able to extract semantic information from its surroundings and maintain an internal representation of relationships between objects that it observes.

A core example of this work focuses on tasks that may be assigned to a robotic office custodian. We can take the following mission which highlights the key components of such a task.

Example 1. *The agent must water all of the plants which lie specifically in the hallways of the building in under an hour. The agent begins at a start location, and must plan a path to move to and water all relevant plants in the building. The agent must avoid obstacles in the environment for the entire duration of the mission. In this setting, we can assume that the office has been mapped out prior to planning, so there is no exploration necessary to locate objects or locations of interest.*

As custodial agents will be expected to complete many tasks in the same environment, users should be able to query the agent for a variety of tasks without extensive manual input. A user should be able to provide a simple description of a mission, and the agent should be able to plan and execute the mission or state that the mission is impossible. Ideally, an

agent should be initialized with a detailed map of the environment, which it can then access in order to derive all relevant mission information which the user doesn't provide. The user will then only be providing information which cannot be gleaned from the environment such as time constraints and mission objectives.

We would like to create a system which enables agents to reason over three categories of constraints in order to complete the above task. Information from the environment, such as the locations of objects of interest and obstacles, will affect navigation and task planning. System constraints, such as battery life and vehicle dynamics, will determine local trajectories and long horizon plans. Task restrictions, such as time constraints, objectives, and risk bounds, provide additional structure to the problem. Adherence to these three constraint categories is necessary if we want agents to reliably complete a variety of tasks.

There are many methods which address the first reasoning requirement, and are able to utilize environmental information to accomplish a variety of tasks (see [1] for a survey of approaches). Certain approaches allow for autonomous semantic search, where an agent searches for a target object in a new environment. For example, these methods may allow an agent to find a pair of scissors in an unmapped room. One such technique leverages a Partially Observable Markov Decision Process (POMDP), and rolls out future trajectories in order to maximize new information gleaned by future movements [2]. Other techniques attempt to infer the highest probability regions where the target object may lie [3], [4]. There are also techniques which directly learn a mapping from image space to actions in order to locate objects in unknown environments [5]. What these techniques fail to incorporate are the possible failure modes for a full plan, such as not completing tasks in the allotted time or running out of resources before completion. For some of the methods ([2]–[4]), the agent also must be retrained before being applied to a new environment or target object. It's clear that these types of techniques will be insufficient for agents which must adhere to all three constraint categories.

In the planning community, there are relatively mature constraint based planners which succeed in problems where timing, resource consumption, and mission objectives are critical, enabling agents to reason over the second and third categories of constraints [6]–[8]. These approaches utilize task and internal constraints as part of their planning and execution process, enabling robust agents which can complete complex missions. While these techniques satisfy the need to adhere to mission priorities and various constraints, they cannot directly access information from the environment to develop a rich understanding of the world. Domain models, such as action models and environmental structures, are highly abstracted and manually constructed for these planners, making it very cumbersome to specify missions related to arbitrary semantic goals within a complex environment. For example, we

can order such a system to navigate to a specified location denoted by a convex region, but cannot tell the system to navigate to a plant without expressly telling it where the desired plant is. If we would like an agent to water all plants in a building, we need to manually search the building for plants and specify their exact positions with bounding boxes. The abstracted bounding boxes are then passed to the planner, making planning over semantic goals extremely tedious.

Constraint based planners get us very close to meeting all the requirements we mapped out earlier. If we can construct a way to automatically convert information from the environment into representation which understands object types and their relationships to each other, we can pass this information to the planners to enable robust, semantically grounded planning. Conveniently, there exist perception systems which can generate 3D maps of an environment with semantic labels and relationships between objects in real time [9], [10]. While such information wouldn't help to automatically define action models, it has the potential to completely automate the environmental modeling for a task. Despite the potential for advanced planning and perception approaches, there is limited work in planning over these structures [11]. It is clear that the current state of the art is unable to satisfy the need for constraint based planning for semantically grounded tasks.

In this thesis, we develop an architecture which is able to specify, plan, and execute a variety of semantically grounded missions with minimal user input. The approach relies on a constraint based activity planner to generate a series of actions with dispatch times [6], a constraint based plan executive to generate trajectories and monitor agent progress during plan execution [7], and a semantic perception engine to provide the detailed map information necessary to understand semantic goals [9]. To fuse the three technologies together, I construct a series of modules which translate information from the perception system into environment and mission models for the planner and executive. The integrated system can take in a simple mission statement from the user, then automatically specify detailed inputs for the planner and executive which are based on the map generated by the perception engine. These inputs can then be passed directly to their corresponding planning or execution modules, enabling robust planning over a wide variety of semantic goals.

The rest of the thesis is organized as follows. In chapter 2 I will further specify the semantically grounded planning problem, and present the state of the art in solving these types of problems. In chapter 3, I will detail the proposed architecture to enable robust planning and execution over semantic goals. Chapter 4 describes the automatic environmental translation process. In chapter 5, I will discuss optimizations made to the executive to plan over large maps. In chapter 6, I will describe the experimental procedure used to verify the proposed system, and discuss avenues for further research based on discoveries made during the course

of this thesis work.

Chapter 2

Semantically Grounded Planning and Execution

There is a rich reservoir of literature regarding the concepts that guide this paper. In section 2.1, I will define the general category of semantically grounded problems, and present a subset of methods which can solve these problems to varying degrees of sophistication. While there are many ways in which these methods push the boundaries of what is possible, they lack the ability to respect constraints which many long horizon planning approaches are able to incorporate. In section 2.2, I will present some methods for long horizon planning. Long horizon planners are generally better at encoding and respecting complex constraints regarding mission success and plan feasibility, but have been generally developed for use in abstracted environments which don't include a large amount of semantic or perceptive information. Section 2.3 details several cutting edge semantic perception systems, which can generate useful information for planners by combining environment structure with semantic labels. While the maps generated by these systems are very useful for a planner, they have yet to be widely adopted into many classical planning architectures. The approaches presented in this section are powerful in their own respects, but individually have limitations. In order to solve semantically grounded missions with critical constraints, we must use an integrated approach which leverages the unique strengths of each individual system. We need to leverage the robustness of long horizon planners, and feed in semantic information such that we can automatically plan for semantically grounded goals without major user input.

2.1 Semantically Grounded Problems

Semantically grounded problems can describe any kind of task which require an understanding of the environment at a direct, non abstracted level. The task could be to locate a target object in an environment, take measurements from a research site, or move material from one place to another. A formal definition of such a problem is below.

Definition 1. *A **semantically grounded planning and execution problem** is given by $\langle \mathcal{M}, \mathcal{G}, \mathcal{C} \rangle$. $\mathcal{M}$ is a map of the environment, containing obstacle geometry and objects with semantic labels. $\mathcal{G}$ is a semantically grounded goal specification for the mission (I.E. Water all plants in the hallways.). $\mathcal{C}$ is the set of internal and external constraints which must be respected throughout the mission, such as the total mission duration, minimum battery requirements, and internal dynamics of the agent.*

With the problem defined, we can then define a solution to a semantically grounded planning and execution problem.

Definition 2. *A **solution to a semantically grounded planning and execution problem** is given by $\langle \mathcal{P}, \mathcal{T} \rangle$ where $\mathcal{P}$ is the activity plan generated in order to satisfy the goal specification $\mathcal{G}$, and $\mathcal{T}$ is the set of discrete actions and continuous trajectories required for the agent to execute the activity plan while obeying all constraints.*

A key feature of a semantically grounded problem is that an understanding of the semantic meaning of objects or features of the environment is necessary in order to solve the problem. For example, the simple mission presented in 1 requires knowledge of what a plant and hallway is in order to water the plants in the hallways.

As semantically grounded problems are extremely diverse, approaches to solve it have been similarly varied [1]–[5]. For search based problems, some methods model the environment as a Partially Observable Markov Decision Process (POMDP) and use sampling techniques to maximize new information as the agent moves through the environment [2]. This approach relies on forward simulation to model potential roll outs for decisions, and takes the decision which would provide the most new information to the system. While this enables an agent to explore a new environment, it doesn't accommodate specific goals or use mission objectives as a guiding heuristic for complex tasks. In the pre-mapped office environment, where adherence to mission goals and constraints are the primary objective, this method lacks the understanding required to prioritize those objectives. Other techniques are able to accomplish more specific tasks, such as locating an object of interest in an unknown environment, by utilizing probabilistic models [3] or common sense associations

[4] to predict the most likely locations of the target object given recent observations. While a promising technique for active search tasks, these methods also avoid considering mission specific constraints which would often be necessary for long term deployment, such as a time, communication, or resource consumption. The ability for these methods to search for objects in unknown environments is useful for search applications, but in a pre-mapped environment they don't offer much in the way of robust planning or execution.

2.2 Long Horizon Planning

There has been significant work in the planning community regarding how to execute plans with various mission or agent constraints. One approach is able to reason over temporal constraints and make choices among discrete options in order to control multiple agents for long horizon tasks [6].This method, in contrast to other planners which operate directly over state space to find a sequence of actions from start to goal states, plans over a non-deterministic program where the mission has been outlined but not yet scheduled. The method models the mission as a temporal plan network (TPN) and executes a graph search algorithm over the network to find sequences of actions which satisfy the network's constraints. It's important to note that this method can only output a sequence of actions in order to achieve a goal, but cannot perform closed loop execution monitoring or generate agent trajectories to accomplish these tasks. It is assumed that lower level planners will fill the gaps in this architecture. One of these lower level planners is presented in [7], [8], which can robustly execute a plan generated by [6]. This approach uses Model Predictive Control (MPC) to generate motion trajectories and monitors the state of the agent as it executes said trajectories. If the agent deviates from the desired trajectory, the system can re-plan based on the new position in order to adapt to unforeseen dynamics. The plans generated by [7], [8] can consider continuous constraints such as fuel usage and maximum distance, and reason over discrete choices such as the activation of certain sensors at certain times. It is limited by the fact that it has always been used on relatively simple maps with no concept of semantic features which may reside within them, and therefore has not been able to demonstrate usage in semantically grounded problems. With the existing technology a user must manually specify regions of space that contain objects of interest, convert map geometry into a form which the planner can understand, and provide a sketch of the plan with all actions and objects specified. Essentially, the user carries the burden of providing all semantic understanding of the environment, and must manually encode it in a form the planner can understand. A key motivator of this thesis is how to automate this process so that the system can understand the environment without the user passing in large amounts of information. If the system can

autonomously convert information from a detailed map into a planning model, we are one step closer to enabling fully autonomous agents which can respond to semantic goals.

2.3 Hierarchical Semantic SLAM

3D Dynamic Scene graphs [12]–[15] (DSG) have become a very widely accepted data structure in the robotics community to contain semantic and geometric data from an environment. Systems which are capable of generating high quality 3D Scene Graphs in real time have shown incredible promise in recent years [9], [10]. These systems are capable of taking in RGBD and IMU data and computing a 3D scene graph in real time [9] and even fusing maps from multiple agents in order to enable multi-agent semantic SLAM [10]. The information contained within such structures ranges from detailed map geometry to object labels and locations to high level structure of the environment, including a hierarchical graph of rooms, locations, and buildings. DSGs have the potential to drastically increase semantic understanding of the environment for autonomous agents, but there is limited literature on algorithms which are able to plan over these structures to execute semantically grounded plans with rich hierarchical organization. One method for navigation planning over scene graphs is presented in [11]. In this approach, the planner can be queried with specific rooms to navigate to, and will generate a path to the goal via the A* graph search algorithm using the native graph structure of a DSG. As it relies on graph search rather than a more expressive optimization to generate paths, this method is unable to incorporate task specific constraints and doesn't utilize semantically grounded goals in its problem formulation.

