External Advisor for Combined Task and Motion Planning in Belief Space

Sunny Amatya

DIBRIS - Department of Informatics, Bioengineering, Robotics and System Engineering

University of Genova

In partial fulfillment of the requirements for the degree of

Masters of Science in Robotics

August 30, 2018

Acknowledgements

I would like to acknowledge my supervisors, Associate Professor Marco Baglietto and Assistant Professor Fulvio Mastrogiovanni, for their constant support and feedback. I would like to thank my colleague, Antony Thomas for the research input and guidelines. I would further like to extend my gratitude to Sarah Bernadini and Chiara Piacentini for their continuous support in understanding the state of the art research. Finally I would like to thank my parents for believing in me, my siblings for required sense of humor and friends, Aayush and Antonio, for their love and support.

To dad and mom, for without them I am nothing

Abstract

Planning algorithms are at the heart of the cooperation problems and use various combinations of logic, graph, probabilistic and control theory aspects. The general idea for a planning algorithm has always been to find the least cost path, satisfying a set of given constraints. These constraints may be logical, environmental, kinematic, resource-associated or physical limits. Yet, most often uncertainties exist in such scenarios and planning can only be done in the belief space, the probability distribution over the possible states of the robot. Task planning domains are specified using the Planning Domain Definition Language (PDDL). Techniques like, planning graphs, planning as satisfiability and heuristic-search planning have led to the dominant task planning approaches like fast downward and FF planning systems, capable of extracting highly efficient plans. The field of motion planning has also developed profusely. Sampling based motion planning approaches provides faster execution and better efficiency. Probabilistic roadmaps (PRMs) and rapidly-exploring random trees (RRTs) were two seminal works in this area, laying the foundation for other planners to build on.

Autonomous navigation in complex real world scenarios involve different layers of decision making, a high-level task planning layer to decompose tasks into subtasks and a low-level motion planner to execute these tasks. Therefore planning should be performed in the discrete-continuous space of task and motion planning. Over the last few years significant research has been carried out in the area of combined task and motion planning (TAMP).

Building upon the works in AI and robotics planning, we develop a TAMP planner capable of reasoning in the belief space while extracting a plan. We use PDDL+ to model the discrete planning task with logical predicates and numeric fluents. The problem file for the planner contains all the information regarding the environment being operated upon, given as initial conditions. The PDDL+ planner outputs a sequence of actions (a plan) that can then be passed on to the low-level motion planner for execution. Assuming Gaussian models,

extracting an efficient plan requires performing the belief updates at each planning stage. However PDDL+ is incapable of handling such rigorous numerical calculations within it. Most approaches that need to perform such calculations require an external module or semantic attachments to perform these. Yet these semantic attachments being external calls, blocks the task planning until the calls are returned to the planner. Recently Bernardini et al. developed an approach to implicitly trigger such external calls via external advisors. They also use an approximation heuristic for the fluents that needs an external advisor for its update. This approximation heuristic thus helps guiding the planner faster towards the goal. We incorporate this approach within our planner and provide the plan to the motion planner for execution.

Furthermore, RRT with Potential Field has been introduced for sampling and robust map traversing. The framework is being handled in the ROSPlan for communication between the environment, accumulated state estimation, external solver, task planner and the low-level controller.

Contents

List of Figures

Chapter 1

Introduction

Autonomous robots operating in complex real world scenarios require different levels of planning to execute their tasks. High-level (task) planning helps to break down a given set of tasks into a sequence of sub-tasks, depending on the required level of abstraction. Actual execution of each of these sub-tasks would require low-level control actions (i.e., motions). Hence, planning should be performed in the task-motion or the discrete-continuous space.

In recent years, combining high-level task planning with the low-level motion planning has been a subject of great interest among the Robotics and Artificial Intelligence (AI) community. This is inevitable as one of the ultimate goals in Robotics is to create autonomous agents accepting high-level task descriptions and executing them without further human intervention. Planning frameworks such as the Planning Domain Definition Language (PDDL) [1] mainly focus on high-level task planning supposing that the geometric preconditions (e.g., grasping poses for a pick-up task [2]) for the robot motion to carry out these tasks are achievable. However, in reality, such an assumption can be catastrophic as an action or sequence of actions generated by the task planning algorithm might turn out to be unfeasible at the controller execution level.

Let us consider a simple scenario where a robot is given the task of picking up an object. In terms of task planning, a *pick_up* action would suffice, subjected to satisfying the action preconditions, i.e., the robot hand being free and the object being graspable. However, it is possible that the robot is too close to the object and the *pick_up* action cannot be performed due to robot's end-effector's reachability workspace. This would require the robot to assume a different grasping pose, invoking a motion command that leads to a suitable position. Though a simple scenario, it clearly illustrates the need for a combined TMP strategy. Several recent works [3, 4, 5, 2, 6, 7] have motivated the need for a combined Task and Motion Planning (TAMP) approach pointing out the drawbacks of treating them separately.

Planning algorithms are at the heart of the co-operation problem and use various combinations of logic theory, graph theory, probabilistic theory and control theory. The general idea for a planning algorithm has always been to find the least cost path that satisfies a set of given constraints. These constrains may be logical, environmental, kinematic, resource-associated or physical limits. While operating in complex scenarios, robots encounter uncertainties. Such uncertainties may arise due to insufficient knowledge about the environment (partial observability), inexact robot motion, imperfect sensing. In these scenarios, the robot poses or other variables of interest (states) can only be dealt with in terms of probabilities. Planning is therefore done in the *belief* space, which is the probability distributions over the possible robot states. Such a problem falls under the category of Partially Observable Markov Decision Processes (POMDPs) [8]. To perform efficiently in these scenarios the robot needs to plan for actions, which when executed help gain more information, thereby improving the robot's belief. Hence the task planner should take into account it's current belief while synthesizing a plan. The motion planner might encounter unexpected scenarios notwithstanding the plan provided. This calls for a re-plan, updating the task planner with the new belief, resulting in a cyclic interdependency. Consequently, both task and motion planning are interdependent and should not be considered separate. TAMP presents challenges both in algorithm design and software engineering. Hence a successful co-operation is essential between the task and motion planners.

1.1 Task Planning

Task planning or classical AI planning can be defined as the problem of synthesizing a sequence of actions from the current state, that will achieve a goal state [9]. Since the seminal work of Fikes and Nilsson [10], planning has emerged as a specific field within AI. Their STanford Research Institute Problem Solver (STRIPS) provided a simple yet powerful way of describing a planning problem. STRIPS provided a syntax for writing a planning tasks in terms of action schema for states, with preconditions, add and delete effects for the actions. This syntax provided a platform for the development of frameworks like the Planning Domain Definition Language (PDDL) [1]. PDDL was introduced in AIPS-98 planning competition, a higher level language for planning. The language supported basic strips action, conditional effects, universal quantification over dynamic universe, specification of constraints, hierarchical actions. A PDDL task specification is split into two parts

1. *domain*- which specifies the list of objects, predicates and constants and type of actions available along with its preconditions and effects.

2. *problem*- specifying the initial state and the goal state.

Furthermore, information about the objects in the world and their relationship to one another are required to determine the actions applicable at a given state, via the action preconditions. The action effects change these relationships leading the system to a new state. Since the aim of the planner is to synthesize a set of actions that take the start state to a goal state, efficient task planning requires heuristics. A heuristic is an estimate of the cost from any given state to reach the goal state. Given a planning task, efficient planning systems computes an admissible heuristic first, before proceeding to plan synthesis [11]. The task planner also requires awareness of what is uncertain or unknown, allowing the robot the opportunity to take further actions subject to satisfying certain constraints [12] or properties, providing decision theoretic trade-offs. In this thesis we use a PDDL+ [13] based task planner, which will be discussed in detail in the coming sections.

1.2 Motion Planning

Motion planning can be defined as finding a sequence of collision-free poses (position and orientation) starting from a given initial pose and terminating at the goal pose for a robot in a given configuration space [14]. Motion planning identifies a continuous path and breaks down a desired movement into discrete motion that satisfies movement constraint and optimizes movement aspects. A basic problem in the motion planning requires the production of continuous motion from the start configuration (pose) to goal configuration while avoiding obstacles. Hence the representation of the world, robot configuration and obstacle configuration are essential for the motion planning calculations.

There are three configuration spaces $\mathcal{C}$ to be considered for the motion planning, namely obstacle space $\mathcal{C}_{obs}$, free space $\mathcal{C}_{free}$ and target space $\mathcal{C}_{goal}$. The configuration space is the set of possible states that the robot can assume in the environment. The 3 dimensional (3-D) configuration space of the robot can be represented by 6 parameters, (x, y, z) for position and $(\alpha,\ \beta,\ \gamma)$ for orientation. For the 2-D ground robot, the position and the orientation can be reduced to (x, y) and (θ) respectively, where the rotation is about the z-axis. The obstacle space $\mathcal{C}_{obs}$ is the set of configurations whose intersections with the robot configurations result in null space. $\mathcal{C}_{free}$ represents the set of configurations that avoids the collision with the obstacles, i.e., $\mathcal{C}_{free} = \mathcal{C}/\mathcal{C}_{obs}$. Forward kinematics are used generally to determine if the robot's geometry and position results in collision with the obstacles in the environment. Lastly, the target space is the free space denoted by the goal of the robot. Depending on global or local motion planning, the trajectory to the goal position can go through several discretized way-points based on the observability of the map.

To compute the connectivity of the free space, grid-based algorithms are generally used for low-dimensional problems overlaying a grid on top of configuration space. Sampling-based motion planners are widely used for high-dimensional system [15]. Such sampling-based planners offer probabilistic completeness, guaranteeing that the planner will eventually find a solution if one exists. However, if solution does not exist, a sampling-based planner cannot prove this negative; in such case, the planner would not terminate or would run until a timeout [16]. Probabilistic Roadmap (PRM) [17] and Rapidly-exploring Random Trees (RRT) [18] and the variations of the techniques are used for successful sampling based motion planning. Motion planners based on gradient descent or optimization are also common and highly efficient, but they do not offer the same probabilistic completeness guarantees as the sampling-based motion planners. Sampling based planners offer probabilistic completeness guarantees because probabilistic completeness of the overall framework is a desired property.

Future integration of alternate motion planning approaches is possible with their accompanying set of trade-offs. However, the integration of motion planners in TAMP needs special attention to address the coupling of task planning and motion planning.

1.3 Combined Task and Motion Planning

TAMP combines the discrete action selection of task planning with the continuous path generation of motion planning. A functioning requirement of the planner is to establish a correspondence between task operators and the associated motion planning. The task-motion interface must translate between the low-level scene geometry and the high-level task descriptions. For a given problem which includes the geometry and the environmental states, the related task description should be formulated. Furthermore, given a task action, the corresponding motion planning problem must be addressed.

A further requirement of the task-motion planning is the requirement of completeness in planning. Motion planning approaches can only provide probabilistic completeness. As a result the failure of motion plan does not ascertain the corresponding task plan to be infeasible. The motion planning might have failed because of non-existent path or due to the insufficient planner run time. As a result, we cannot definitively rule out an attempted task action because a motion planner was unable to find a concrete path in the allotted time. Thus, to ensure probabilistically complete TAMP, we must not eliminate failed task actions but instead set them aside to be later reattempted.

Compared to just the traditional task planning approaches, the task planning phase of TAMP should support generation of alternate plans. Iterating between task planning and motion planning phases, the feedback from motion planner influences selected operation which would otherwise be infeasible. The task planner must therefore be able to compute alternate plans for a given domain and ideally reuse the plans for improved performance.

Furthermore, the task planner must support a sufficiently expressive specification format to model the desired domain. Therefore, a general robotics task and motion planner would ideally be independent of the particular domain specification syntax, enabling the use of the most suitable notations like PDDL, temporal logics, etc. Motion planning frameworks functions with the provided robot kinematics, assuming that no change in kinematics equations and changing configurations only when the robot is moving. In contrast, TAMP requires rapid updates to kinematic equations; as the robot performs an action, consequently a change in kinematics or robot configuration is reflected. Consequently, kinematic representations capable of efficient updates are required for TAMP. Even the best laid plans often go awry. A robot must therefore execute its plans in a robust manner, reacting quickly to minor disturbances, and re-planning when faced with major unexpected changes.

1.4 Organization

The rest of this thesis is organized as follows:

1. We discuss the relevant background required to formulate our problem and briefly discuss the current state-of-the-art in **chapter 2**.

2. In **chapter 3** we formulate our TAMP problem.

3. In **chapter 4** we discuss the key aspects of our method using a synthetic simulation in Gazebo.

4. **Chapter 5** provides concluding remarks and discuss potential future work.