Chapter 3

Supporting Technology

Building a semantically grounded TAMP and execution system requires multiple interlocking parts. First, there must be a module capable of generating a series of actions to accomplish a goal which satisfy timing and logic constraints. This module is called the activity planner, and will be discussed in section 3.1. In order to execute the generated plans, we must incorporate a plan executive that dispatches commands to the agent and monitors mission progress. The proposed execution system is discussed in section 3.2. There must also be a system which can take in sensor data and build a semantic map of the environment, discussed in section 3.3. The key component to enable a seamless integration of the above technologies is to automate the domain specification for the planner and plan executive so that all relevant semantic and spatial information is readily accessible without manual effort. Automatic domain specification is detailed in section **??**. As the proposed environments for semantically grounded TAMP are considerably more complex than previous environments used for plan execution, we also must update the execution module to handle large, complex maps effectively. A discussion of methods to increase efficiency in complex environments is presented in 5.

3.1 Activity Planner

The first step to creating an agent which can plan and execute complex tasks is to define an activity planner. Activity planners come in many forms, but generally output a series of actions or events which complete a task or mission. Features of the task that we would like the agent to complete can drastically affect the most effective choice of planner. For an office custodian, we can leverage information about the environment or tasks we want to complete to generate a sketch of events which we need to accomplish. For example, in many tasks we know the general order or outline of sub-tasks that the agent should complete, such as

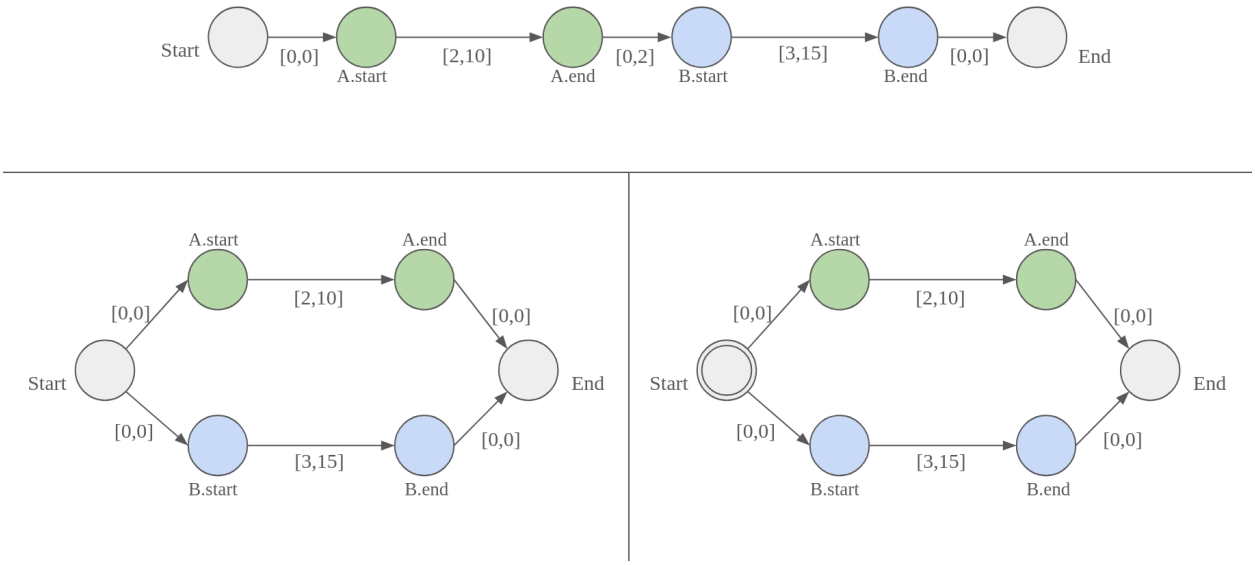

Figure 3.1: A variety of different TPN structures. Top is two actions in sequence, bottom left is two actions that can be completed in parallel, bottom right is two actions with a non-deterministic choice to perform one or the other.

knowing that we want an agent to do the dishes before folding laundry. We often have an idea of how long these tasks can take, which allow us to estimate an upper and lower bound for their duration. This information can be incorporated into the planning approach to boost plan feasibility and reduce solution time. With these features in mind, the Kirk activity planner fits well into this project [6]. The input to Kirk is a domain description which defines every class and action that Kirk can reason over, along with a non-deterministic program that provides a sketch of activities that we would like to complete. This sketch is referred to as a Temporal Planning Network(TPN), a directed, acyclic graph containing information regarding the ordering of events and any temporal and state constraints that are associated with each event. As shown in figure 3.1, a TPN can contain a variety of different structures in order to represent a rich set of mission descriptions. The recursive nature of TPNs ensure that they are capable of representing almost any mission structure by combining those three basic forms.

Kirk generates activity plans by searching through the TPN to find a viable path to the end state that doesn't violate any temporal or state constraints, and tightening any constraints in order to generate a dispatchable set of actions. Notably, Kirk does not generate a series of actions which complete the mission from scratch. Rather, it takes a proposed outline of events and searches for a schedule which satisfies all temporal and logical constraints of the mission. During planning, Kirk is able to recognize implicit constraints (referred to as

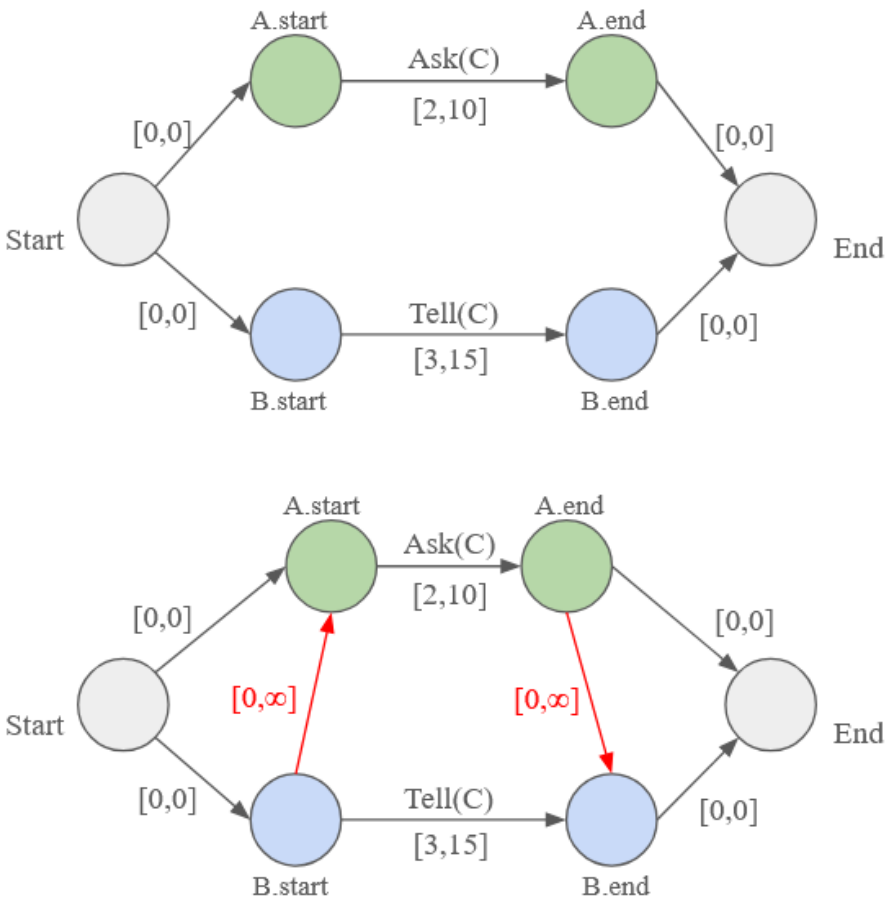

Figure 3.2: TPN with an open constraint. The original TPN(top) is updated by Kirk to contain a causal link defining the implied constraint of the TPN(bottom).

open conditions) and insert them as causal links to ensure plan feasibility[16]. Consider the TPN in figure 3.2. Action A requires that condition C be valid for the duration of the entire action, while action B asserts that condition C be valid for the duration of the entire action. Kirk is able to recognize this open condition and insert the causal link (red) which requires that action A occur within the bounds of action B. Along with open conditions, Kirk can identify threats to plan feasibility where overlapping events have contradictory required conditions. Upon identifying such a threat, Kirk inserts the causal links required to ensure that the contradictory events do not overlap.

Kirk will either produce a schedule of actions that satisfy all constraints or a failure message that indicates the plan cannot be accomplished. When planning is successful, Kirk outputs a series of time windows called episodes and transitions called events where state variables have set values. Events indicate a transition from one episode to another, and the specific action contained within the event determines how the state variables change as the

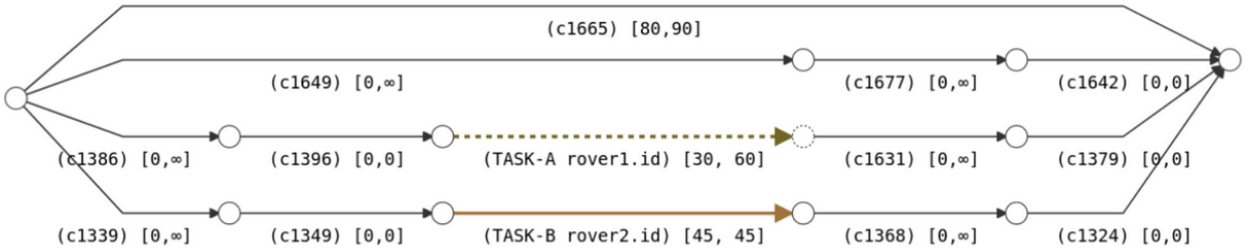

Figure 3.3: QSP describing a simple mission. Rover 1 is tasked with action A while Rover 2 completes action B in parallel.

plan transitions. The combination of episodes and events is known as a Qualitative State Plan (QSP) [7]. In figure 3.3, we can observe a simple QSP detailing a mission where two agents (Rover1 and Rover2) are tasked with completing two actions (A and B) in parallel. Each episode, designated by an arrow in the visual representation, contains the temporal and state constraints that must hold for the duration of the episode. Action A can take between 30 and 60 time units, while action B is fixed at 45 time units.

3.1.1 Representation Language

The key motivation of this thesis is to automate the translation from environmental information into domains which our planner can operate over. The interface through which this translation is interpreted is known as a representation language. A representation language is a set of predicates and actions which describe relationships between various object types, or actions that are possible for a given object. There are two types of representations which are important to any planning process, prior knowledge and situated knowledge. Prior knowledge consists of any information that an agent would have before entering an environment, such as how to move, what types of objects it can understand, and how it can interact with various objects that it might encounter. Situated knowledge describes the current state of the world based on observations. This can also be interpreted as the initialization of abstract concepts into real world entities. We can examine a simple example to better understand the relationship between these two types of knowledge. Consider a universe with an agent and two types of object: blocks and spheres. The prior knowledge in this universe may be that the agent can move, and the agent can pick up and place blocks but not spheres. Upon initialization, the agent may observe that a block exists at a certain location which we can denote as location $\mathcal{A}$. In this case, the situated knowledge is that one block exists and is at

location $\mathcal{A}$.

In the context of this work, we would like to automate the process of generating the situated knowledge for an environment and task. Situated knowledge can be automated by taking in perception data and using it to populate a state of the world based on that data. Prior knowledge requires a domain expert to provide information which is hard or impossible to infer directly from the environment, so would be much harder to automate. Despite these difficulties, see section 6.2 for a discussion on how advancements in Large Language Models may enable autonomous generation of prior knowledge.

To implement a representation language for this project, I elected to use the Reactive Model-Based Programming Language (RMPL) [6] which is able to express a wide variety of object classes and actions between them with requirements and effects. This is a lisp-like language, which encapsulates statements within parentheses to isolate them from the rest of the code. RMPL is supported by Kirk, presenting a more straightforward integration between the semantic perception and planning systems.

RMPL allows for the expression of classes and actions (control programs) in order to define the prior knowledge for an agent. Within classes, a user can define predicates which further specify how the class can interact with other objects or the world. We can analyze some of the classes and actions from the office scenario to ground this discussion. In the office scenario, the class definition for a plant describes the various attributes that a plant can have.

Listing 3.1: Example class definition in RMPL

```
(defclass plant ()
  ((watered
    :initform nil
    :type boolean
    :accessor watered
    :documentation
    "A boolean stating if the plant has been watered.")
   (location
    :initarg :location
    :type integer
    :accessor location
    :documentation
    "The graph location of the plant as an integer.")))
```

As shown, the plant has a location and a boolean attribute `watered` which describes whether

the plant has been watered or not. RMPL is able to effectively define an arbitrary number of classes with arbitrary attributes, allowing for a flexible and robust understanding of the world prior to a mission. Control programs are another essential component of RMPL, and define how various objects can interact with each other or the world. Control programs come in two variations. Primitive control programs, such as the `water` program defined below, require a certain set of predicates to be valid in order to execute, and results in the `watered` attribute being set to true for the desired plant.

Listing 3.2: Primitive control program for watering a plant

```
(define-control-program water (robot plant)
  (declare (primitive)
           (requires (and
                      (over :all (= (location robot) (location plant)))))
           (effect (at :end (= (watered plant) t)))
           (duration (simple :lower-bound 2 :upper-bound 5))))
```

Primitive control programs directly change or assess the environment in order to execute. Non-primitive control programs consist of multiple control programs (primitive or otherwise) which can be oriented sequentially or in parallel. The `move-and-water` control program simply composes the primitive programs `move` and `water` sequentially so that they can be executed as a single action.

Listing 3.3: Composite control program combining move and water

```
(define-control-program move-and-water (robot plant)
  (sequence (:slack t)
    (move robot (location plant))
    (water robot plant)))
```

Non-primitive control programs allow for the addition of a `:slack` parameter which defines the allowable amount of time between events that the planner can work with. The `t` value indicates that there is an arbitrary amount of time allowed between any events in the control program, giving the planner the most freedom to adjust timings to satisfy temporal constraints on duration.

Setting the situated knowledge for Kirk to operate over requires two additional constructs in RMPL. These are also the constructs which we will automate the generation of later in this thesis. Firstly, we need to set the initial state of the environment by initializing all relevant classes with their attributes. Environment initialization is accomplished via the `(define-initial-state)` form, where we can instantiate any relevant classes. Below is an example of what this form looks like.

Listing 3.4: Environment initialization for plant watering example

```
(define-initial-state start-state ()
  (:objects
    (plant-1 (make-instance 'plant :location 8070450532247991688))
    (plant-2 (make-instance 'plant :location 8070450532247997291))
    (plant-3 (make-instance 'plant :location 8070450532247954865))
    (robot-1 (make-instance 'robot :id 1 :location 5774839202384754346))
  )
)
```

The object oriented structure of RMPL allows for a natural specification of initial state in terms of the attributes of all relevant classes. If we wanted to expand an environment it would be as simple as initializing more classes of any type defined in RMPL.

Once the environment has been set, we specify the TPN for Kirk to schedule by creating a non-primitive control program called `main`. The `main` keyword is reserved by Kirk to be the only control program that it interprets as a TPN. Below is an example TPN declaration in RMPL for the plant watering scenario.

Listing 3.5: TPN declaration for a plant watering scenario

```
(define-control-program main (robot-1 plant-1 plant-2 plant-3)
  (with-temporal-constraint (simple-temporal :lower-bound 5 :upper-bound 100)
    (sequence (slack :t)
      (move-and-water robot-1 plant-1)
      (move-and-water robot-1 plant-2)
      (move-and-water robot-1 plant-3))))
```

As we can see above, the temporal constraint of the TPN is contained within the `(with-temporal-constrai` form. This allows us to specify bounds on the total duration of the mission, which Kirk then leverages while formulating an activity plan to satisfy the mission.

3.2 Plan Execution

Let's take a moment to step back and remember the core goal of this thesis. We would like to automate the process of translating environmental information into planning domains so that agents can plan and **execute** complex tasks which require a detailed understanding of the environment without major human effort. With execution being a key goal, we cannot purely rely on an activity planner. We must introduce capabilities to execute the plan, and automate the translation of environmental information required for execution along the way.

An executive is a module which is capable of generating an activity plan and dispatching appropriate commands to individual agents in order to achieve the mission goals. Dispatched commands could take the form of waypoints, velocity vectors, or even direct control inputs depending on the design of the system. We can split up an executive into two key components, the activity planner and the dispatcher. While Kirk is technically a full executive that can generate activity plans and dispatch actions to agents, it lacks some sophistication in how it dispatches commands. In particular, Kirk has trouble generating motion trajectories that are feasible for an agent's dynamics and doesn't have a proper understanding of environment structure in order to perform obstacle avoidance. To create a robust executive, we can leverage the activity planning of Kirk while replacing the dispatcher with a more effective system.

The Magellan executive produced by the MERS lab is capable of dispatching plans for agents which consider obstacles, agent dynamics, and convex constraints such as battery depletion and communication range [7], [8]. By leveraging a hybrid continuous and discrete time formulation for trajectory optimization, Magellan can generate fine grained motion trajectories which are influenced by the overall mission goals in order to maintain plan feasibility. A QSP of the mission, generated by an activity planner, provides the mission constraints that influence long term behavior. Magellan can monitor plan execution in real time to identify deviations and trigger replans based on new observations. By combining the dispatcher and state monitor from Magellan with the activity planning from Kirk, we are able to construct a plan executive with robust capabilities that can handle complex missions. Since Kirk outputs plans in the form of QSPs, we can easily integrate the two modules so that activity planning is handled by Kirk and execution is performed by Magellan. Magellan takes an initial model consisting of an environment and internal dynamics definition along with optional hyper parameters, then generates feasible motion trajectories for agents to achieve the mission goals.

The environment definition for Magellan contains all relevant information regarding the external world, such as the locations and geometry of obstacles and the locations of interest regions which denote the locations of task goals. Obstacles can be grouped so that they only apply to certain agents, which allows the specification of complex environments where certain agents will never interact with some obstacles. A simple example of this is in an underwater exploration scenario, where an AUV must avoid deep water obstacles but a communication vessel on the surface can move directly over them.

A careful reader may infer that this thesis will focus on automating the generation of this environmental definition. That reader would be correct. Automating the environmental definition brings a few key benefits which we would like to leverage in the pursuit of true

autonomy. Firstly, real world environments can be very complex, with many obstacles and almost infinite variations of interest regions depending on which tasks we assign our agent. If we were to manually generate the environment model for each environment and task we want to execute, the definition step alone would likely outweigh any benefits from using an autonomous agent. Secondly, all of the information which we would like to use in the environment definition comes directly from observations of the world. This means that we have the opportunity to use perception systems to perform the work that a human would typically be doing to identify obstacles and objects or locations of interest.

The generation of vehicle dynamics is slightly different. Either the dynamics of a vehicle would be known prior to deployment via expert knowledge, or they would be learned during the course of execution. In both cases, semantic environmental information is not particularly useful to generate a dynamics model, so we will avoid automatically generating it in this work. That being said, see [17] for an interesting approach to performing long horizon execution with learned or uncertain dynamics.

With the executive we've assembled above, we can now specify the problem that this thesis seeks to solve in service of semantically grounded planning and execution.

Definition 3. *The **automatic environmental translation** problem is given by $\langle \mathcal{P}, \mathcal{O}, \mathcal{T} \rangle$ where $\mathcal{P}$ is the set of prior knowledge in our representation language, $\mathcal{O}$ is a set of observations of the environment, and $\mathcal{T}$ is the task which we would like our agent to execute.*

And a solution to this problem is as follows.

Definition 4. *A **solution to the automatic environmental translation problem** is given by $\langle \mathcal{D}_p, \mathcal{D}_e \rangle$ where $\mathcal{D}_p$ is a complete domain for the activity planner, specifically the initial state of any relevant objects and the plan outline to complete the task. $\mathcal{D}_e$ is the complete domain for the execution module, with a full description of obstacle geometry and locations which we would like to travel to in service of the task.*

3.3 Semantic Mapping

As we would like to automate the definition of the planning domain for Kirk and environment model for Magellan, we first need a method to convert observations of the environment into some kind of data structure. The data structure will need a few key pieces of information. In order to generate the domain model for Kirk, we need access to every object in the environment and their attributes. Objects can then be filtered for specific tasks, and passed to Kirk via the RMPL representation language. To generate the environment model for Magellan, we need access to detailed map geometry so that we can define obstacles for

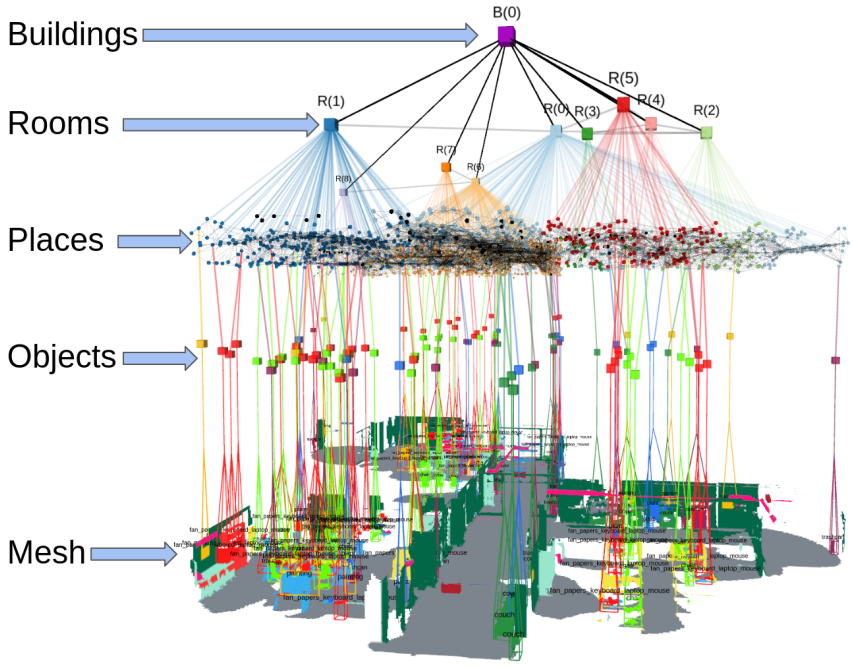

Figure 3.4: Dynamic Scene Graph of Office Environment

navigation. We also need the list of objects which we've filtered based on the specific task so that we can generate interest regions surrounding each object we would like to travel to.