Chapter 2

Background and State-of-the-art

Given an initial state and a suitable goal state, Task and Motion Planning (TAMP) synthesizes a plan interleaving discrete high-level tasks with continuous low-level motion. In robotics planning, reaching a goal pose from an initial pose requires continuous collision-free motion planning. However, reaching a goal alone is not sufficient as most often we require the goal condition to be satisfied subjected to mission dependent costs. As a result, certain decisions are to be made with regards to visiting specific landmarks, the order, and type of actions. Planning and decision making should hence be performed by combining the continuous, collision-free motion planning with the discrete task planning actions. Yet, the most important challenge for TAMP problems is finding the right correspondence between the task planner and the motion planner. Given a discrete action, TAMP planner should be able to recognize the corresponding geometry requirements to trigger the action. Similarly, once the action is triggered, the corresponding motion planning problems are to be identified.

In this chapter, we discuss the related work and other required background for introducing our TAMP algorithm in Chapter 3.

2.1 Related Work

The genesis of TAMP can be credited to Fikes and Nilsson for their work on STRIPS [10] which further led to the Shakey project [19]. Shakey's planner had access to basic geometric knowledge like the objects in a room, connectivity between rooms etc. However, it performed a logical search first, assuming that the resulting robot motion plans can be formulated. This assumption limits the capability of the agent as the high-level actions may turn out to be non-executable due to geometric limitations. The later works either carried out the logical plans created by validating them using a robot motion planner [20] or performed a combined search in the logical and geometric spaces using a state composed of the both the symbolic and geometric paths [3]. The aSyMov planner used in [3] uses a combination of Metric-FF [21] a sampling based motion planner. In contrast, [22] use a hybrid temporal task planner incorporating robot state uncertainty. Srivastava et al. [2] implicitly incorporate geometric variables within the PDDL framework. An interface layer then converts the PDDL plans to numeric values of the symbols to check the validity of each action in the configuration space. In practice, many of the planning problems need to plan well ahead of time which also results in increased dimensionality, as more and more objects and constraints get added. Longer planning horizons and higher dimensionality directly affects the computational time for these planners. Kaelbling and Lozano-Péres [4] proposed a hierarchical approach that tightly integrates the logical and geometric planning. The complexities arising out of long horizon planning are tackled to an extent since the planning is done at different levels of abstraction, thereby reducing the long horizons to a number of feasible sub-plans of shorter horizon. This regression based planner assumes actions are reversible actions while backtracking. In contrast to their earlier work [23] the serializability assumption of the subgoals are relaxed. Kaelbling and Lozano-Péres [24] further extended their work to consider the current state uncertainty, modeling the planning problem in the belief space planning problem. Uncertain outcomes are modeled by converting a Markov decision processes (MDP) into a weighted graph, thereby modifying their earlier approach of *hierarchical planning in the now*. Belief update is then performed when observations are obtained. [25] discusses an ongoing work for TMP under partial observability, computing long-horizon policies that are arborescent in nature.

The above discussed approaches focus on finding feasible plans sacrificing optimality and hence emphasize on the performance. The work of Toussaint [6] is intriguing in the sense that optimization is performed over an objective function based on the final geometric configuration (and the cost thereby), finding approximately locally optimal solution by minimizing an objective function. The planning problem is modeled as a constraint satisfaction problem with symbolic states used to define the constraints in the optimization. [26] models the motion planning as a constraint satisfaction problem over a subset of the configuration space. Iteratively Deepened Task and Motion Planning (IDTMP) [7] is a constraint based task planning approach that incorporates geometric information (motion feasibility) at the task planning level. In our proposed algorithm, the waypoints fed into the task planner are generated using the motion planner, similar to the motion planner information that guides the IDTMP task planner.

2.2 Task Planner

To model the task domain we use an extension of the standard modeling language, PDDL which has gained much popularity among the Automated Planning community. PDDL+ is an extension of PDDL ([1], [27], [28]) which provides more modeling flexibility through the use of autonomous processes and events. Unlike its predecessors, PDDL+ is able to model the interaction between the agent's behavior and changes that are initiated by the world. The inclusion of process that runs over time allows to have a continuous numeric effect in the system. PDDL+ distinguishes processes responsible for continuous change, from events and actions responsible for discrete changes. The processes are similar to durative actions and the events are akin to instantaneous actions. However, processes and events are distinct from actions since a process or an event is triggered as soon as its precondition is satisfied whereas an action trigger depends on the planner search strategy. Hybrid automaton, which is a formal model for a mixed discrete-continuous systems, forms the basis for the development of the PDDL+ semantics. This formalization of PDDL+ leveraging hybrid automata paradigm bridges the gap between planning and real time systems.

2.2.1 Heuristics in PDDL+

Hybrid domain models capture a more accurate representation of real world problems which involve continuous process than possible using discrete systems. However, solving problems represented as PDDL+ domains is very challenging due to the construction of complex system dynamics, including non-linear process and events. Unlike the traditional planners based on PDDL2.1, PDDL+ based planners should be able to accommodate events and processes to reason with discrete and continuous changes caused by actions and exogenous events. Such an approach is computationally expensive if all the possible states are explored and hence the planners should resort to efficient heuristic based search strategies.

One of the earlier planners, TM-LPSAT [29] is a fully automated planner that can solve PDDL+ planning problems with linear continuous change. It uses a SAT-based compilation which uses an LP solver to manage numeric constraints while giving a discrete set of time points. However, this approach has limitations in scalability.

UPMurphi [30] uses the planning as model-checking paradigm based on a hand-crafted discretization of time, which enables it to handle non-linear continuous changes. However, it requires human expertise to get a good discretization and lacks heuristic guidance which adds to state space explosion. Hence, UPMurphi suffers from scalability issues as well.

POPF2[31] extended from POPF [32] planner is capable of reasoning with

PDDL+ events and linear processes. As a forward chaining planner, the heuristic and enforcing techniques are used to collect best heuristic in each Relaxed Plan Graph(RPG) state. Only the best heuristic is expanded forward to the next state. This way it selects the first plan approved.

DiNo[33] is based on UMurphi [30] which is based on the Discrete and Validate approach. Using planning as a model checking paradigm and relying on discretize & validate approach, DiNo further uses Staged Relaxed Plan Graph+ (SRPG+), a novel relaxation-based domain-independent heuristic. The heuristics help guide the search plan for the goal based approach. DiNo further exploits the deferred heuristic evaluation for completeness of plan in a discretized search space with finite horizon. The planner is able to handle linear as well as non-linear processes.

SMT+ [34] planner is motivated by numerous SAT-based planning approaches, which uses SMT encoding on the domains with nonlinear continuous change. Happening are introduced to capture the change in state at a given point in time due to the effects of actions, processes or events. A plan is found by building a formula with multiple copies of the set of variables. Each Happening models discrete change. Between Happenings there is only continuous numerical change. The initial state is modeled and the goal constraints are added.

2.2.2 Semantic Attachments in Task Planning

Automating the navigation through the provided environment and grid requires a range of planning and optimization techniques. PDDL+ [13] is used to model the planning task, providing the robot a sequence of actions (a plan) that can be passed on to the low-level motion planner for execution. Assuming Gaussian models, extracting an efficient plan requires performing the belief prediction and update within PDDL+ effects. Nonetheless, PDDL+ is restricted, as it is incapable of handling rigorous numerical calculations. Most approaches perform such calculations via an external module or semantic attachments, e.g. [5]. These semantic attachments are explicit external calls that perform numerical calculations during planner runtime. Yet, the effects returned by these semantic attachments are not exploited in identifying *helpful actions* and hence do not provide any heuristic guidance.

PREFPEA* planner [35] takes early ideas from STRIPS and uses Relaxed Plan Heuristics in the Relaxed Plan Graph using Iterative Landmark Algorithm. There are primary and secondary pre-conditions for actions, the action applied will directly affect the primary variables. For example, in block world the shifting of block from one column to next column, the secondary variables could be the volume of the liquid in the column related to the weight of the blocks in calculation of heuristics. This secondary variable calculation has been done in semantic attachment. Partial Expansion A* (PEA*) algorithm has been used for the shortest path with the staged expansion which helps in avoiding heuristic evaluation in sibling node. Temporal aspects are not analyzed in the domain.

Recently [36] developed a PDDL based POPF-TIF planner to implicitly trigger such external calls via external advisors. These *external advisors* are specialized semantic attachments customized to solve specific problems given list of variables and conditions. They classify variables into direct, indirect and free variables. Direct (free) variables are the normal PDDL function variables whose values are changed in the action effects, in accordance with the PDDL semantics. The indirect variables are affected by the changes in the direct variables. A change in a direct variable triggers the external advisor which in turn updates the indirect variables. This planner is based on a temporal extension of the metric-FF planner [21]. An intriguing feature of the POPF-TIF planner is its ability to approximate the values of the indirect variables at the Temporal Relaxed Plan Graph (TRPG) construction stage. Approximate effects of the indirect variables are used while constructing the TRPG. Since the TRPG construction at each state is performed to extract the heuristic value for that state, the approximation values help in an efficient goal-directed search. During the forward state space search the external advisor is called, updating the indirect variables with the exact values. POPF-TIF planner is based on PDDL2.1, hence processes and events cannot be programmed and mixed discrete-continuous planning is not possible.

2.3 Preliminaries

Let x_k denote the robot pose at any time k defined by $x_k \doteq (p_k, q_k, \theta_k)$, p_k and q_k are the robot Cartesian coordinates and θ_k is the heading. We use z_k to denote the measurement acquired at time k and u_k for the control action applied at time k. We use Extended Kalman Filter (EKF) to represent the robot belief at a given time, using the robot state mean and covariance. The standard odometry based motion model [37] is used to represent the robot dynamics. This nonlinear model is then linearized around the current state and control, to predict the next states which are then updated (corrected) using the measurement(s) obtained. In following sections, we discuss in detail the underlying model used in 2.3.2.

2.3.1 Extended Kalman Filter

The general form for the robot motion and observation models are given by

$$x_{k+1} = f(x_k, u_k, w_k) \ , \ w_k \sim \mathcal{N}(0, W_k) \tag{2.1}$$

$$z_k = h(x_k) + v_k \ , \ v_k \sim \mathcal{N}(0, Q_k) \tag{2.2}$$

where $w_k \sim \mathcal{N}(0, W_k)$ is the Gaussian noise in motion and v_k, Q_k are the noise in the measurement and the corresponding covariance.

The models 2.1, 2.2 can be written probabilistically as $p(x_{k+1}|x_k, u_k)$ and $p(z_k|x_k)$ respectively. Given the current state $x_k \sim \mathcal{N}(\mu_k, P_k)$, the control u_k, the next state parameters, namely the predicted mean $\bar{\mu}_{k+1}$ and the predicted covariance $\bar{P}_{k+1}$ can be computed using the standard EKF prediction as

$$\begin{aligned}\bar{\mu}_{k+1} &= f(\mu_k, u_k)\\ \bar{P}_{k+1} &= F_k P_k F_k^T + V_k W_k V_k^T\end{aligned} \tag{2.3}$$

where F_k is the Jacobian of $f(.)$ with respect to x, V_k is the Jacobian of $f(.)$ with respect to u. For ease of representation, we denote $R_k \doteq V_k W_k V_k^T$. Upon receiving a measurement z_k, the updated parameters using EKF are given by

$$\begin{aligned}K_k &= \bar{P}_k H_k^T (H_k \bar{P}_k H_k^T + Q_k)^{-1}\\ \mu_{k+1} &= \bar{\mu}_{k+1} + K_k(z_{k+1} - h(\bar{\mu}_{k+1}))\\ P_{k+1} &= (I - K_k H_k)\bar{P}_k\end{aligned} \tag{2.4}$$

where H_k is the Jacobian of $h(.)$ with respect to x, K_k is the Kalman gain and I is a 3×3 identity matrix.

2.3.2 System Modeling

The position of the mobile robot given by the planar Cartesian coordinates (p, q) together with its heading θ determine the pose of the robot. So our state variable at time k is the pose $x_k \doteq (p_k, q_k, \theta_k)$. The robot dynamics is modeled using the standard odometry based motion model

$$\begin{aligned}p_{k+1} &= p_k + \delta_{trans_k} \cdot \cos(\theta_k + \delta_{rot1_k})\\ q_{k+1} &= q_k + \delta_{trans_k} \cdot \sin(\theta_k + \delta_{rot1_k})\\ \theta_{k+1} &= \theta_k + \delta_{rot1_k} + \delta_{rot2_k}\end{aligned} \tag{2.5}$$

where $u_k \doteq (\delta_{rot1_k}, \delta_{trans_k}, \delta_{rot2_k})$ is the control applied. The model 2.5 assumes that the robot implements the following commands in order: rotation by an angle of δ_{rot1_k}, translation of $\delta_{translation_k}$ and a final rotation of δ_{rot2_k} orienting the robot in the required direction (see Figure 2.1). These control parameters are calculated as

$$\begin{aligned}\delta_{rot1_k} &= \arctan(\frac{q_{k+1} - q_k}{p_{k+1} - p_k}) - \theta_k\\ \delta_{trans_k} &= \sqrt{(p_{k+1} - p_k)^2 + (q_{k+1} - q_k)^2}\\ \delta_{rot2_k} &= \theta_{k+1} - \theta_k - \delta_{rot_1}\end{aligned} \tag{2.6}$$

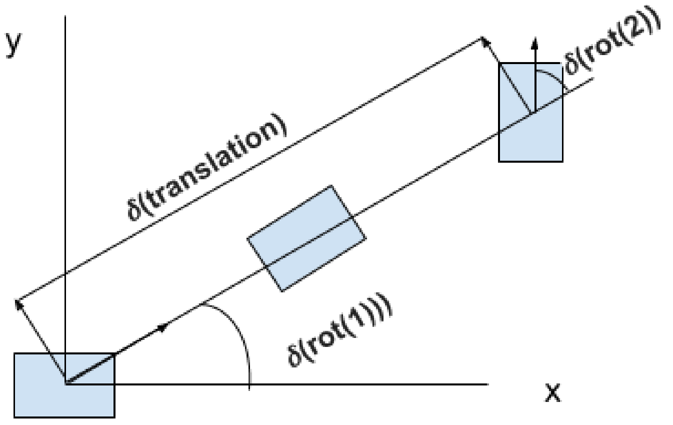


Figure 2.1: Control parameters in odometry motion model.