To go from raw observations to a detailed map with semantic labels, we need a semantic perception engine. There have been major advances in semantic SLAM and perception in recent years, allowing for the possibility of truly autonomous systems which are able to understand the semantic structure of their environments. The semantic perception system presented in [9] (Hydra) is one of the most expressive semantic perception engines to date, and offers several key capabilities which lend themselves to advanced planning systems. The primary capability is to label objects in the environment and situate them in the physical map. This is the most basic required feature for semantically grounded planning, as we must be able to locate objects of interest in the real world. One of the hallmarks of Hydra is the hierarchical structure of the Dynamic Scene Graph (DSG) which it generates.

As shown in Figure 3.4, there are five layers to the Hydra output: Mesh, Objects, Places, Rooms, and Buildings. Each of these layers contains a graph of the corresponding type of entity, and interlayer edges denote containment by higher layers. If a place ϕ in layer three is connected to a room ρ in layer four, we know that ϕ is contained within ρ. By connecting objects to the rooms and buildings which they are contained, we easily answer questions such as "Where are the plants in the hallways?" or "Does the conference room contain a painting?" which may otherwise be difficult to answer.

Hydra is especially useful for environment translation because it provides easy access to everything we need for our main translation task. Objects can be accessed easily via the Objects layer in the graph, and map geometry is stored in the mesh layer for generating the set of obstacles.

Chapter 4

Environment Translation

With the tools discussed in chapter 3, we're ready to build a bridge between semantic perception and long horizon planning and execution. A bridge must be able to effectively convert the information from the perception system so that the planner can quickly generate activity schedules for various missions and the plan executive can dispatch the plan to an agent. As shown in Figure 4.1, a data processing block intakes the query, DSG, and initial domains for the activity planner and executive. It then transfers the necessary information to each domain specification block so that they can fully specify the domains for the planner and executive. Section 4.1 describes the data processing block of Figure 4.1, while sections 4.2 and 4.3 describe how each domain specification module operate.

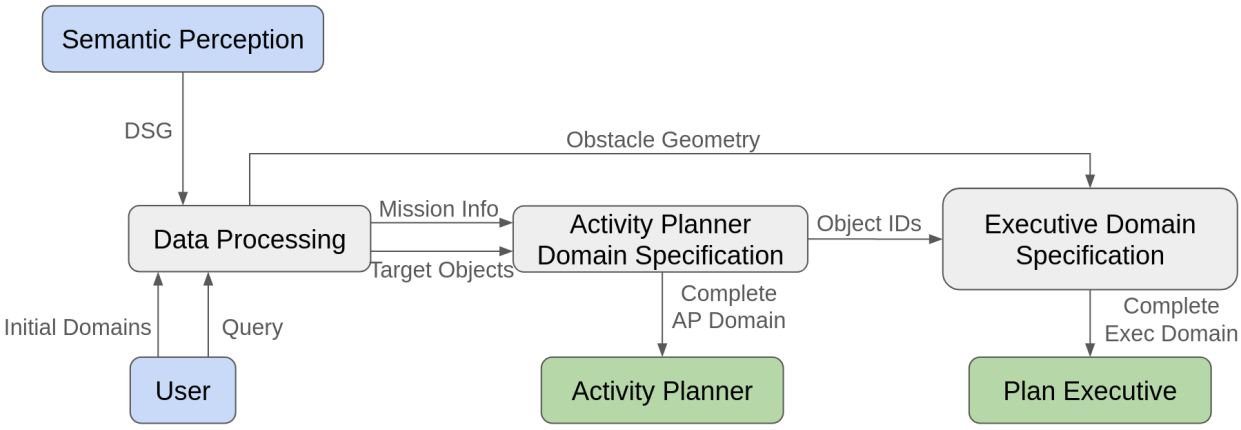

Figure 4.1: Data Flow Block Diagram of Domain Specification Modules

4.1 Command Verification

The system starts by processing the mission command to extract the important objects and actions. Commands are passed as a list with a regular structure. The first element is the name of the agent which we would like to direct for our mission. The second element is the action we would like the agent to execute on the target object or objects. The third element is the name of the target object class, which may correspond to more than one object in the environment. The fourth argument is the room type which contains the target object(s), such as a hallway or bathroom. The fifth argument is a lower and upper bound for the total duration of the mission, in seconds. One task query from the office scenario is the list `["robot", "move-and-water", "plant", "hallway", (0,3600)]`. This command, as you may guess, directs the robot agent to move to and water all plants in the hallways in between 0 and 360 seconds.

It is vital to verify that the provided query makes logical sense in the scope of the mission and the environment. The requested action must be something the desired agent can do, and it should be able to perform it on the desired target object. If we were to request an agent to water all trashcans in a building, the verification system should report that this is infeasible because a trashcan cannot be watered. To verify a query, the system parses the RMPL file of predicates and actions to understand which objects exist and which actions apply to each object. As long as the requested objects have a class definition in RMPL and are able to engage in the requested action, the verification will pass. Verification does not take into account the possibility of objects not existing in the actual environment, as this is handled by the activity planner domain specification module. In algorithm 1 we can inspect the query verification algorithm in detail.

4.2 Activity Planner Domain Specification

If the provided command is valid for an agent α, target object class ω, and room type $\Re$ the Activity Planner Domain Specification (APDS) module searches the DSG for objects of the ω class within any room of type $\Re$. The objects found this way are relevant to the mission. The module then generates an initial state for the planner by initializing all instances of ω type objects with their specific traits. In the current implementation the only non-default attribute for an initialized object is its location, but see section 6.2 for a discussion on ways to expand the attributes for initialized classes during domain specification. The system then writes a main control function in RMPL which stitches all desired actions together for each relevant object (objects of the correct type that are in the correct rooms), and saves it to a

Algorithm 1 Validate Command in RMPL Model

Input: `model_path` (path to RMPL model), `command` (list of strings representing the agent's command)

Output: `True` if the command is valid, otherwise raises an error

```
actions ← {}                                   ▷ Initialize empty dictionary for actions
objects ← {}                                   ▷ Initialize empty set for objects
Open file at model_path as file:
for each line in file do
    action_match ← regex search for action definition
    relevant_obs ← regex search for objects involved with the action
    class_match ← regex search for class definition
    if action_match is found then
        action ← action name
        obs ← objects from relevant objects
        actions[action] ← obs
    else if class_match is found then
        ob_class ← class name
        objects.add(ob_class)
    end if
end for
agent, task, ob, _, _ ← command
if task is not in actions.keys() then
    Raise ValueError("Desired action is not defined in RMPL.")
end if
if agent is not in objects then
    Raise ValueError("Requested agent type does not exist.")
end if
if ob is not in objects then
    Raise ValueError("Requested object type does not exist.")
end if
if agent is not in actions[task] then
    Raise ValueError("Agent cannot execute the desired action.")
end if
if ob is not in actions[task] and "move" is not in task then
    Raise ValueError("Desired action is not applicable to provided object.")
end if
Return True
```

file to be passed to Kirk. After writing the main control function to RMPL, the module also sends the location ID's of each relevant object to the Plan Executive Domain Specification Module (see section 4.3). The returned ID's correspond to nodes within the places layer of the DSG (Layer 3 in figure 3.4), which are known to be in free space. In algorithm 2, we may observe the flow from verification to domain population in more detail.

Algorithm 2 Activity Planner Domain Specification

```
Input: agent_type, action, object_type, room_type, time_bound, DSG
Output: initial_state, main_control_program
Objects ← []
for each room in room_layer do
    if classify_room(room) == room_type then
        for each object in room do
            if object.class == object_type then
                Objects.add(object)
            end if
        end for
    end if
end for
Object_locations ← {}
Action_defs ← []
for each object in Objects do
    if edge between object and place then
        Object_locations[object] ← place.id
        Action_defs.add("(action agent_name object)")        ▷ RMPL formatted string
    end if
end for
Object_defs ← []
for each object_name, locationinObject_locations do
    Object_defs.add("(object_name (make-instance 'object_type :location location))")
end for
Agent_def ← ["(agent_name (make-instance 'agent_type :location agent_location))"]
Initial_state ← "(define-initial-state start-state () (:objects object_defs + agent_def))"
Main_program ← "(define-control-program main (Objects & agent) (Action_defs))"
WriteToFile(Initial_state, Main_program)
Return Object_locations
```

This system can generate many RMPL models for a given environment, automatically locate and reason over desired instances of defined classes, and generate a plan via Kirk without manual specification of object locations and action ordering. In many cases the manual effort required to specify various missions in a set environment can be extensive, and also introduces the possibility of user error when operating over many objects and agents.

By automating the domain specification process, we can drastically increase the accessibility and reliability of our planning systems.

4.2.1 A Note on Room Classification

In order to satisfy hierarchical semantic queries such as "Water the plants in the hallways", we need to have a classification method to determine what type of rooms we have in a given environment. While there are a variety of classification methods to determine a given room type, including a recently published method that works directly on 3D Scene Graphs [18], novel room classification methods are not a focus of this thesis. For the purposes of this thesis, room classification is executed via a simple decision tree that determines a room's type based on the objects contained therein. The room classification module of this work can be swapped for any other method which fits the DSG data type without compromising any of the other components. Any reader interested in a more advanced approach to room classification should see [18] for an in-depth discussion of the topic. Also see 6.2 for a discussion of how advances in LLMs may allow for a more diverse room description language.

4.3 Plan Executive Domain Specification

Specification of the Magellan environment requires translation of all relevant obstacles from the DSG, as well as a set of interest regions which correspond to the locations of target objects. In the office scenario, where the task is to water all plants in the hallways, relevant obstacles are any obstacles positioned such that the agent could possibly collide with them. Interest regions are convex regions surrounding the location of each plant. The plan executive domain specification problem thus consists of determining which obstacles are relevant to our agent, and where the interest regions are for each target object. Once these have been identified, we can write them to the model file for Magellan to begin trajectory planning and execution.

To maintain a high degree of granularity in obstacle geometries, we generate the relevant set of obstacles from the mesh layer of the DSG (layer 0 in figure 3.4). This layer contains every triangular mesh face created by the mapping system, where each face is a set of three vertices in 3D. While this choice of obstacle formulation yields the high granularity necessary for reasonable navigation without human intervention, it also poses a problem to the optimization formulation which Magellan originally relied on. With tens of thousands of very small obstacles, floating point errors and optimization time become prohibitive to normal operation. See section 5 for a discussion of how Magellan was updated to work with

Algorithm 3 Obstacle Filtering

Input: mesh_layer Ω, agent_height $\mathcal{H}$
Output: Filtered mesh layer of dsg projected to 2D

```
Z_offset = 0.05
M_2D = empty set
for triangle ∈ Ω do
    for vertex ∈ triangle do
        if Z_offset ≤ v_z ≤ H then AddFace2D(M_2D, triangle)
        end if
    end for
end for
return M_2D
```

complex environments.

It is beneficial to define the environment specification problem concretely. We denote the set of all mesh triangles in the environment as Ω, the height of the agent as $\mathcal{H}$, and the set of object location ID's from the APDS module as Λ. For this module, we assume a ground based agent that doesn't change in height. Therefore, we may filter Ω by only looking at the z component of the mesh vertices. The filtering process is then a simple comparison, where any face with a vertex with z component in the range $[0.05, \mathcal{H}]$ is kept as a relevant obstacle. The 0.05 lower bound is designed to filter out the ground, and assumes that the ground is flat and the agent is able to move over small obstacles under 5 centimeters in height. At this point, Ω will only contain mesh faces that have the potential to intersect with the agent. Magellan generates trajectories over 2D environments, therefore we project all faces onto the $z = 0$ plane. Algorithm 3 depicts the filtering process, using the `AddFace2D` subprocess to project the face to 2D and add it to the new map.

In figure 4.2, we can see the 2D projection of the office mesh with no filtering. Navigation would be impossible for Magellan in this environment. Figure 4.3 demonstrates the drastic reduction in mesh triangles after filtering for relevance. While the filtered environment still has drawbacks regarding optimization time and numerical errors, navigation is theoretically possible for Magellan.

Interest regions are convex regions that serve as potential goals for Magellan's trajectory planner. Not every interest region needs to be traveled to, but any goal location must be encoded as an interest region. Λ contains all of the attributes for each target object, including the bounding box from the segmentation model of the mapping system. By enlarging the bounding box in each direction by some dilation constant δ and projecting it to 2D, we can easily construct an interest region for each object. Scaling the bounding box is necessary

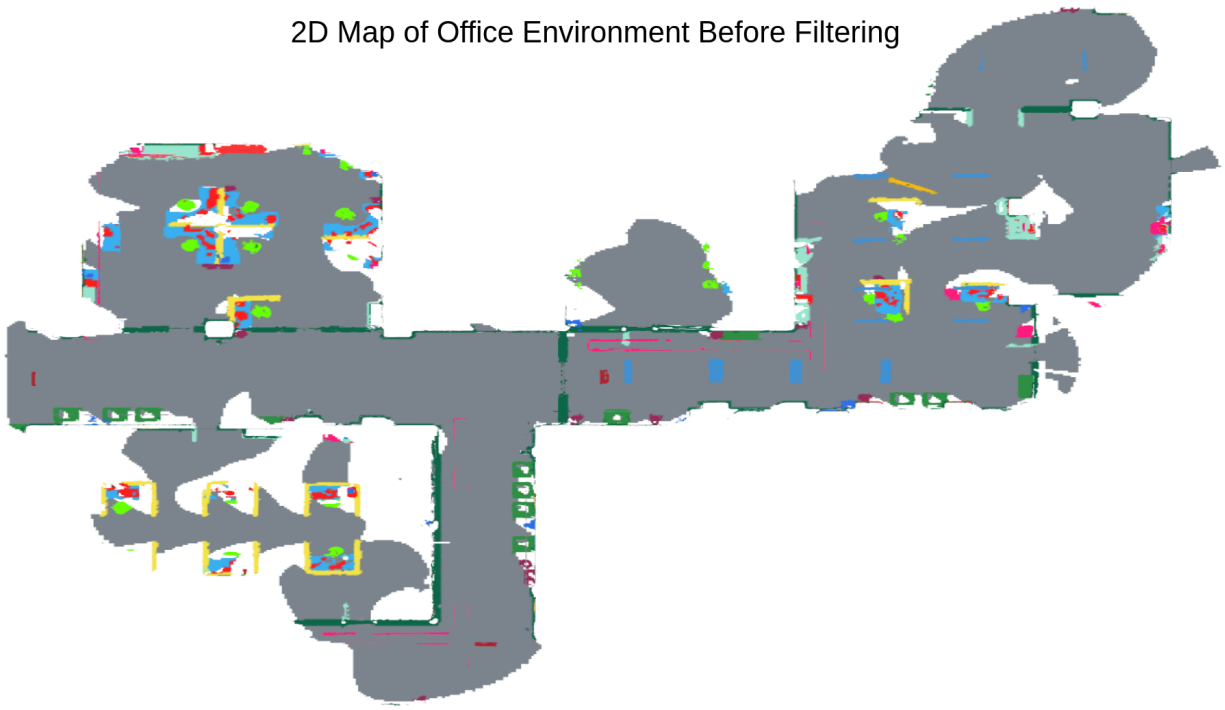

Figure 4.2: 2D map of the office environment before any mesh triangles have been filtered out. This map contains over 600,000 individual triangles.

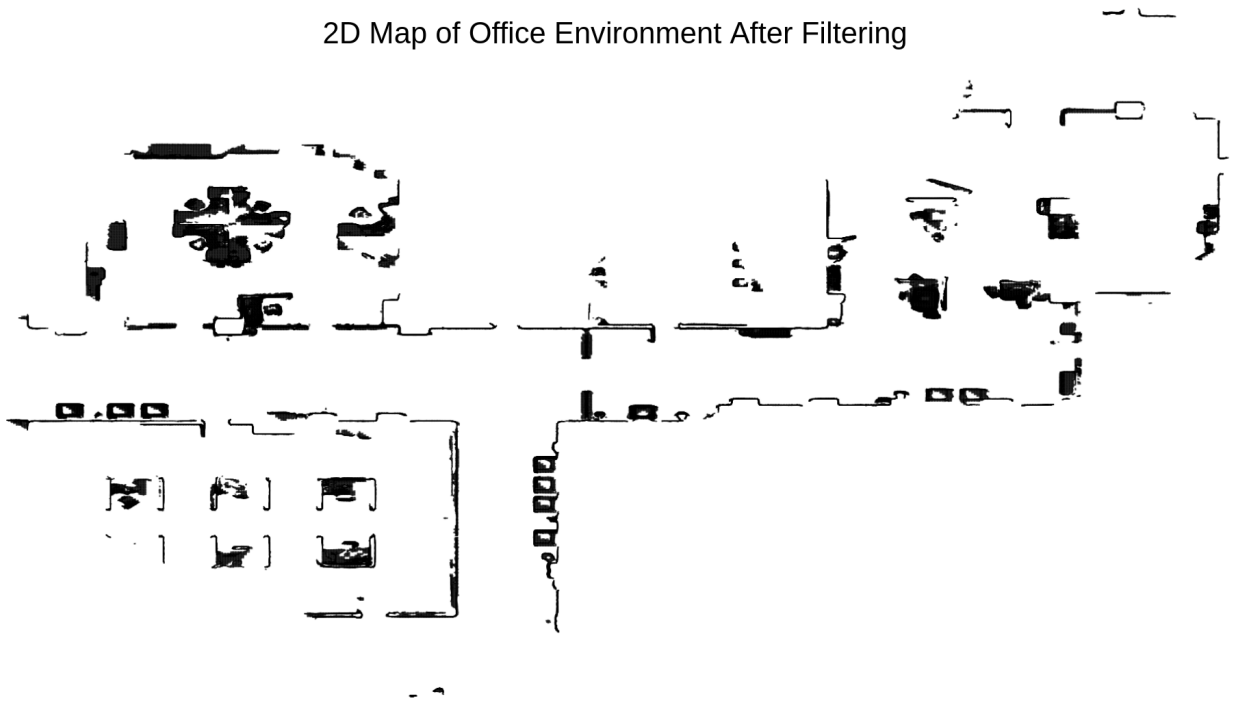

Figure 4.3: 2D map of the office environment after filtering. This map contains roughly 28,000 triangles.

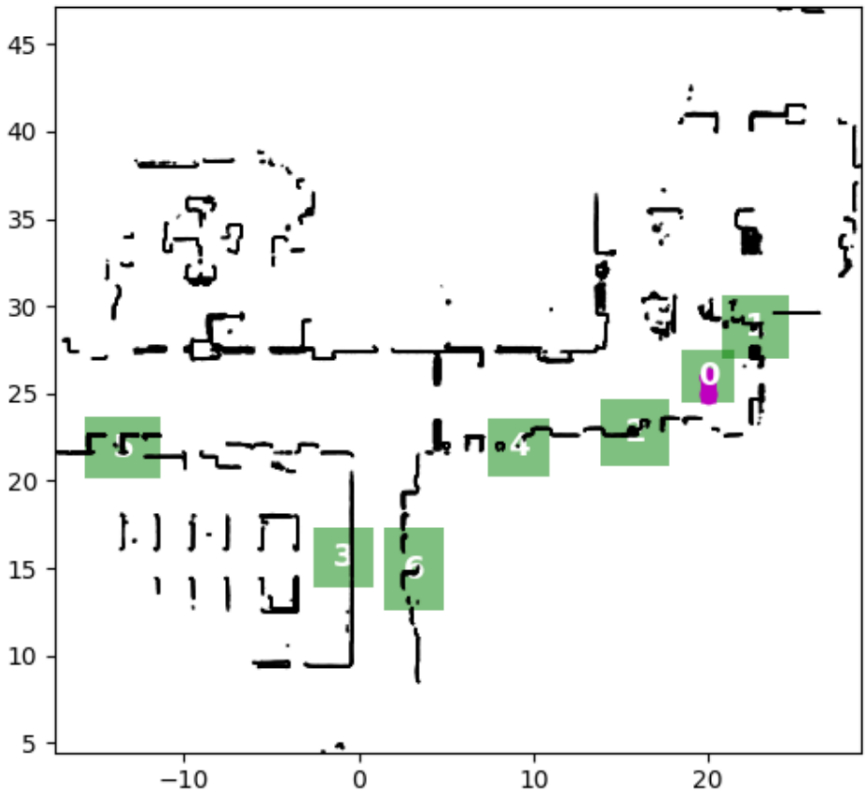

Figure 4.4: Visualization of complete Magellan model. Obstacles are in black while interest regions are in green.

because every object in the environment is also part of the mesh, and is therefore an obstacle. By dilating the bounding box, we denote a range from the object through which we can still interact with it. After relevant objects and interest regions have been constructed and written to the model file for Magellan, the environment is ready for planning. Figure 4.4 shows the model for the office environment, where the task is to water all plants in the hallways.

Chapter 5

Scaling to Large Maps

Over the course of this project, it became clear that Magellan would have trouble handling large scale maps with many obstacles. The canonical examples that Magellan was used on had less than 10 obstacles each, and each obstacle was large and well defined. The mixed integer program approach that Magellan leverages to perform trajectory optimization is designed to be highly reactive to a dynamic environment, while sacrificing optimization power on highly detailed, static maps. As noted in [7], Magellan can also be outfitted to handle large scale static environments by switching from the mixed integer program formulation to a combination of search and convex trajectory optimization through safe regions. The method for scaling to complex, static environments is similar to that proposed in [19]. Firstly, construct a graph of convex safe regions, where vertices in the graph are intersections between the safe regions. Trajectory optimization is then a series of convex optimizations within safe regions, where the endpoint of the trajectory must be inside of the intersection region between the current safe region and the next safe region. Trajectory segments for each convex region can then be stitched together to efficiently generate a trajectory through a potentially complex environment.