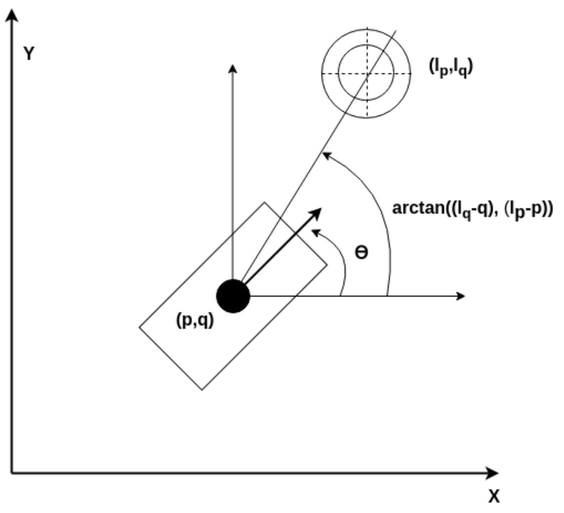


Figure 2.2: Robot's measurement model

To process the landmarks in the environment we measure the range and the bearing of the landmark relative to the robot's local coordinate frame. It is to be noted that we assume the data association problem is solved and hence given a measurement we know the corresponding landmark that generated it. Such a model (see Figure 2.2) can be represented by

$$z_k = \begin{bmatrix} r = \sqrt{(l_p - p_k)^2 + (l_q - q_k)^2} \\ \\ \phi = \arctan(\frac{l_q - q_k}{l_p - p_k}) - \theta_k \end{bmatrix} + v_k \ , \ v_k \sim \mathcal{N}(0, Q_k) \tag{2.7}$$

where (l_p, l_q) is the landmark coordinate. In the remainder of this section we derive the Jacobian matrices discussed in section 2.3.1 for our motion (Eq. 2.5) and observation (Eq. 2.7) models.

For the motion model given in Eq. 2.5, the Jacobian with respect to the state x_k denoted by F_k is given by

$$F_k = \begin{bmatrix} \frac{\partial f}{\partial x_k} & \frac{\partial f}{\partial y_k} & \frac{\partial f}{\partial \theta_k} \\ \frac{\partial f}{\partial x_k} & \frac{\partial f}{\partial y_k} & \frac{\partial f}{\partial \theta_k} \\ \frac{\partial f}{\partial x_k} & \frac{\partial f}{\partial y_k} & \frac{\partial f}{\partial \theta_k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\delta_{trans_k} \cdot \sin(\theta_k + \delta_{rot1_k}) \\ 0 & 1 & \delta_{trans_k} \cdot \cos(\theta_k + \delta_{rot1_k}) \\ 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

Similarly, the Jacobian with respect to u_k is given by

$$V_k = \begin{bmatrix} \frac{\partial f}{\partial \delta_{rot1_k}} & \frac{\partial f}{\partial \delta_{trans_k}} & \frac{\partial f}{\partial \delta_{rot2_k}} \\ \frac{\partial f}{\partial \delta_{rot1_k}} & \frac{\partial f}{\partial \delta_{trans_k}} & \frac{\partial f}{\partial \delta_{rot2_k}} \\ \frac{\partial f}{\partial \delta_{rot1_k}} & \frac{\partial f}{\partial \delta_{trans_k}} & \frac{\partial f}{\partial \delta_{rot2_k}} \end{bmatrix} = \begin{bmatrix} -\delta_{trans_k} \cdot \sin(\theta_k + \delta_{rot1_k}) & \cos(\theta_k + \delta_{rot1_k}) & 0 \\ \delta_{trans_k} \cdot \cos(\theta_k + \delta_{rot1_k}) & \sin(\theta_k + \delta_{rot1_k}) & 0 \\ 1 & 0 & 1 \end{bmatrix} \tag{2.9}$$

Furthermore, the noise covariance matrix W_k is formulated as below

$$W_k = \begin{bmatrix} \alpha_1 \cdot \delta^2_{rot1_k} + \alpha_2 \cdot \delta^2_{trans_k} & 0 & 0 \\ 0 & \alpha_3 \cdot \delta^2_{trans_k} + \alpha_4 \cdot (\delta^2_{rot1_k} + \delta^2_{rot2_k}) & 0 \\ 0 & 0 & \alpha_2 \cdot \delta^2_{trans_k} + \alpha_1 \cdot \delta^2_{rot2_k} \end{bmatrix} \tag{2.10}$$

where α_1 to α_4 are robot-specific error parameters [37] modeling the accuracy of the robot motion. As evident from the matrix above, greater accuracy implies smaller values in α's.

The measurement matrix H_k for the model in Eq. 2.7 is obtained by taking the Jacobian of $h(.)$ (see Eq. 2.2) with respect to x_k

$$H = \begin{bmatrix} \frac{\partial r}{\partial x_k} & \frac{\partial r}{\partial y_k} & \frac{\partial r}{\partial \theta_k} \\ \frac{\partial \phi}{\partial x_k} & \frac{\partial \phi}{\partial y_k} & \frac{\partial \phi}{\partial \theta_k} \end{bmatrix}_{x_k} = \begin{bmatrix} -\frac{(l_p - p_k)}{r} & -\frac{(l_q - q_k)}{r} & 0 \\ \frac{(l_q - q_k)}{r^2} & -\frac{(l_p - p_k)}{r^2} & -1 \end{bmatrix} \tag{2.11}$$

2.3.3 System Parameters

The The initial state x_0 is taken to be equal to the first position of the robot, in our case $x_0 = [0, 0, 0]$. The initial state error covariance matrix is initialized to the value of the expected system error noise covariance $P_0^- = Q$, where

$$Q_k = V_k W_k V_k^T \tag{2.12}$$

During the predict state, we have to calculate the value of matix W_k. Here α_1, α_2, α_3 and α_4 are robot related parameters[37]. For our calculation we will be using values 0.05, 0.005, 0.1, 0.01 respectively for each α value. To being with, the position noises are assumed to be uncorrelated and time-invariant, therefore the matrix only has time-invariant diagonal values. The standard deviation of position noise in the system is 60 cm for x & y position. Allowing the angle to be 8 degrees in orientation, covariance matrix P is as follows:

$$P = \begin{bmatrix} 0.6 & 0.0 & 0.0 \\ 0.0 & 0.6 & 0.0 \\ 0.0 & 0.0 & 0.02 \end{bmatrix}_{x_k}$$

For the update the distance (d_i) and bearing angle (α_i) measurements put using the standard variance in the system. To begin with, we assume that the variance of the distance is 0.025m. For the bearing, we assume the initial variance of 0.001 radian. Hence, the noise covariance entries are:

$$R = \begin{bmatrix} 0.025 & 0.0 \\ 0.0 & 0.001 \end{bmatrix}_{x_k}$$

In absence of the sensor reading, we use the range and bearing calculation and introduce randomness in each measurement. With maximum 8 cm in distance and for the bearing, the variance of -5 to 5 degrees is introduced.

2.4 Integrated Task and Motion Planning

Combining task and motion planning has been a widely researched field. Different platforms have been used to integrate task and motion planning. TAMP problem can be considered as an AI problem where heuristics are implemented over given constraint satisfying certain goals.

Combined Task and Motion Planner (CTMP) [38] compile task and motion planning problems into probabilistically complete classical AI planning problems by planning over finite and discrete state spaces with a known initial state, deterministic actions, and goal states. A collision checking and motion planner is implemented in execution stage. Functional STRIPS [39] are used for programming to express *procedures* and *constraints* while best-first width search (BFWS) has been used for searching. The simulation is performed in standard benchmark and simulated on ROS, Gazebo and MoveIt.

[40] integrate task planning and motion planning in hierarchical form in real time simulation. Limiting the length of the plan by committing in a top-down

fashion and being goal driven, the planner operates on detailed continuous geometric representations rather than a-priori state-space discretization. Constructing plan at an abstract level rather than detailed construction of the state-space enables the planner to recursively plan and execute actions to achieve the first step in the abstract plan. Representing the task operation in STRIPS like format, the required motion plan is generated using A* search based standard regression-planning algorithm that works backward from the goal. This algorithm is used to solve single planning sub-problems. This works by generating sub-goals that are the weakest precondition of the goal under each applicable action.

[2] integrates off-the-shelf task and motion planners without using task-specific heuristics or hierarchical information besides initial primitive PDDL actions. This also allows the integration model to scale with different task or motion planners. Their task planners are handled using the PDDL framework while motion plan should be expressible as logical predicates at task level to handle failure cases. Their integration increases flexibility of the platform to build and benefit from the literature and advances in different task and motion planners.

Unlike most of the Task and Motion Planning systems, ROSPlan [41] is a framework that allows integration of different types of task planner and motion planner as well as implement the results on a working environment. ROSPlan is based on Robot Operating System (ROS) framework which is a set of software libraries and tools used to build robotic systems. Known for a distributed and modular design, ROS leverages these modules and nodes for function and data sharing as well as storing. In ROSPlan, using environment model, task and motion planning is done to minimize cost while relying on less constrained domains to push forth intelligent and robust behaviour. For visualization of module and results, ROSPlan uses Gazebo and Rviz which are modules based on ROS that simulates model environment and provides feedback for the robot.

Chapter 3

Problem Formulation and Implementation

Planning in hybrid domains is a challenging problem that has found increasing attention in the Planning community, mainly motivated by the need to model real-world domains. In addition to modeling discrete changes, hybrid domains allows us to model continuous temporal behavior using continuous variables that evolves over time. PDDL+ planning framework allows us to model mixed discrete-continuous changes in the environment as required by the Task and Motion Planning paradigm. PDDL+ can handle non-linear changes while many PDDL+ based planners are not able make use of the full set of PDDL+ features. In our case, we use the DiNo planner since it enables heuristic search for linear and non-linear systems using the entire set of PDDL+ features.

Furthermore, planning in belief space requires computationally expensive algorithms with much higher complexities. For example it is required to perform matrix operations and PDDL based planning frameworks are incapable of carrying out the same. To facilitate such computations, we introduce an external advisor which can also handle such calculations for non-linear changes in the planning phase. To automate and integrate the system further, ROSPlan framework has been altered and utilized which allows merging of PDDL+, our planner, external advisor and visualization tools. Below we discuss the formalism of our external advisor for TAMP.

3.1 Domain and Problem Description

In this thesis, we consider a mobile robot in a known environment (i.e., map is given) with uncertainty in its initial pose. The set of landmarks in the environment are given by $l = \{l^1, l^2, ..., l^n\}$. The landmarks are features in the environment and are not to be confused with the landmarks in heuristic planning where they are intended as a set of operators such that each plan must contain some element of this set. The goal is to reach a certain final state x_g with the localization uncertainty not greater than a given bound. Starting from an initial pose, the corresponding goal leading plan is to be synthesized, minimizing the makespan. We use the trace of the state covariance to quantify the uncertainty. The uncertainty condition is mathematically written as $\mathrm{Tr}(\Sigma_k) < \eta$. To incorporate belief evolution while planning, the DiNo planner is extended to support external calls evaluating the belief at each planning stage. This procedure is summarized in Algorithm 1. The belief, the process and the measurement noise are assumed to be Gaussian.

The focal elements for modeling our planning domain are given below

- Planner discretization Δ
- The action *goto_ waypoint*
- The event *belief_ update* that triggers the external call to perform the belief updates
- The process *odometry* that simulates the robot motion between each planner discretization Δ.

In the following sections, we elucidate these elements in the context of our TAMP planner.

3.1.1 Belief Updates with Semantic Attachments

State uncertainty is incorporated in our model and synthesizing an efficient plan requires performing the belief updates within the task planner. PDDL+ processes enable the simulation of robot motion with time and the events are leveraged to perform the corresponding belief updates.