While trajectory planning through a graph of convex sets is efficient, generating the convex regions can be cumbersome for a complex environment with many obstacles. For this reason, the convex safe region approach is most applicable when operating in a complex, static environment. In these cases, the convex safe regions can be precomputed and the planner can rely on the more efficient convex optimization to generate trajectories through those safe regions. In many offices, the general layout is constant with some minor changes to the positions of personal belongings. Therefore, the convex optimization approach will allow efficient planning while preserving plan accuracy in the office scenario.

To generate convex safe regions for a given environment, we can rely on the Iterative Region Inflation by Semidefinite Programming (IRIS) Algorithm [20]. This algorithm utilizes

an optimization procedure to generate convex regions of maximal area which lie entirely in free space within an environment containing an arbitrary number of obstacles. The output of IRIS is a set of regions which we can utilize for motion planning within a complex environment.

Updating Magellan to work with the convex safe region formulation is a relatively simple task. The key step is during the MPC optimization problem formulation, where obstacle avoidance constraints are declared for each time step. Previously, Magellan would enforce obstacle avoidance by declaring that the position of all agents needed to be outside of the polygons which define each reachable obstacle. Constructing the reachable set of obstacles as well as defining the exclusion constraints are presented in [7], [8]. The key insight is that by using half space representations for obstacles, a set of linear inequality constraints can be added to enforce that the position of all agents are outside of all relevant obstacles.

To use a convex safe region approach, we reverse this constraint to declare that all agents need to be inside of at least one polygon defining a safe region at each time step. As safe regions are convex polygons, we can construct a set of linear inequality constraints $\{x|\boldsymbol{A}x \leq \boldsymbol{B}\}$ where the matrix $\boldsymbol{A}$ and vector $\boldsymbol{B}$ have rows for each edge of the polygon and x is any point in the 2D space of the environment. If the inequality holds, x is contained within the polygon defined by $\boldsymbol{A}$ and $\boldsymbol{B}$. In the discrete time formulation of the trajectory planner, agent k's trajectory at time step i can be contained within safe region R by adding the constraint

$$\begin{aligned} \boldsymbol{a}_s \boldsymbol{x}_{k,i} + M \cdot (1 - p_{n,i}) &\leq \boldsymbol{b}_s \quad n = 1, \ldots, N \\ \sum_{n=1}^{N} p_{n,i} &= 1 \end{aligned} \tag{5.1}$$

where $\boldsymbol{a}_s$ is row s in matrix $\boldsymbol{A}$, $\boldsymbol{b}_s$ is row s in vector $\boldsymbol{B}$, $p_{n,i} \in \{0, 1\}$ are binary decision variables for each obstacle and each time step, M is a large constant, and N is the number of convex safe regions in the environment [8], [21]. We are essentially adding a containment constraint for all edges of a safe region n if the decision variable p_n is triggered, then enforcing that only one safe region is able to claim "ownership" of the agent at any time step i. While the agent can theoretically be in multiple safe regions at the same time, restricting containment to a single safe region at each time step is important for the next step of the constraint definition process.

Corner cutting is an edge case where an agent traverses from one safe region at time step i to another region at time step $i + 1$ without moving through the intersection region

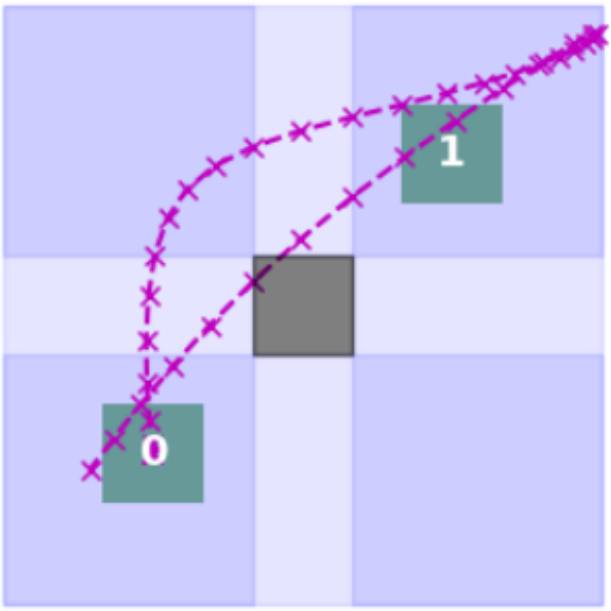

Figure 5.1: Simple example of corner cutting. While each time step (denoted with an 'x') is inside of the safe regions (blue), the actual trajectory crosses through the obstacle (gray).

between them. As shown in figure 5.1, even if the position of the agent at each individual time step is inside of a safe region, that doesn't imply that the entire trajectory is safe. To prevent corner cutting, we can leverage the ownership constraint defined in 5.1 and define a few key terms. $\boldsymbol{A}_{j,h}$ and $\boldsymbol{B}_{j,h}$ define the convex intersection between regions j and h. $c_{j,h,i} \in \{0, 1\}$ is a binary decision variable associated with transitions between regions. $p_{j,i}$ is the "ownership" binary variable defined in 5.1. We can then add the following constraints to our model

$$\begin{aligned} p_{j,i} + p_{h,i+1} = 2 \implies c_{j,h,i} &= 1 \\ \dot{\boldsymbol{a}}_s \boldsymbol{x}_{k,i} + M \cdot (1 - c_{j,h,i}) \leq \dot{\boldsymbol{b}}_{\boldsymbol{s}} \quad j, h &= 1, \ldots, N \end{aligned} \tag{5.2}$$

where $\dot{\boldsymbol{a}}_s$ is row s in the matrix $\boldsymbol{A}_{j,h}$, $\dot{\boldsymbol{b}}_{\boldsymbol{s}}$ is row s in vector $\boldsymbol{B}_{j,h}$, M is a large constant, and N is the number of convex safe regions in the environment. Intuitively, we track the regions which claim each time step, then construct a decision variable which denotes transitions between two regions at a certain time step. When the transition indicator is active, the agent must be inside the intersection between the two regions. This ensures that transitions between regions always occurs in a safe region, and all possible trajectories will exist entirely inside of the safe regions. This approach is similar to the solution to corner cutting presented in [22], but is designed for the safe region formulation instead of the obstacle avoidance

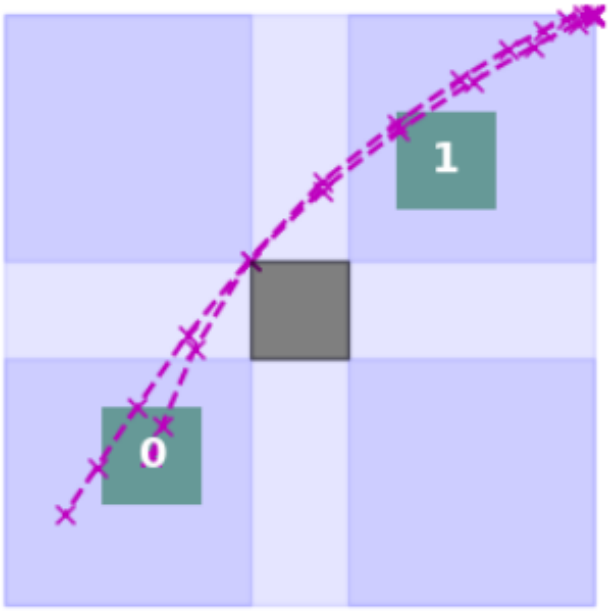

Figure 5.2: Simple example which avoids corner cutting. The agent is forced to transition between safe regions (light blue) through the intersections (darker blue). This avoids any cases where the agent might accidentally pass through an obstacle.

formulation. In figure 5.2, we can see a simple trajectory using the additional constraints. When transitioning between two safe regions, the entire trajectory always stays within the safe regions.

By incorporating these changes to how Magellan formulates obstacle avoidance during MPC, we are able to efficiently generate trajectories to satisfy multi-goal plans in highly complex environments. In the motivating scenarios of this work, this method is much preferred over the more reactive optimization protocol presented in [7], [8], as we assume static environments with large numbers of very small obstacles.

Chapter 6

Experiments and Results

The key motivation of this thesis is to enable long horizon, constraint based planning in service of semantically grounded goals by providing an automated bridge between semantic information created by a perception module and constraint based planners which can generate robust activity plans and trajectories. By automatically translating information generated by Hydra into forms which Kirk and Magellan can plan over, the integrated system is able to complete semantically grounded queries without the user needing to carefully understand and translate the environment. With a simple command, a user can generate complex mission descriptions that would take significantly longer to generate manually and may contain errors due to the complexity of the map.

6.1 Demonstration

To demonstrate the effectiveness of this system, we can analyze the canonical example within the office custodian application presented in chapter 1. There are four inputs which we will need to provide to the system in order to plan and execute a task. Three of the inputs only need to be generated once per application environment, and will be described in section 6.1.1. The final input is the mission command, which is generated once per mission.

6.1.1 Office Environment Specification

As described in section 4.2, we must have an initial RMPL model for Kirk which contains "prior knowledge" about the environment and agent capabilities. The initial RMPL model for the office environment contains definitions of all relevant object and agent classes, as well as all actions that our agent is able to execute.

It is important to note that this model would need to be specified for any new environment

or robot. Information regarding the types of agents that can be controlled, types of objects which may exist, and actions that can be taken between certain agents or objects cannot be inferred by this system and must be provided externally. In this limitation, there is potential for future work to develop automatic methods to derive this information directly from the environment, which will be discussed in section 6.2.

As described in section 4.3, we also need an initial model for Magellan which contains the dynamics of any controllable agent in the environment along with some optimization parameters. There is only one agent in the office environment, so the initial model for Magellan is as follows.

```
vehicles:
  robot1:
    accel_bounds:
    - - -2
      - 2
    - - -2
      - 2
    bounding_box: 0.45
    name: robot1
    rmpl_vars: "robot1.location"
    velocity_bounds:
    - - -2
      - 2
    - - -2
      - 2
```

As with the initial RMPL model for Kirk, the initial model for Magellan only needs to be generated once for a given deployment environment, and many missions can be assigned based on these initial models.

6.1.2 Watering Plants

The final input from the user is the mission specification which we would like to execute. In this example, we would like the agent (of type "robot") to water all plants in the hallways of the building within one hour. The mission description is as follows.

```
["robot", "move-and-water", "plant", "hallway", (0,3600)]
```

This input will need to be changed for each new task we would like to assign to our agent.