The are a number of ways to model the domain that is consistent with our requirement. For initial testing, we considered the belief prediction as an event which is fired at every time passing action. If the precondition for the event is satisfied, the operation is carried out in external advisor. As shown in 3.1.1.1, we introduce action **goto_waypoint** which takes our defined robot and waypoints as parameters. The preconditions for the robot to go to a waypoint are that the system is in **observed** state, the selected robot is at the waypoint **from** and the waypoint **to** has not been visited. The effect of the action is to move the robot, initiate the process **odometry** effect that simulates the robot motion and reset **observed** state to false. Here we also assign **counter** variable to zero to notify that a new **goto_waypoint** action has been initiated. Furthermore, we increase

Algorithm 1 Problem Statement

Input: Number of waypoints N, initial pose wp_0, goal pose $wp\_\mathbf{k}$ $\mathbf{k} \in \{1, ..., N\}$, distance between the waypoints $relativeD$, discretization distance for prediction $dFactor$, trace of initial pose covariance cov, process discretization Δ, upper bound on the goal trace $finalTrace$

```
1: $robot\_at\ wp\_0$
2: while ($cov > finalTrace$ && $\neg robot\_at\ goalpose$) do
3:   $go\_to\_wp\_i \leftarrow robot\_at\ wp\_j$
4:   $relativeD = $ distance$\{wp\_j, wp\_i\}$
5:   while $relativeD > -(\Delta * dFactor)$ do
6:     $relativeD \leftarrow \{relativeD - (\Delta * dFactor)\}$
7:     $cov \leftarrow EKF\_predict\{cov\}$
8:     for $l \in landmark$ do
9:       if observable$\{l\}$ then
10:        $cov \leftarrow EKF\_update\{cov, l\}$
11:      end if
12:    end for
13:  end while
14:  $robot\_at\ wp\_i$
15: end while
16: return $\{cov\}$
```

predict_covariance and **update_covariance** by zero, so that the variables are kept in use without altering their actual values. This is necessary to maintain the value to the variables between different continuous actions.

We also initiate **predict_covrariance**, **update_covariance** and **cov** as functions in domain, because the planner creates variables and methods related to the variables only if they are defined as functions. For time-passing action the planner triggers all satisfying events and processes. In our initial case **belief_predict** event increases **covariance** while update action decreases **covariance**. The **odometry** process then decreases distance by a factor of time-step. Once the relative distance has been covered, action reached is triggered. This action sets the robot at **to** waypoint and marks it as visited. Based on these initial settings, planner finds a valid trajectory to go to the goal endpoint described in 3.1.1.2. In initial problem, waypoints declaration has been done manually. An over-sight that all waypoints are connected has been made for simplicity. **dFactor**, our discretization length and **finalTrace**, our maximum acceptable covariance are assigned in the problem file. The initial covariance of the system, **cov**, **predict_covariance**, **update_covariance** are also instantiated. In this mode, **cov** is the system's current covariance while **predict_covariance** and **update_covariance** is the value by which system covariance is updated after each predict or update is sent from semantic attachment. Predict is an event based handle here, which takes place at each **time_passing** event and changes system covariance by **predict_covariance** value while also updating the position by **dFactor**. However, update is an action that takes place only when a waypoint is reached, thus making it a single event per goto action. This is a set limitation of the initial domain format.

3.1.1.1 Initial Domain Declaration

```
(define (domain landmark)
(:requirements :typing :durative-actions :fluents :time :strips :
   ↪ disjunctive-preconditions :durative-actions
:negative-preconditions :timed-initial-literals)

(:types
  waypoint
  robot
)

(:predicates
       (robot_at ?r - robot ?wp - waypoint)
       (visited ?wp - waypoint)
       (observe)
       (moving ?r - robot ?to - waypoint)

)

(:functions (distance ?wp1 ?wp2 - waypoint) (cov) (dFactor)
(finalTrace) (relativeD) (counter) (predict_covariance)
(update_covariance)
)
;; relativeD- a variable to store the distance between wps,
;; dFactor- distance factor, to see how many times
;; kalman prediction to be done
;; cov is the initial covarianc trace,
;; finalTrace- the required trace upon reaching the goal state
;; Move between any two waypoints, along the straight line
;; between the two waypoints
```

```
(:action goto_waypoint
  :parameters (?r - robot ?from ?to - waypoint)
  :precondition (and (robot_at ?r ?from) (observe)
  (not(robot_at ?r ?to)) (not (visited ?to )))
  :effect (and (not (robot_at ?r ?from))
           (assign (relativeD) (distance ?from ?to))
           (moving ?r ?to)
           (not (observe))
           (assign (counter) 0)
           (increase (predict_covariance) 0)
           (increase (update_covariance) 0)
           )
)

(:action reached
    :parameters(?r - robot ?to - waypoint)
    :precondition(and (moving ?r ?to) (<= (relativeD) 0))
    :effect(and (robot_at ?r ?to) (visited ?to)
    (not (moving ?r ?to)))
)

(:event belief_predict
  ;;:parameters(?r - robot ?to - waypoint)
  :parameters()
    :precondition(and (= (counter) 1) )
    :effect (and
             (increase (cov) (predict_covariance))
             (assign (counter) 0)
        )
)

(:action belief_update
  :parameters (?r - robot ?to - waypoint)
  :precondition (and (robot_at ?r ?to) (not(observe)) )
  :effect (and
         (decrease (cov) (update_covariance))
         (observe)
         )
)
```

```
;; to calculate the number of prediction steps needed
(:process odometry
  :parameters()
  :precondition (and (> (relativeD) (-dFactor)) )
  :effect (and (decrease (relativeD) (* #t (dFactor)))
  (increase (counter) (* #t 1))
)
)
)
```

3.1.1.2 Initial Problem Declaration

```
(define (problem landmark_prb)
(:domain landmark)
    (:objects
    kenny - robot
    wp0 wp1 wp2 wp3 wp4 wp5 - waypoint
)
    (:init
    (= (distance wp0 wp1) 4)
    (= (distance wp0 wp2) 3)
    (= (distance wp0 wp3) 2.82843)
    (= (distance wp0 wp4) 6.32456)
    (= (distance wp0 wp5) 5.09902)
    (= (distance wp1 wp0) 4)
    (= (distance wp1 wp2) 5)
    (= (distance wp1 wp3) 2.82843)
    (= (distance wp1 wp4) 8.48528)
    (= (distance wp1 wp5) 5.83095)
    (= (distance wp2 wp0) 3)
    (= (distance wp2 wp1) 5)
    (= (distance wp2 wp3) 2.23607)
    (= (distance wp2 wp4) 3.60555)
    (= (distance wp2 wp5) 2.23607)
    (= (distance wp3 wp0) 2.82843)
    (= (distance wp3 wp1) 2.82843)
    (= (distance wp3 wp2) 2.23607)
    (= (distance wp3 wp4) 5.65685)
    (= (distance wp3 wp5) 3.16228)
    (= (distance wp4 wp0) 6.32456)
    (= (distance wp4 wp1) 8.48528)
    (= (distance wp4 wp2) 3.60555)
    (= (distance wp4 wp3) 5.65685)
    (= (distance wp4 wp5) 3.16228)
    (= (distance wp5 wp0) 5.09902)
    (= (distance wp5 wp1) 5.83095)
    (= (distance wp5 wp2) 2.23607)
    (= (distance wp5 wp3) 3.16228)
    (= (distance wp5 wp4) 3.16228)
    (robot_at kenny wp0)
    (= (cov) 1.02)
    (= (relativeD) 0)
    (= (finalTrace) 0.2)
    (= (dFactor) 0.5)
    (observe)
    (= (predict_covariance) 1.02)
    (= (update_covariance) 1.02)
    )
     (:goal (and (robot_at kenny wp5) (< (cov ) finalTrace)))
     (:metric minimize(total-time))
)
```

Although the predictions and updates were being handled for external evaluation, in correspondence to how an EKF behaves, the declaration is relaxed because predict is handled as event which triggers at each **time_passing** event while update is an action called only after reaching a waypoint. This is only partly correct localization as update is not as frequent as predict, and hence renders the plan to be comparatively relaxed than an actual EKF implementation. To improve upon the precision, both predict and update should be changed to events handled explicitly by the external advisor. For details regarding the algorithm refer to 3.3. The change made in domain file compared to its predecessor is one event **belief_update** which does automatic predict and update of the covariance. A snippet of the change made can be seen in 3.1.1.3.

3.1.1.3 Predict and Update Event Domain Declaration

```
(:action goto_waypoint
  :parameters (?r - robot ?from ?to - waypoint)
  :precondition (and (robot_at ?r ?from) (observe)
  (not(robot_at ?r ?to)) (not (visited ?to )))
  :effect (and (not (robot_at ?r ?from))
            (assign (relativeD) (distance ?from ?to))
            (moving ?r ?to)
            (not (observe))
            (assign (counter) 0)
            (assign (update_covariance) 0)
            )
)



(:event belief_update
  ;;:parameters(?r - robot ?to - waypoint)
  :parameters()
    :precondition(and (> (counter) 0) )
    :effect (and
              (assign (cov) (update_covariance))
              (assign (counter) 0)
        )
)
```

To make the planner more robust, we introduce the concept of **connected** waypoints based on the sampling done by RRT algorithm 3.13. The idea is to connect waypoints such that only those within certain radius are considered for connection and the path with least cost to robot's initial base is connected. This expands the waypoint map in a tree fashion covering more surface area while maintaining complexity of least cost trajectory finding. As opposed to PRM approach in which all waypoints within a connecting distance and clear line of sight are considered as connected, RRT only connects waypoints which creates least cost path, hence reducing computation complexity. Since ROSPlan automatically generates Problem file based on pre-defined ROSPlan services, declaring certain variables in domain file become essential. For this reason, we also introduce **covariance** as a type and use **(lessthan ?c ?f - covariance)** as predicate which allows us to define the goal covariance and final trace. Compared to previous problem, idea for **endpoint** as goal waypoint has been introduced which is the final end point which can be pre-defined or randomly generated by the RRT algorithm 3.3.2.

3.1.1.4 Domain Declaration with RRT

```

(define (domain landmark)

(:requirements :typing :durative-actions :fluents :time :strips :
   ↪ disjunctive-preconditions :durative-actions
:negative-preconditions :timed-initial-literals)

(:types
  waypoint
  robot
  covariance
)

(:predicates
       (robot_at ?r - robot ?wp - waypoint)
       (visited ?wp - waypoint)
       (observe)
       (moving ?r - robot ?to - waypoint)
       (connected ?from ?to - waypoint)
       (lessthan ?c ?f - covariance)
)

(:functions (distance ?wp1 ?wp2 - waypoint) (cov) (counter)
(update_covariance) (predict_covariance) (relativeD) (dFactor)
       (finalTrace)
)


(:action goto_waypoint
  :parameters (?r - robot ?from ?to - waypoint)
  :precondition (and (robot_at ?r ?from) (observe)
  (not(robot_at ?r ?to)) (not (visited ?to ))
  (connected ?from ?to))
  :effect (and (not (robot_at ?r ?from))
           (assign (relativeD) (distance ?from ?to))
           (moving ?r ?to)
           (not (observe))
           (assign (counter) 0)
           (increase (update_covariance) 0)
           (increase (predict_covariance) 0)
           )
)

(:action reached
    :parameters(?r - robot ?to - waypoint)
    :precondition(and (moving ?r ?to) (<= (relativeD) 0))
    :effect(and (robot_at ?r ?to) (visited ?to)
    (not (moving ?r ?to)) (observe))
)


(:event belief_update
  ;;:parameters(?r - robot ?to - waypoint)
  :parameters()
    :precondition(and (> (counter) 0) )
    :effect (and
              (assign (cov) (update_covariance))
              (assign (counter) 0)
        )
)

;; to calculate the number of prediction steps needed
(:process odometry
  :parameters()
  :precondition (and (> (relativeD) (-dFactor)) )
  :effect (and (decrease (relativeD) (* #t (dFactor)))
  (increase (counter) (* #t 1))
)
)
)
```

3.1.1.5 Automatic Problem Declaration from ROSPlan with RRT

```
(define (problem landmark_task)
(:domain landmark)
(:objects
    dummy1 dummy2 - covariance
    kenny - robot
    endpoint wp0 wp1 wp2 wp3... wp29 - waypoint
)
(:init
    (connected endpoint wp0)...
    (connected endpoint wp10)
    (connected wp0 wp1)...
    (connected wp0 endpoint)
    (connected wp1 wp0)...
    (connected wp1 wp18)
    (connected wp10 wp8)
    (connected wp10 endpoint)
    (connected wp11 wp0)...
    .......
    (connected wp9 wp23)
    (observe)
    (robot_at kenny wp0)
    (= (cov) 1.22)
    (= (dFactor) 0.5)
    (= (distance endpoint wp0) 2.56223)....
    (= (distance endpoint wp8) 3.45579)
    (= (distance wp0 wp1) 4.28077).....
    (= (distance wp0 endpoint) 2.56223)
    (= (distance wp1 wp0) 4.28077)....
    (= (distance wp1 wp18) 2.90259)
    (= (distance wp10 wp8) 2.77308)....
        ......
    (= (distance wp9 wp23) 2.88531)
    (= (finalTrace) 0.5)
    (= (predict_covariance) 0)
    (= (relativeD) 0)
    (= (update_covariance) 1.22)
)
(:goal (and
    (robot_at kenny endpoint)
    (< (cov) finalTrace)
))
(:metric minimize(total-time)))
```

3.2 Software Architecture

For our TAMP approach we use the PDDL+ based DiNo planner to describe our domain. Hence, it is essential to first understand the DiNo planner framework to understand our proposed modifications. Amalgamating different softwares, our framework uses DiNo with External Advisor. Since DiNo itself is an upgrade of UPMurphi, in section 3.2.1, we give an in-depth over-view on the software architecture of UPMurphi. In addition, in section 3.2.2 we explain how DiNo is merged with our External Advisor for precise state prediction and plan generation. Finally, we give an overview of our External Advisor in 3.2.3. It is to be noted that we use the term External Advisor to refer to the specialized heuristic aiding semantic attachments that is used in this thesis.

3.2.1 Planner Framework

The need of discretization is aptly emphasized based on our domain and problem structure. This discretization also defines the predict and update phase, which is critical part of our plan formulation. Hence, the most essential portion of the chosen planner is it's handling of the discretization. Furthermore, successful heuristic calculation is also essential for devising an admissible plan. Below we discuss the key features of DiNo in these regards.

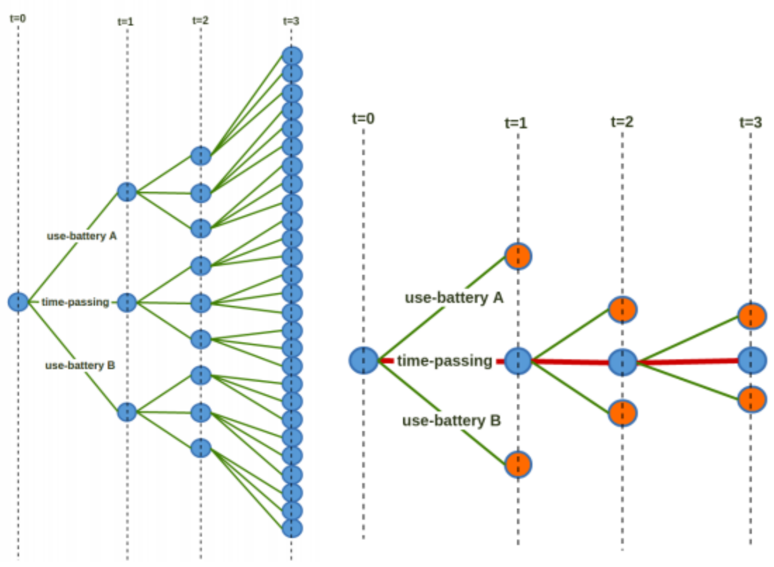
The time-passing action plays an important role as it propagates the search in the discretized time-line. During the normal expansion of the Staged Relaxed Plan Graph (SRPG), the time-passing is one of the δ-actions and is applied at each fact layer. Time-passing can be recognized as helpful action [11] when its effects achieve some goal conditions (or intermediate goal facts). However, if at a time t, no helpful actions are available to the planner, time-passing is assigned highest priority and used as helpful action. This allows the search to quickly manage states at time t where no Happenings of interest are likely to occur.

Innovating from the UPMurphi which expands at each time step, even during idle periods, DiNo's SRPG+ identifies time-passing as a helpful action during idle periods and thus advances time, mitigating the state explosions. An example of the state expansion between the two planners is in figure 3.1. The domain explored here is Solar Rover domain [33]. The continuous effects and processes are handled such that at each action layer, the numeric variables' upper and lower bound are updated based on the time-step functions used in the discretization to approximate the continuous dynamics of the domain. Events are then checked immediately after the process and their effects are relaxed as for the instantaneous action.

The DiNo algorithm has been divided into following sections

1. Explore_bfs()
   Uses SRPG for calculating the heuristics based on the rules, makes the list of the state and transition based on the rules (in present case, the approximate heuristics)

2. Build_Dynamics()
   Essentially a graph manager. Uses the list to do model checking. Based on the list of states and transition, there are two type of memory options



(a) UPMurphi (b) DiNo

Figure 3.1: Branching of search trees in (Blues are explored, oranges are visited. Red edges correspond to helpful actions)

(MemGraphManager, DiskGraphManager). In the graph form, states are nodes and edges are transition which contains [from, to , weight and duration]

3. Find_Paths()
   Based on metric and distance formula for each edge, creates a list of chosen edges

4. Collect_Plans()
   There is a subroutine here (Build_Plan_From) which finds a list of controllable plans using the chosen edges. Out of all controllable plans, one with best values is chosen.

5. Output_Results()
   Prints the result in PDDL+ format

6. Validate_Results()
   Uses Val file for validating

3.2.2 Planner Framework with Semantic Attachment

Since the current DiNo model only handles fixed limited changes and is not able to do complex and precise computation based on the PDDL+ algorithms, we introduce semantic attachments in DiNo for higher level approximations with better error and precision handling. Specifically, we develop an external library capable of handling modules such as extended kalman filter along with the domain stated which enables a more realistic path planning for the SRPG+ framework implemented in DiNo. The improvement here is shown by getting realistic and approximate position of the robot with relation with surroundings and every discretized time-step which helps SRPG+ framework to create new plans which will be followed in real scenarios.

DiNo assumes static changes in the robot's motion variables which does not compensate for errors and misalignments that has already occurred during the navigation. We implement EKF based variable updating which considers all previous motions along with all the errors that have occurred in them and gives a mean and covariance of the robot position within certain standard deviation. This covariance is then passed to DiNo which will account in the heuristic calculation of SRPG+.

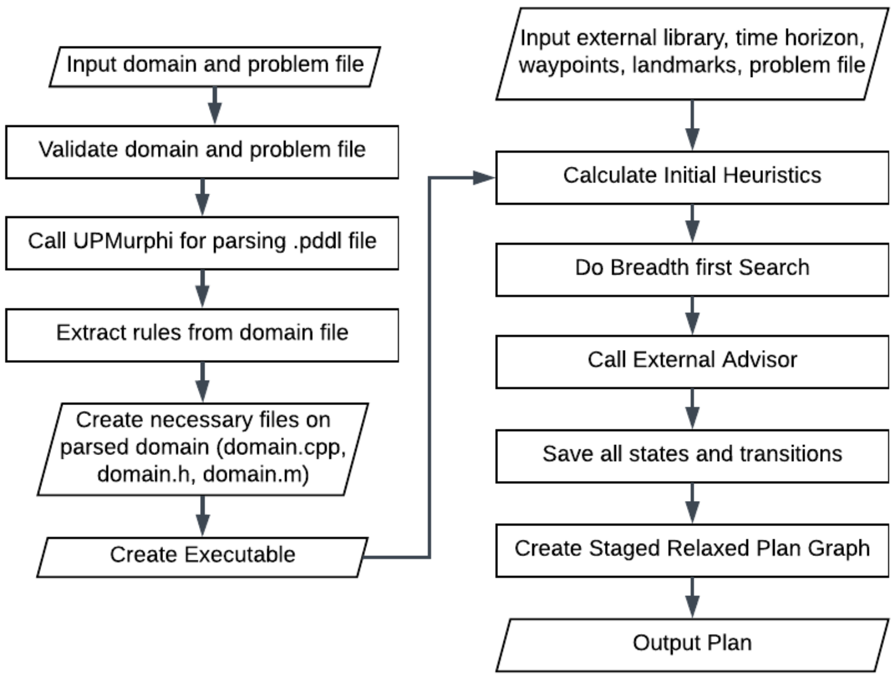

Input domain and problem file
Validate domain and problem file
Call UPMurphi for parsing .pddl file
Extract rules from domain file
Create necessary files on parsed domain (domain.cpp, domain.h, domain.m)
Create Executable
Input external library, time horizon, waypoints, landmarks, problem file
Calculate Initial Heuristics
Do Breadth first Search
Call External Advisor
Save all states and transitions
Create Staged Relaxed Plan Graph
Output Plan

Figure 3.2: Working of DiNo framework

As show in figure 3.2, DiNo is integrated with the external parser to create valid SRPG. Initially, an executable is generated which is based on given domain and problem file.

The Breadth First Module, *Explore_Bfs()* is designed to check every possible state from a given state with a set of satisfying rules. For doing so, it creates a stack of all possible states sorted by heuristic value calculated for that state. It calculates next state for the state with lowest heuristic value *h_val* and then adds it to the stack. This process does not rely on following through same state until it is completed, rather it saves and resumes search based on the layer of search as breadth first search does. An essential part of the module has thus been to save the states in each layer of the breadth first graph. State with unreachable condition are given significantly high heuristic value and are not included in the stack.

Since the standard DiNo planner does not handle variable change through external call, the planner has been modified accordingly. First the system file, *ump_system.cpp/h* A.1.1 which is responsible for handling all submodules like *Explore_Bfs()* and SRPG is modified to load and call external solver if the external flag is given during runtime. This load and call is dynamic so it can work with any external solver library passed by user during runtime and does not have to be recompiled every time. This is to provide robustness to the system. Then the current working state is passed to external solver which does the specified heavy calculation and returns updated variables. To reflect these updates back to DiNo's working state, extra getter and setter methods are implemented in UPMurphi files *upm_util A.1.3, ump_state A.1.1, cppcode.cpp A.1.5*. Argument handling and integration is modified in *upm_io A.1.4, ump_epilog A.1.6*. Furthermore, as variables for each domain file would be different, such extra variables which should also be handled from external solver can be accessed and changed directly from current state object in main system file *upm_system* A.1.1. After that, DiNo uses the updated variables to calculate the next state of SRPG instead of pre-defined variables. This changes the way each graph is built and inadvertently changes the final plan while enabling higher precision calculations.

3.2.3 External Library

The function of the external library is to perform the computations for the non-linear model and provide precise state estimate values which cannot be computed by the standard PDDL based models. As computations are done based on the robot's current state and all the effecting previous states, the external module has to incorporate the current state utilized by DiNo as well as all the related previous states of the expanded branch. First, the state initialization is performed with the same parameters in the domain and problem description. The initialization parameter file also includes waypoint and landmark locations. Each action in the domain is mapped to certain rules. These rules rely on fixed predicates, which are then verified in the external library to make changes based on the actions fired. In order to synchronize the state graph with DiNo which relies on a breadth first search based on the heuristic value, the external library also saves corresponding states which DiNo stores in its stack. When DiNo explores a state and loads its values, the corresponding values used for calculation in external library is then recalled from the external stack and values for next state is calculated. More specifically, when a **goto_waypoint** action is passed, the external solver sets robots known position as origin variables and initiates the EKF matrices. During a **time passing** action between waypoints, EKF prediction and update is done for robots new position and an estimated update covariance value is returned. The EKF update only happens when robot is within observable distance and direction of landmarks. Doing so will realistically update the Kalman gain for robot and reduce error based on observable mapping of robot in world map. The returned update covariance is set as covariance of state and used for heuristic calculation in DiNo. We also notice some floating point precision error during calculation which was affecting covariance calculations, and have implemented checks to handle such cases.

Every successive movement induces change in covariance. Though out the trajectory, the predict and update covariance affects the state and position of the robot. Hence, when the robot is close to the target waypoint, an allowable covariance error threshold must be determined. We use the covariance of position as the allowable range. This is essential to avoid negative prediction. The implementation can be observed in the git provided A.2.

3.3 Integration Architecture

Since ROS is a widely used and well developed framework that handles easy integration of multiple sub-modules, we use the ROSPlan framework for handling the communication to integrate all the different hierarchies of planning module. For a complete planning, execution and feedback, ROSPlan architecture has been shown in 3.4.