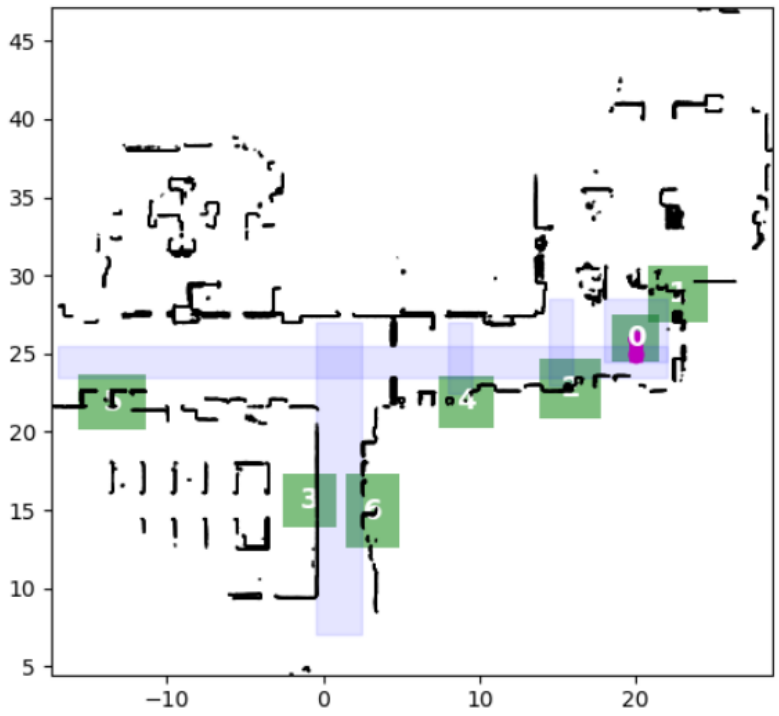

Figure 6.1: Visualization of complete Magellan model for the plant watering task. Obstacles are black, safe regions are blue, and locations of various plants are green. The agent's start location is in pink.

As the input is very simple to generate, it is easy to command our agent to execute many different tasks in a given environment.

The mission command, along with the three environment specifications, provide all necessary context for the integrated system to fully specify, plan, and execute the mission. The resulting complete environment model for the mission is shown in figure 6.1. Note that the safe regions (blue) are manually created in this example for simplicity of visualization. Generally, the IRIS algorithm mentioned in section 5 will generate safe regions for planning but is considerably worse for visualization. For a fully IRIS generated set of safe regions in the office environment, see figure A.2 in the appendix.

We can visualize various legs of the mission in figure 6.2. Note that the visualizations are simplified for clarity, and aren't the exact same as the trajectory for watering all six plants. The full trajectory is visualized in figure A.1 in the appendix, but is considerably more dense. From figure 6.2, we can see that the agent always stays within the blue safe regions to ensure safe operation. Additionally, the existence of the trajectory implies that the mission is able to be completed while respecting all timing constraints. Magellan is then able to dispatch timed control commands to the agent in order to execute the mission.

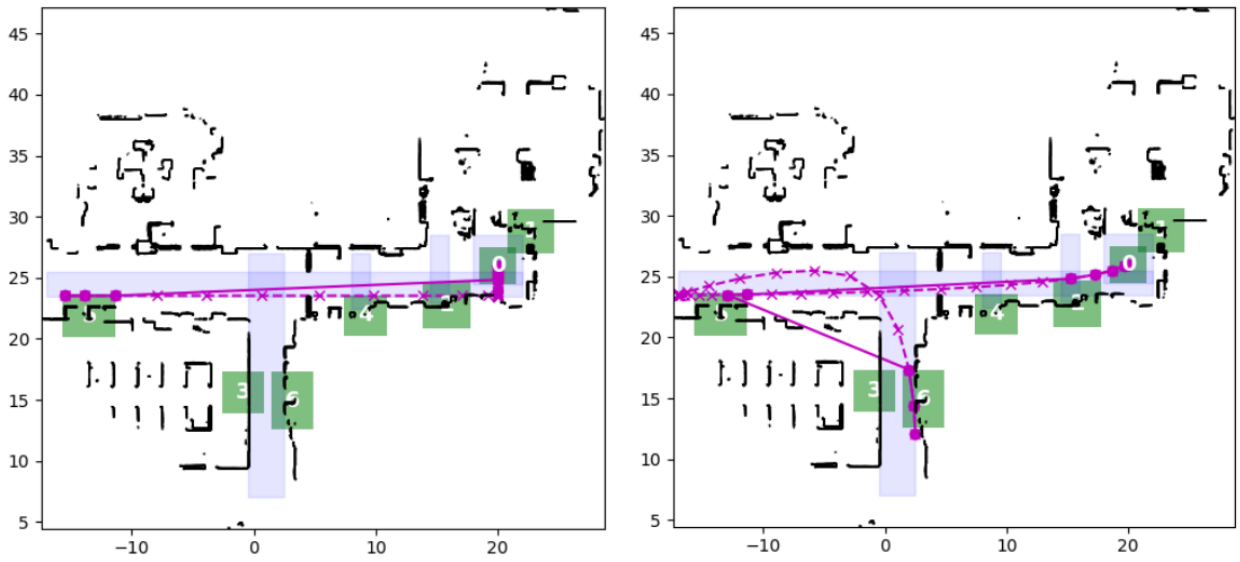

Figure 6.2: Visualization of plant watering trajectories, simplified for clarity. Left is the trajectory when only watering one plant, right is the trajectory when watering two plants. Solid lines connect locations for each event in the plan, while dashed lines track the MPC trajectory with vehicle dynamics.

6.2 Future Work

This thesis was an exploration into what could be possible with a synthesis of advanced semantic perception and constraint based planners. Throughout the exploration process, many features, limitations, and opportunities for future efforts were discovered. There are many ways in which future researchers can further increase the autonomy of this system, from automating the few remaining manual tasks to increasing the range of representable problems for these technologies. By using this research as a starting point, it may be possible to create truly autonomous, goal conditioned agents with robust planning and execution capabilities.

6.2.1 Initial Domain Specification

As noted in section 6.1.1, the initial domains for Kirk and Magellan are manually generated. These initial domains contain information that is specific to the environment and agents which we are deploying, but agnostic to the specific task which we would like to execute. In this respect, we can understand the initial domains as the "prior knowledge" that any operator would know purely based on the environment and available agents. For example, an operator should know which objects a certain robot will be able to pick up, move, or survey with available sensors. Many of these features fall into the general category of "common

sense" knowledge, and therefore might be able to be automated via Large Language Models, which perform well at tasks requiring common sense knowledge. In applications from this thesis where the environment has been pre-mapped, an LLM may be able to generate feasible actions for each object in the environment based on the types of agents present. This would significantly decrease the amount of manual effort required to perform initial domain specification, and open the door to higher degrees of autonomy.

6.2.2 Object Representations in Kirk

Over the course of this thesis, Kirk proved highly effective in planning over complex problems. Unfortunately, Kirk was significantly limited when it came to expressing relational concepts between objects in the environment. Transitive relations are impossible to express or infer in the current implementation. For example, if we know that object $\mathcal{A}$ is above object $\mathcal{B}$, and that object $\mathcal{B}$ is above object $\mathcal{C}$, we cannot directly infer that object $\mathcal{A}$ is above $\mathcal{C}$. Comparisons between class attributes are also quite limited. Consider the following class definitions in RMPL.

```
(defclass class1 ()
   ((location
    :initarg :location
    :type integer
    :accessor location)))
(defclass class2 ()
   ((location
    :initarg :location
    :type integer
    :accessor location
    :documentation)))
```

As we can see, both `class1` and `class2` have the integer typed location attribute. Despite the fact that they contain the same type of object, the equality check `(= (location class1) (location c` is not implemented. Rather, comparisons are only possible between two direct values, or a single class attribute and a direct value. By increasing the expressiveness of Kirk for object oriented functionality, we may increase the set of semantically grounded problems which we can express and plan over.

6.2.3 Room Classification

In this thesis, room classification is performed via a decision tree based on the objects contained inside the room. While this works well for simple environments, there are limitations when it comes to generalizing to new environments or new types of objects. To further increase the level of autonomy in this type of system, it may be useful to leverage advances in LLM reasoning to perform room classification. One such technique, used specifically on 3D Scene Graphs, is able to classify regions of space based on the geometry and objects contained within, and can generalize well to various indoor and outdoor environments [18]. An application of this technique or similar techniques to classify rooms may allow this system to generalize to a wide range of environments quickly, further reducing the amount of manual effort required to deploy into new settings.

6.2.4 Planning From Goals via PDDL Planners

Currently, Kirk takes a sketch of events with possible choices and generates an assignment of events which satisfies all constraints. We cannot simply pass a goal condition which we would like to satisfy at the end of the mission. This limitation means that more complex semantically grounded missions are infeasible to specify. For example, we cannot specify a mission to **make me a cup of coffee in fifteen minutes**. The mission cannot be specified because making a cup of coffee requires multiple different actions. The agent needs to first grab a cup, then place the cup under the coffee maker, then run the coffee maker. If we would like to build up generalized functionality in our autonomous agents, we need to be able to express complex actions as the composition of primitive actions which are easier to define.

In the planning community, PDDL planners are designed to find a series of actions which take the agent from the start state to the goal state, enabling complex missions where it would be too cumbersome for a user to sketch the mission manually. By integrating a PDDL planner with Kirk's constraint based scheduling algorithms, we may be able to reap the rewards of simple goal conditioned planning with robust constraints.

6.3 Conclusion

In the future, autonomous agents will need to be able to plan and execute semantically grounded missions while obeying diverse constraints. Autonomous agents will therefore need to understand the meaning of objects in their environment, and utilize their understanding in service of semantic goals. To satisfy the need for constraint based planning with

semantic environmental understanding, it's clear that we need a tight integration of semantic perception and planning systems.

In this thesis, I presented an architecture which is able to solve semantically grounded problems with diverse constraints. The key elements of this approach are a semantic mapping system, constraint based activity planner, constraint based plan executive, and automated system for translating information from the semantic map to the planner and executive. While largely a synthesis of existing technology, this approach required changes to the plan executive's trajectory optimization formulation in order to optimize trajectories over large, complex maps. Below is a summary of the contributions of this thesis.

6.3.1 Semantic Mapping

An autonomous, semantically driven agent must extract semantic information directly from the environment without user input. The Hydra perception engine [9] provides a rich set of semantic and structural data directly from sensor readouts which we can then use for semantically grounded planning. Hierarchical relations in the Hydra output allow for a rich expression of object and location relationships, enabling a wide range of mission and goal specifications.

6.3.2 Activity Planning

To enable agents to respect mission timing constraints, the Kirk activity planner and scheduler [6] is leveraged to generate feasible activity plans. Kirk takes in a loose sketch of activities in a mission, and generates a feasible assignment of activities in order to complete the plan while obeying all necessary constraints. The resulting plan can be dispatched and executed by an agent to complete a mission. The RMPL modeling language allows for the expression of object and agent classes, allowing Kirk to reason over plans including multiple agents and objects.

6.3.3 Plan Execution

Alone, activity plans generated by Kirk aren't sufficient for robust execution of plans. In order to generate robust plans for execution, the Magellan plan executive provides a trajectory optimization formulation that obeys all activity plan constraints along with agent dynamics and environment obstacles [7]. An activity plan from Kirk, along with an environment model describing obstacles, regions of interest and vehicle dynamics are passed to Magellan in order to generate feasible motion trajectories for our agent to complete a mission.

6.3.4 Automatic Semantic Translation

The key component in this work is the creation of an automatic bridge between semantic perception and constraint based planning and execution. The Activity Planner Domain Specification and Executive Domain Specification modules perform translation between the DSG generated by Hydra and the domains that Kirk and Magellan expect to plan over. This translation allows for the expression of complex, semantically grounded missions without the user manually specifying all relevant objects, regions of interest, and obstacles. Manual specification of these fields would take considerable amounts of effort for each mission and new application, and may introduce human error that further limits the reliability of an autonomous system.