Since this thesis is more tilted towards the planning paradigm, we shall not be discussing in-depth on the sensors, sensor parsing and the execution. In regard to our framework integration architecture, figure 3.5 would be discussed further. The domain and the problem files are stored in the knowledge base of the ROSPlan architecture. Using ROS services, problem files can be generated. Knowledge base server is used for storing values from domain file, problem file and map file. Since the prerequisites for path-planning are the waypoints, both the options of

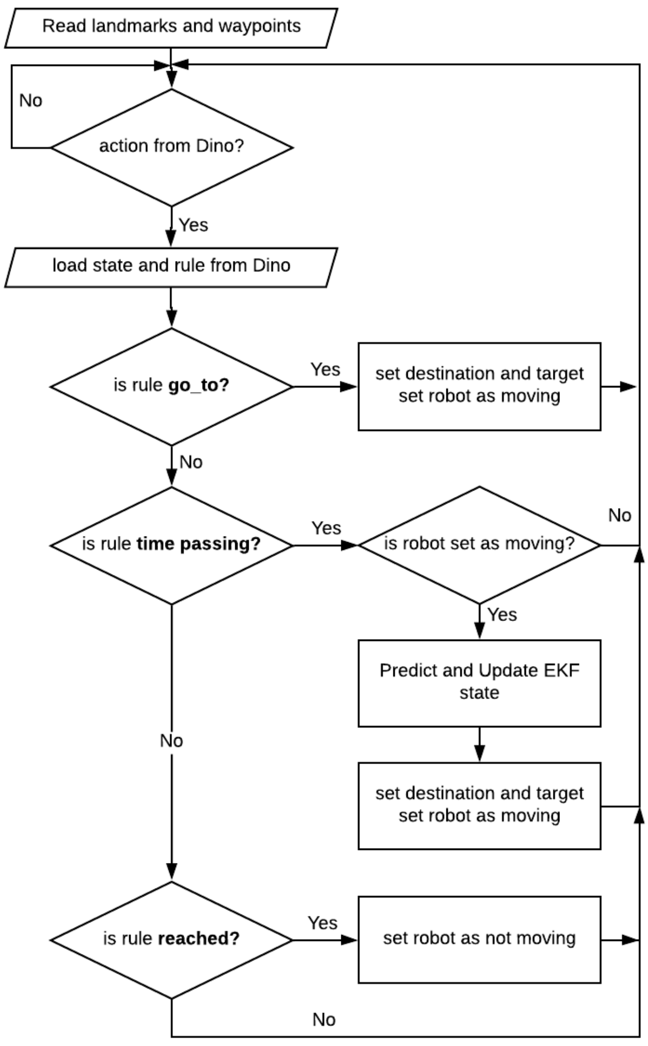

Read landmarks and waypoints
No
action from Dino?
Yes
load state and rule from Dino
is rule **go_to?**
Yes
set destination and target set robot as moving
No
is rule **time passing?**
Yes
is robot set as moving?
No
Yes
Predict and Update EKF state
set destination and target set robot as moving
No
is rule **reached?**
Yes
set robot as not moving
No

Figure 3.3: External Call Flowchart

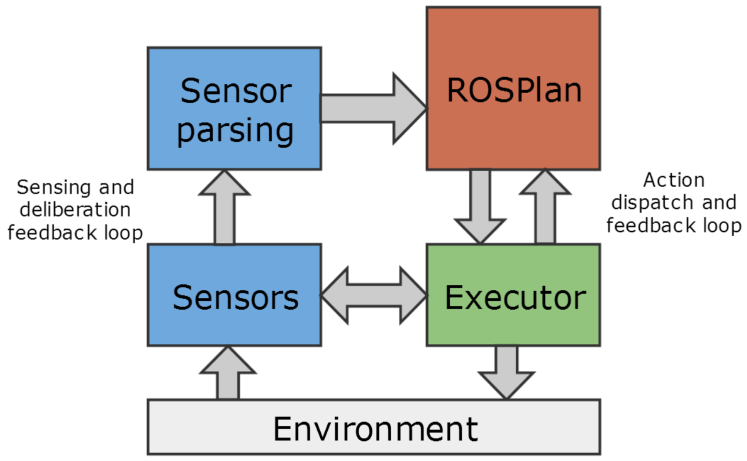

Sensor parsing
ROSPlan
Sensing and deliberation feedback loop
Action dispatch and feedback loop
Sensors
Executor
Environment

Figure 3.4: ROSplan Framework

using either pre-defined waypoints or generating new ones should be available for robustness and replication.

ROSPlan Framework allows the user to build node for communicating with different planners. In our case the PDDL+ problem instance and the domain is used for executing the planner. After the planner has completed generated a meaningful plan, the plan parser service and the plan dispatcher service is used which then interrogates the knowledge base to validate the plan and finally the actions are applied on the platform. Based on the platform, the action is applied and feedback is sent to the knowledge base. In case of plan failure, generation of new problem file based on new knowledge of the environment is done. Otherwise, next action from the stack is applied.

Detailing further on the used nodes, services and applied changes, the integration architecture can be submoduled as following:

3.3.1 Map Generation

Although no new modules have been added to the map generation, an essential portion of the thesis is the communication between the nodes in the navigation stack in ROS. As shown in figure 3.6, map_server and sensor sources communicate with each other for recovery behaviors as well as in concurrence to the global planner in the system. The odometry data, sensor transforms as well as localization module(AMCL) is necessary for the map generation and continuous localization during execution phase.

SLAM (Simultaneous Localization and Mapping) is executed with ROS for building a map for the environment with the robot. Localization takes place

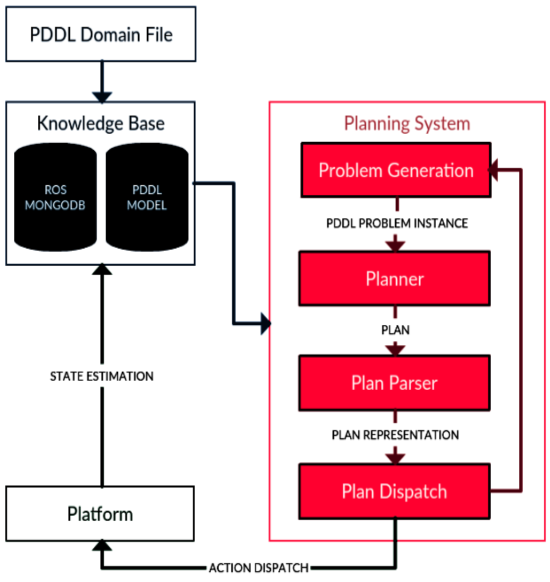


Figure 3.5: ROSPlan Workflow

through the Adaptive Monte Carlo Localization (AMCL) using the data points obtained through SLAM. AMCL is a probabilistic localization system for moving robot in 2D space using particle filter to track the pose of the robot against a known map. To illustrate, the model built in Gazebo 3.7, can be mapped into a portable gray map (pgm) which is used later in the navigation stack. An Occupancy Grid Mapping(OGM) is built from SLAM through sensor and odometry readings. In correspondence, a YAML file is generated with information regarding the mapping.

Next module 3.3.2 is highly influenced by the map generating module since the mapping transformation from robot and environment is vital to successful distributed waypoint generation as well as replication.

3.3.2 Waypoint Generation

We utilize task planning to synthesize a plan by performing search in the motion space and hence require sufficient amount of waypoint distribution throughout the map. Such a distribution is easily obtained by using the standard PRM or RRT based sampling approaches. In our case, a caveat which concerns the task planner is the number of waypoints. A large number of waypoints leads to state space explosion and thereby increased planning time. This calls for a waypoint generation strategy which on one hand should sample sufficient number of way-

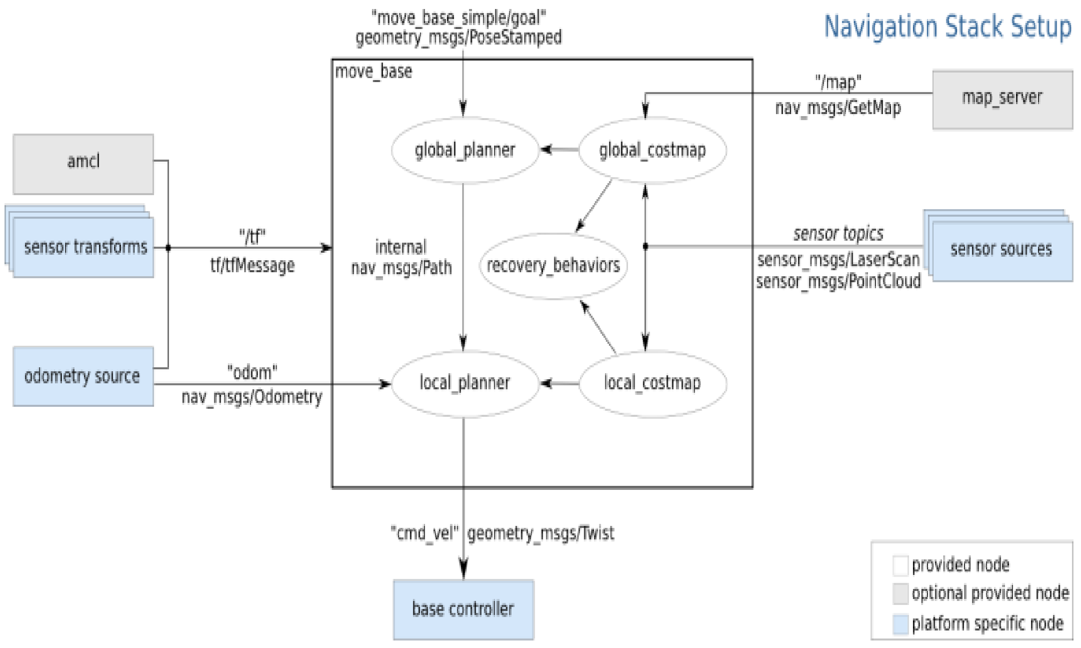
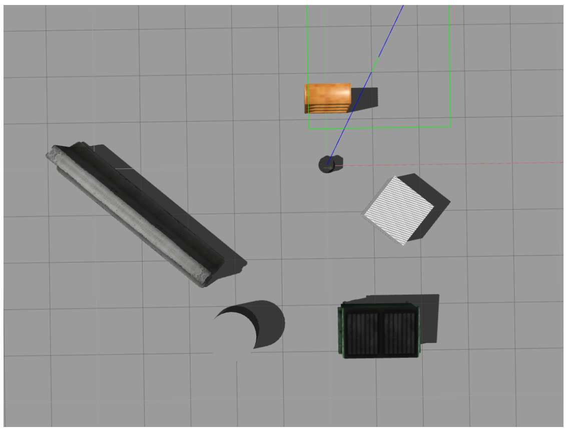


Figure 3.6: Navigation stack



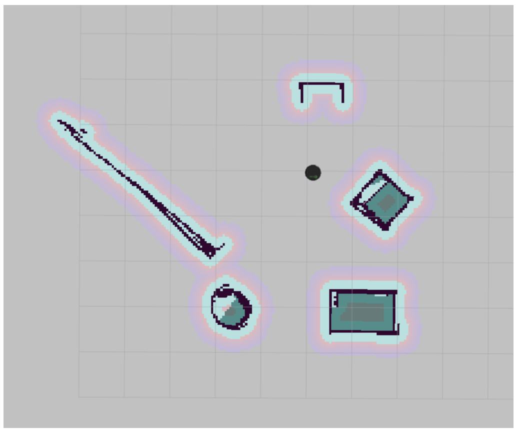


Figure 3.7: Playground world gazebo map

Figure 3.8: Playground world rviz map

points to synthesize satisfying plans, while on the other hand keep this number minimal. Standard RRT based approaches sample waypoints that connects the start and end locations. However, to reduce pose uncertainty it is to be ensured that such connected paths have ample number of waypoints from which landmarks can be observed. To facilitate this perception aware search we implement an RRT based potential field approach to sample such relevant waypoints in the environment.

3.3.2.1 RRT over PRM

First we generated PRM maps and observed the distribution as seen in figure 3.9. We attempted to create 30 waypoints which was only partially successful, only 15 waypoints are created because of PRM's limitation on generation algorithm (number of attempts, number of waypoints, casting distance limitation). Also it connects each waypoint with another within a certain distance, which creates more connection than necessary and increases planner complexity. The generated waypoints are not in closer proximity to the landmarks for successful observations. Compared to PRM approach, when we deployed our RRT based motion planner, it was successfully able to distribute all 30 waypoints throughout the map as seen in figure 3.10. We can observe that the waypoints are closer to the landmarks (denoted by red dots) due to the potential field algorithm in concurrence with the RRT planner. The waypoints are not necessarily connected with each neighbor but instead follow a least cost path to the root node which keeps complexity low for the planner while increasing number of waypoints.

Furthermore, when an endpoint is either randomly generated or selected by the user, the previously generated waypoints will be connected with the endpoint based on proximity as shown in figure 3.11. The endpoint is marked with blue dot. We tested the robustness of the algorithm in different environments. Successful RRT generation and endpoint connection can be seen in the Corridor environment in figure 3.12.

3.3.2.2 RRT based Potential Field Algorithm

ROSPlan has two nodes for planning server, SimpleRoadmapServer and RPRoadmapServer, which is an essential part of our architecture. SimpleRoadmapServer is used for loading pre-generated waypoints file into the knowledge base. Depending on the distance between the waypoints and their visibility in the map, they are connected as neighbors. It also allows user to add waypoints manually as a ROS service. On the other hand, RPRoadmapServer by default creates a PRM based waypoint using the connected distance, threshold and casting distance.