Appendix A

Appendix

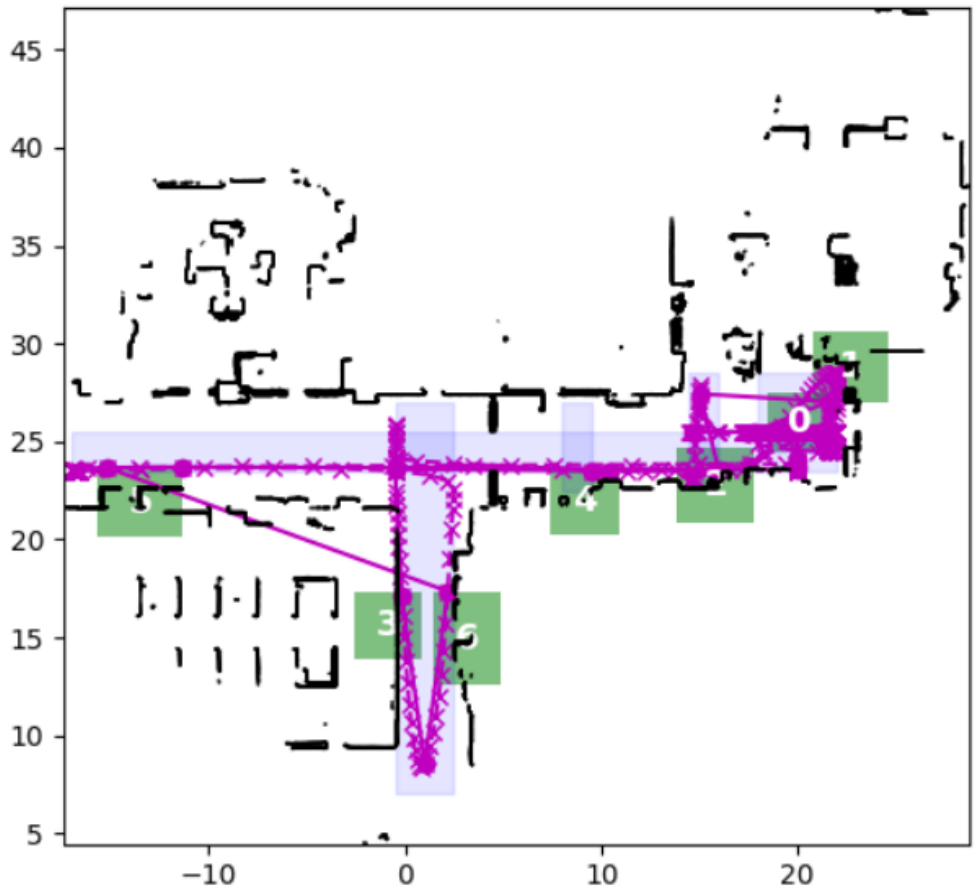

Figure A.1: Visualization of full plant watering trajectory.

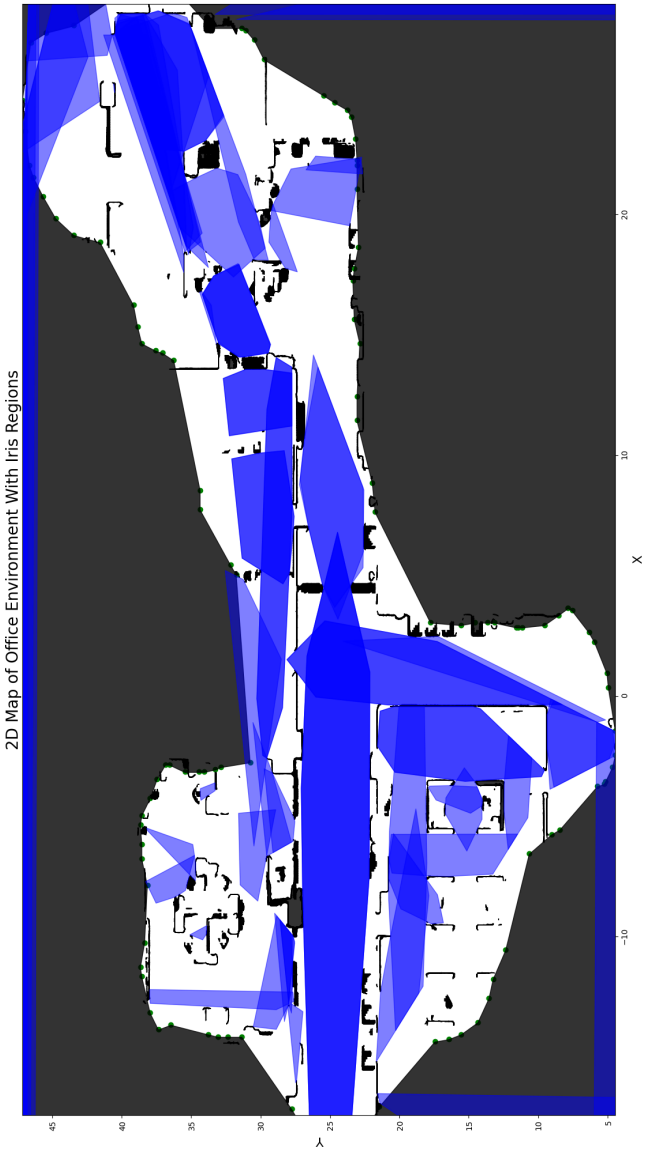

Figure A.2: Visualization of safe regions generated by IRIS in the office environment.

References

[1] I. Lluvia, E. Lazkano, and A. Ansuategi, "Active mapping and robot exploration: A survey," *Sensors*, vol. 21, no. 7, 2021, ISSN: 1424-8220. DOI: 10.3390/s21072445. URL: https://www.mdpi.com/1424-8220/21/7/2445.

[2] D. Silver and J. Veness, "Monte-carlo planning in large pomdps," in *Advances in Neural Information Processing Systems*, vol. 23, Vancouver, Canada, 2010, pp. 2164–2172.

[3] M. Tsuru, A. Escande, A. Tanguy, K. Chappellet, and K. Harad, "Online object searching by a humanoid robot in an unknown environment," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2862–2869, Apr. 2021. DOI: 10.1109/LRA.2021.3061383.

[4] M. Kim and I. H. Suh, "Active object search in an unknown large-scale environment using commonsense knowledge and spatial relations," *Intelligent Service Robotics*, vol. 12, pp. 371–380, 2019. DOI: 10.1007/s11370-019-00288-5.

[5] X. Ye, Z. Lin, H. Li, S. Zheng, and Y. Yang, "Active object perceiver: Recognition-guided policy learning for object searching on mobile robots," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018, pp. 6857–6863.

[6] P. Kim, B. C. Williams, and M. Abramson, "Executing reactive, model-based programs through graph-based temporal planning," in *Proceedings of the IJCAI International Joint Conference on Artificial Intelligence*, 2001, pp. 487–493.

[7] M. Reeves, "Magellan: A robust executive enabling long horizon multi-agent campaigns," M.S. thesis, Massachusetts Institute of Technology, 2020.

[8] M. Reeves *et al.*, "Executing multi-goal mission plans for coordinated mobile robots," in *Proceedings of the ICAPS INTEX Workshop*, 2019.

[9] N. Hughes, Y. Chang, and L. Carlone, "Hydra: A real-time spatial perception engine for 3d scene graph construction and optimization," in *Proceedings of Robotics: Science and Systems (RSS)*, 2022.

[10] Y. Chang, N. Hughes, A. Ray, and L. Carlone, "Hydra-multi: Collaborative online construction of 3d scene graphs with multi-robot teams," in *Proceedings of the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Detroit, MI, USA, 2023, pp. 10 995–11 002. DOI: 10.1109/IROS55552.2023.10341838.

[11] Z. Ravichandran, L. Peng, N. Hughes, J. D. Griffith, and L. Carlone, "Hierarchical representations and explicit memory: Learning effective navigation policies on 3d scene graphs using graph neural networks," in *Proceedings of the 2022 International Conference on Robotics and Automation (ICRA)*, Philadelphia, PA, USA, 2022, pp. 9272–9279. DOI: 10.1109/ICRA46639.2022.9812179.

[12] I. Armeni, Z. He, J. Gwak, A. Zamir, M. Fischer, J. Malik, and S. Savarese, "3d scene graph: A structure for unified semantics, 3d space, and camera," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2019, pp. 5664–5673.

[13] U. Kim, J. Park, T. Song, and J. Kim, "3-d scene graph: A sparse and semantic representation of physical environments for intelligent agents," *IEEE Transactions on Cybernetics*, pp. 1–13, Aug. 2019.

[14] A. Rosinol, A. Gupta, M. Abate, J. Shi, and L. Carlone, "3d dynamic scene graphs: Actionable spatial perception with places, objects, and humans," in *Proceedings of Robotics: Science and Systems (RSS)*, 2020. DOI: 10.15607/RSS.2020.XVI.079.

[15] A. Rosinol, A. Violette, M. Abate, N. Hughes, Y. Chang, J. Shi, A. Gupta, and L. Carlone, "Kimera: From slam to spatial perception with 3d dynamic scene graphs," *International Journal of Robotics Research*, vol. 40, no. 12–14, pp. 1510–1546, 2021. arXiv: 2101.06894.

[16] P. Kim, B. C. Williams, and M. Abramson, "Executing reactive, model-based programs through graph-based temporal planning," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, Seattle, WA, 2001, pp. 487–493.

[17] M. Reeves, "Magellan: A risk-bounded plan executive for autonomous mobile agents," Ph.D. dissertation, Massachusetts Institute of Technology, 2024.

[18] J. Strader, N. Hughes, W. Chen, A. Speranzon, and L. Carlone, "Indoor and outdoor 3D scene graph generation via language-enabled spatial ontologies," *arXiv preprint: 2312.11713*, 2023.

[19] T. Marcucci, J. Umenberger, P. Parrilo, and R. Tedrake, "Shortest paths in graphs of convex sets," *SIAM Journal on Optimization*, vol. 34, no. 1, pp. 507–532, 2024. DOI: 10.1137/22M1523790. eprint: https://doi.org/10.1137/22M1523790. URL: https://doi.org/10.1137/22M1523790.

[20] R. L. H. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Workshop on the Algorithmic Fundamentals of Robotics*, Istanbul, Turkey, Aug. 2014.

[21] A. Richards, T. Schouwenaars, J. P. How, and E. Feron, "Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 4, pp. 755–764, 2019/02/26 2002. DOI: 10.2514/2.4943. URL: https://doi.org/10.2514/2.4943.

[22] M. da Silva Arantes, J. da Silva Arantes, C. F. M. Toledo, and B. C. Williams, "A hybrid multi-population genetic algorithm for uav path planning," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO '16, New York, NY, USA: ACM, 2016, pp. 853–860, ISBN: 978-1-4503-4206-3. DOI: 10.1145/2908812.2908919. URL: http://doi.acm.org/10.1145/2908812.2908919.