For our particular problem statement, an RRT based potential field approach has been implemented. As input, the function takes landmark file and occupancy map information 3.3. In general RRT generates a random point in the grid, finds the closest available node and does an epsilon expansion towards the node provided that the connection is visible. For our alterations, firstly, the waypoints can only follow least cost path to the root node and each waypoint should be between minimum and connecting distance to a neighbor. The need of potential field is to have waypoints close to each landmark while satisfying first rule of least cost path. This insures a higher probability that all landmarks are visited

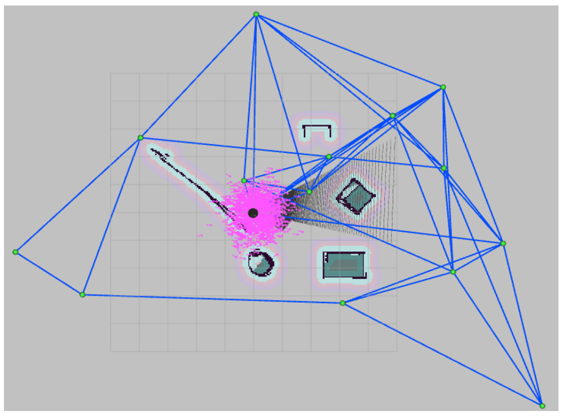


Figure 3.9: PRM with 30 waypoints

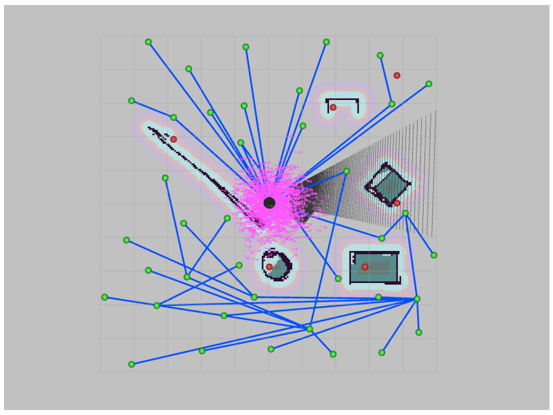


Figure 3.10: RRT with 30 waypoints

by at least one root that connects to the end goal. For the potential field to work, there are two zones that respectively pulls and nullifies the semaphore for next waypoint close to the landmark. Once a waypoint falls into the attractive zone of potential field, the successive waypoint is pulled closer into the second zone. After that the landmark's potential field is set as negative and other waypoints will not be generated within that field. Finally the end-point is randomly generated or generated at a fixed point as per user selection 3.13. All the generated waypoints and landmarks are then pushed to the knowledge base as well as the visualization node. The connected edges and their distances are extracted and pushed to the knowledge base.

Contrary to existing SimpleRoadMapServer, we built another module to load the already built RRT waypoints in RPRoadmapServer. This **LoadRRT** module loads previously generated waypoints and passes it to the knowledge base instead of creating new RRT waypoints. It inputs waypoint file and does waypoint transformations using the map parameters loaded. Hence OGM and map_server play

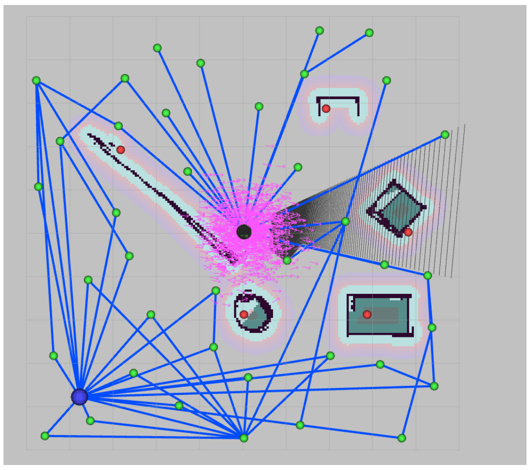


Figure 3.11: Endpoint connection with generated RRT

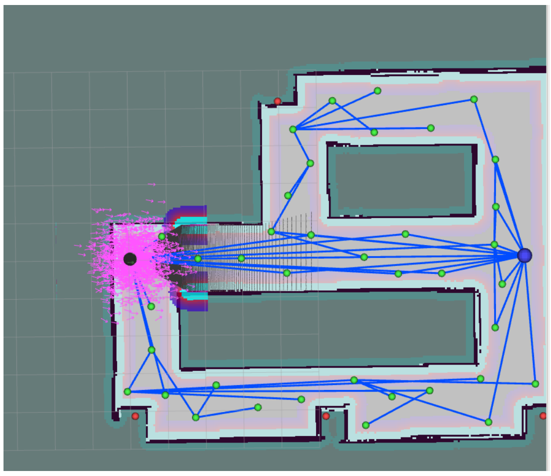


Figure 3.12: Endpoint connection with generated RRT in Corridor environment with 40 waypoints

important role in loading a generated RRT file. The RRT algorithm implementation is found at A.3.6.

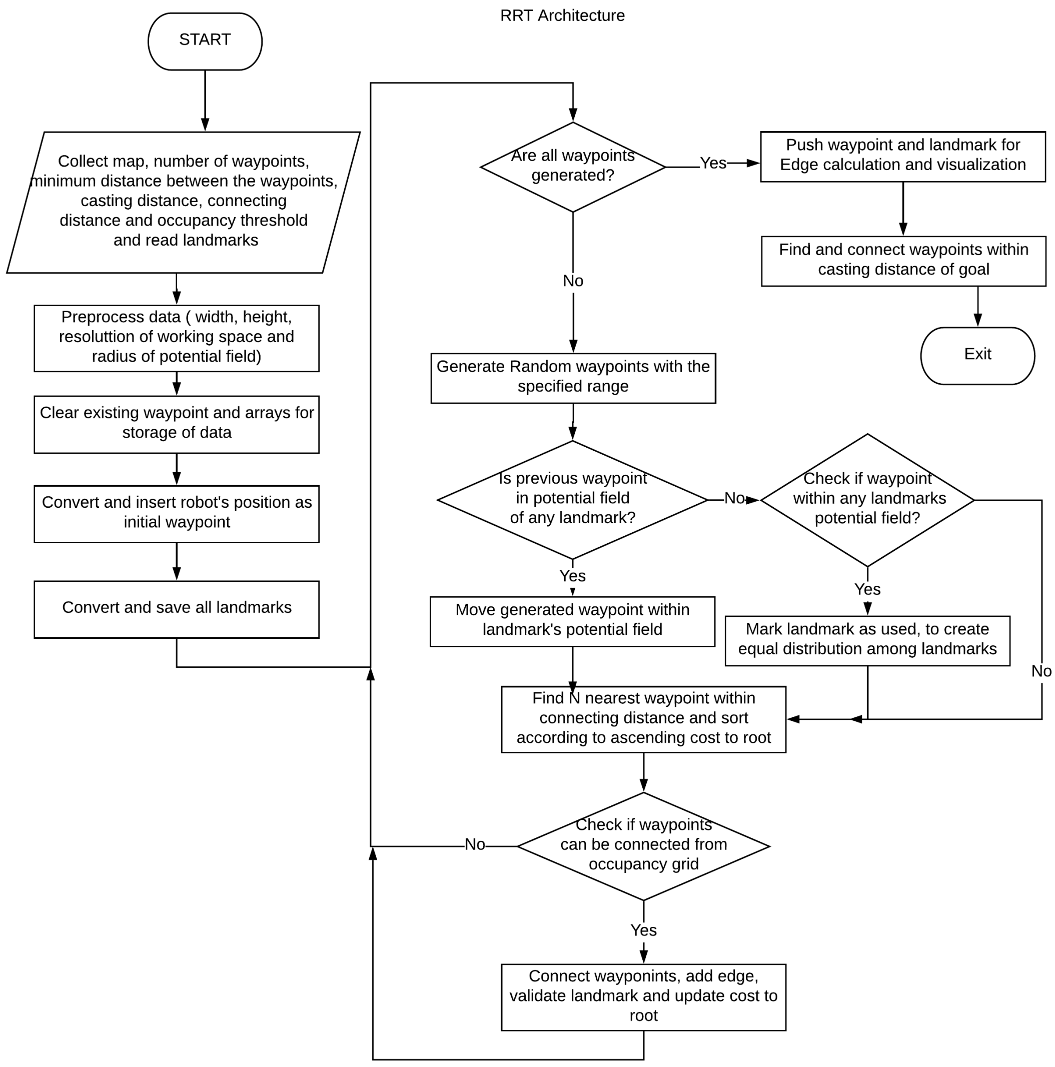

RRT Architecture
START
Collect map, number of waypoints, minimum distance between the waypoints, casting distance, connecting distance and occupancy threshold and read landmarks
Preprocess data (width, height, resoluttion of working space and radius of potential field)
Clear existing waypoint and arrays for storage of data
Convert and insert robot's position as initial waypoint
Convert and save all landmarks
Are all waypoints generated?
Yes
Push waypoint and landmark for Edge calculation and visualization
Find and connect waypoints within casting distance of goal
Exit
No
Generate Random waypoints with the specified range
Is previous waypoint in potential field of any landmark?
No
Check if waypoint within any landmarks potential field?
Yes
Move generated waypoint within landmark's potential field
Yes
Mark landmark as used, to create equal distribution among landmarks
No
Find N nearest waypoint within connecting distance and sort according to ascending cost to root
Check if waypoints can be connected from occupancy grid
No
Yes
Connect wayponints, add edge, validate landmark and update cost to root

Figure 3.13: RRT implemented

3.3.3 Problem Generator

Once the domain is parsed and knowledge base has been updated with waypoint and landmark generation from previous step, the problem generator node is called upon. Note that we can choose to not create problem.pddl file through this node and just use user defined problem file. The service first reads any user input for the problem parameter though the knowledge base such as goal, initial values, literals. Then the connected waypoints and their distance is loaded from knowledge base and put in the problem file in pddl format. Since the original problem generator was not designed to handle goal with complex clauses as well as metric, we develop our own module to handle these cases and direct the readers to Appendix A.3.1 for the link to git.

3.3.4 Plan and Reference Trajectory Generation

Once problem.pddl file is generated the call to the planner framework, in our case DiNo, is made. DiNo takes the domain and problem, waypoints, landmarks, external advisor and other specialized planning parameters as input. It internally handles the plan generation and outputs a plan.pddl file which contains the waypoints robot has to follow. Furthermore it also generates a reference trajectory which is generated by the external advisor during planning section.

3.3.5 DiNo to Esterel Plan Generator

After plan.pddl is generated, plan parser node is called which takes plans in pddl format as input and parses to extract the waypoint which is updated in the knowledge base. Note that the pose of the waypoint is not specified in any .pddl file but it is taken implicitly from the knowledge base based on its name. Then a plan is saved in Esterel format which can be later used for execution as an option. The parser implementation is available at A.3.4.

3.3.6 Default Plan Execution and Trajectory generation

The global navigation is used to create paths for a goal in the map or at a far-off distance. The local navigation is used to create paths in the nearby distances and avoid obstacles, for example, a square window of 4x4 meters around the robot. Once the plan is updated in knowledge base, the planning server side is called. This takes an action from the knowledge base stack and sends it to the */move_base* node. The */move_base* node handles the robot movement and provides feedback to ROSPlan on whether the action succeeded or failed. If the action is successful, then next action from the stack is passed until the end goal is reached. Simultaneously, the node provides the position and velocity of robot during the action which is saved as actual trajectory. The implemented module can be accessed from A.3.2, A.3.3, A.3.5.

Chapter 4

Experiments and Results

To validate our DiNo module with EKF and the upgrades made with external advisor, we implement a toy example, populating an open environment with a set of waypoints wp and a set of landmarks l. This domain helps to fathom the underlying concepts of our TAMP approach and is described in Section 4.1. We also evaluate our approach in simple yet realistic experiments in the Gazebo simulator. Two different environments are considered, (1) the default playground world environment of ROSPlan with slight modification on the occupancy map resolution and (2) a corridor environment as seen in Figure 4.9.

4.1 Abstract Example

For the simulations, we use five waypoints, three landmarks and a goal waypoint. DiNo provides a verbose mode where the numerical values of the variables can be closely observed. For the two different domain declarations, we observe the resulting changes in covariance and discuss the different test cases below.

4.1.1 *action* and *event* for Belief Updates

Corresponding to the domain description in 3.1.1.1, the action **belief_update** is performed only when robot has reached close proximity to the landmark. The environment is shown in figure 4.1. As observed, from 4.2, we see the covariance of the robot decreases only when the landmark is visited. The plan for this action is to visit wp2 from wp0 and then wp5 from wp2. Hence, we see two sharp decrease in the covariance of the system when these waypoints (landmarks are close to waypoints) are reached. Furthermore, this approach fails when multiple landmarks are within range as the external library for this domain supports only single measurement updates. The landmark and waypoint locations are detailed

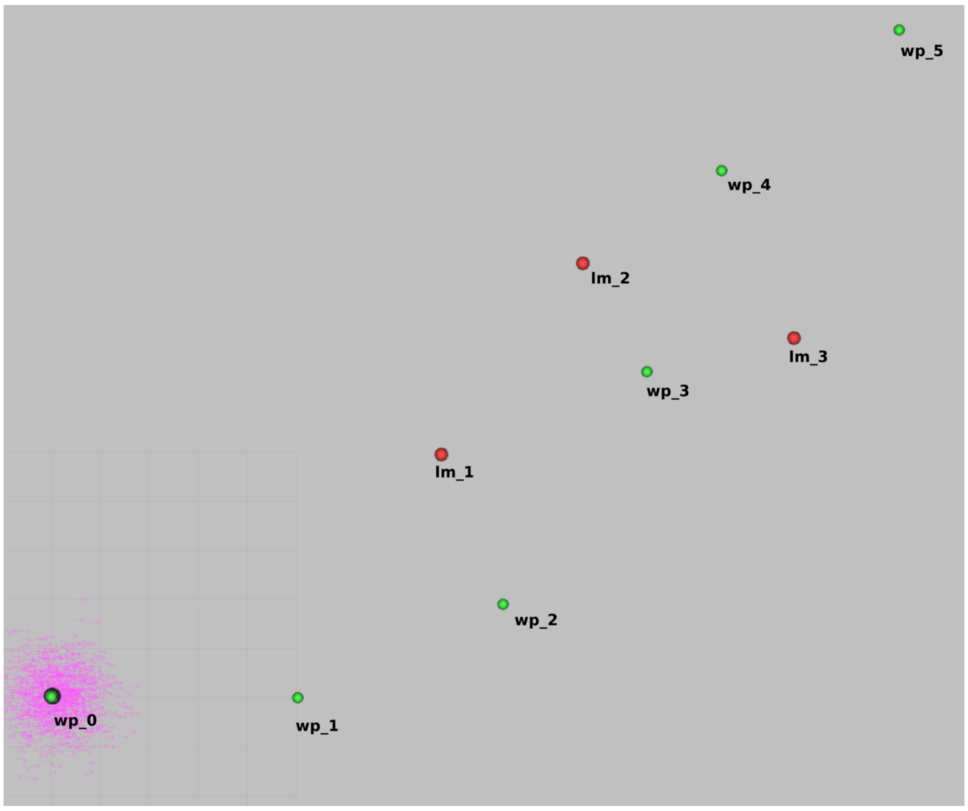


Figure 4.1: Initial test map

in A.8, A.7.

4.1.2 Single *event* Based Belief Updates

For a more realistic scenario, we upgrade the module to perform predict and then look for landmarks that lie within the sensor range. For each landmark within the range, the posterior belief is calculated by simulating the future observation for each corresponding landmarks. With reference to domain 3.1.1.3, we see that both predict and the posterior beliefs are evaluated using a single **belief_update** *event*. The algorithm used is summarized in 1. We closely observe the value of the covariance during the progression of the plan as shown in figure 4.3. Given the closeness of the waypoints and landmarks, the plan generated for the robot is to directly move from wp0 to wp5. The landmark and waypoint locations are given in Appendix A.8, A.7.

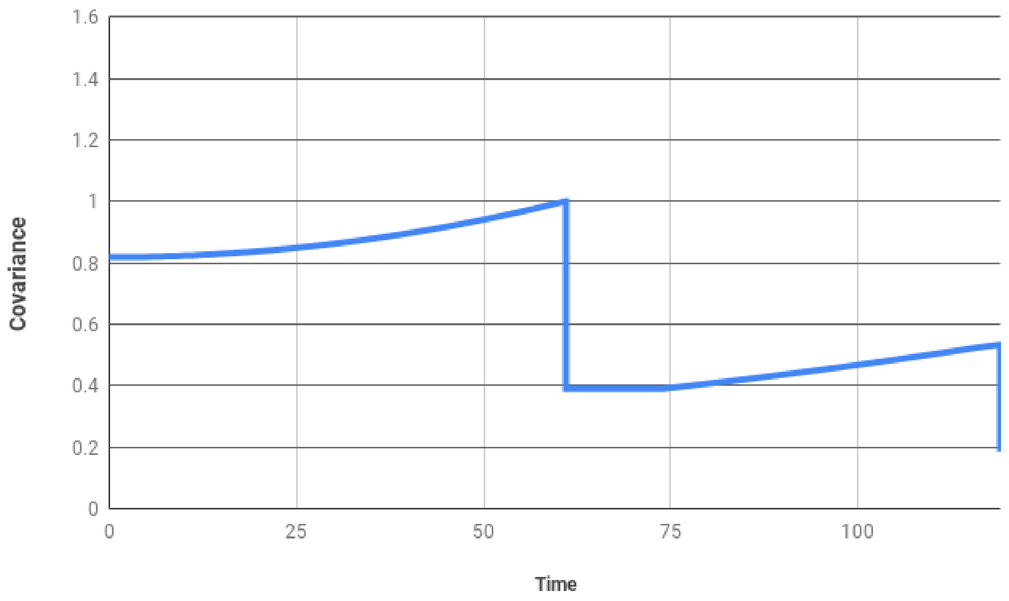


Figure 4.2: Covariance reduction with double update

4.1.3 Belief Update with RRT

The problem and domain description for this particular case has been discussed in A.4 and 3.1.1.4. The associated waypoints and landmarks for this test case are given in A.5 and A.6. We see that the covariance reduces realistically due to the separation of the landmarks being wide enough and the waypoints generated being closer to each other, which is a result of the RRT algorithm. The map associated with the generated covariance is shown in figure 3.11.

4.2 ROSPlan Simulation

In this section we discuss the synthetic simulations in Gazebo to further validate our TAMP algorithm. It is to be noted that for each given landmark we consider a potential field originating at its center. With the inclusion of the occupancy grid obtained during the mapping and the potential field, the RRT is constructed maintaining a closer proximity to the landmarks. The computational complexity is directly dependent on the number of waypoints m generated. The action **goto_waypoint** is triggered only if two waypoints are connected. Hence in the worst case, we have $m(m-1)$ possibilities for encoding (connected ?from ?to). Similarly we have m each for (robot_at ?r ?from), (robot_at ?r ?to), (visited

?to) and (moving ?r ?to). Ignoring other unary encodings, the total states to be explored is 2^{m^2+3m}. The heuristic based search of the DiNo planner, reduces this state space explosion significantly. Furthermore, due to our potential field based RRT sampling, we are able to prune unwanted state expansions by generate parsimoniously connected waypoints which are sufficient for synthesizing satisfying plans. The visualization and initialization changes can be observed at A.3.7,A.3.8, A.3.9, and A.3.10.

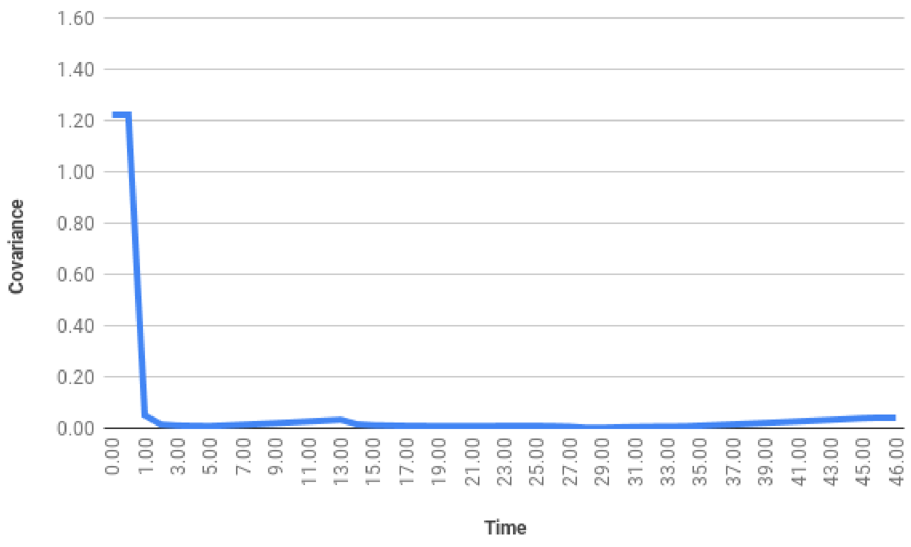


Figure 4.3: Covariance reduction with constant update

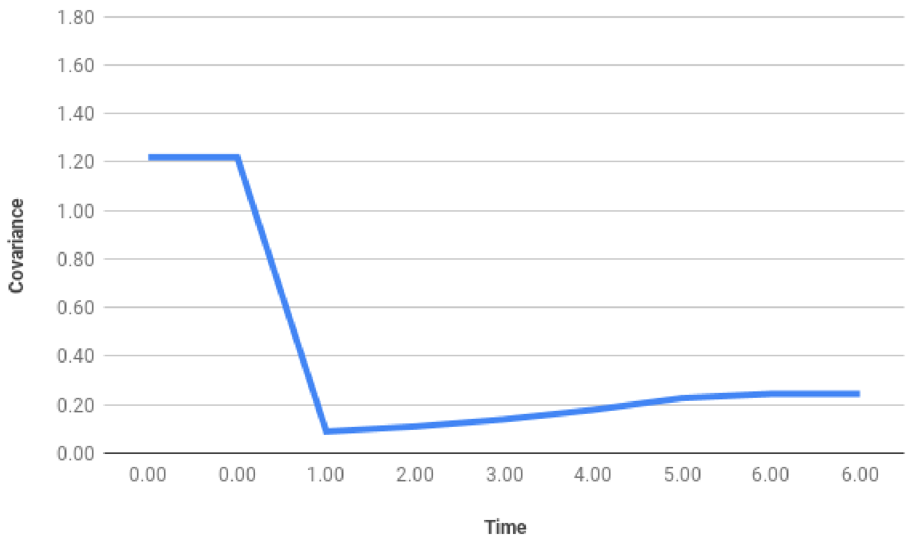


Figure 4.4: Covariance reduction with constant update and RRT

4.2.1 Playground Environment

Here we use the default playground world provided by the ROSPlan framework. The world consists of four different obstacles around the robot as shown in the figure 3.8. A process discretization $\Delta = 1$, distance factor of $dFactor = 0.5$ and a temporal horizon of $T = 12$ is used. Using a discretized model and a finite-time horizon ensures a finite number of states in the search for a solution. The desired goal covariance is also set to $finalTrace = 0.5$ in the problem file. In three of our test cases environment the goal is selected manually to reflect different conditions. In the first case, the goal is selected to be in direct line of sight from robot and in close proximity. In Case 2 and 3, the goal is hidden behind an obstacle and at a reasonable distance from the robot's origin. For last test case, we used an automatic end-point generator node for robust goal selection. This generates a random end-point within the map and connects it with valid waypoints in proximity.

Case 1: In this case the endpoint is in direct line of sight from the starting position (center). For a simple test case, we want the $finalTrace$ for this case to be less than 0.5. Given such simple case, the robot follows a straight path to the landmark as seen in Figure 4.6a. Furthermore, the final trace for the generated path is 0.311644. A total of 32 states were explored for the given state. The plan was computed in 0.86 seconds. Hence the generated planner and ROS trajectory follow a similar path as can be seen in 4.7a. We can observe similar progression in the position, the generated angle follows a different trajectory because of the velocity model used in ROS planner.

Case 2: Since the endpoint is not in line of sight, the path that planner chooses has multiple connected options to choose from. The optimal path selected is using one connection which is way point right above the center (bookcase) 4.6b. For the given case 37 states were explored in 2.38 seconds and with final covariance of 0.301713. The planner and ROS trajectory for the case is pictured in 4.7b. Having similar changes in the x and y position, the abrupt changes in the orientation is handled better in ROS. While the angular position in both Case 1 and this case, have abrupt change in theta, ROS follows a smoother velocity model.

Case 3: In this simulation, the waypoints are more spread apart than Case 2. So the optimal path consists of two connections 4.6c. We notice it has one connection path as well but they are further away from landmarks so the final covariance from those paths are higher than the path with two connections. Given the constraint, a total of 27406 states were explored in 146.52 seconds resulting in final covariance of 0.453272. The ROS and planner trajectory follow similar pattern with smoother features in ROS.

Case 4: Case 4 is an ideal test case where end point is generated in direct line of sight from the starting point. Contrary to Case 1, in this test case, the automatic end-point generation has been used. An interesting observation is the automatic generation of waypoint for RRT. Our algorithm uses potential field to bring new waypoints closer to landmark after the first random waypoint is generated within the field of the landmark. In this case, we can observe that the first waypoint generated is right next to the landmark, this may create problem of collision depending on the nature of the obstacle and occupancy it hold. In the given case 25 states were explored with 1.2 seconds of computational time and 0.244359 covariance. Similar to Case 1, the direct line of sight makes the planning faster.

We can see the change in covariance in the planner from Figure 4.5, in all the four test cases, the initial position of the robot and the chosen trajectory updates the covariance and reduces it for faster approach to goal. Given the plan length and trajectory length of Case 3, we can observe a higher final covariance.

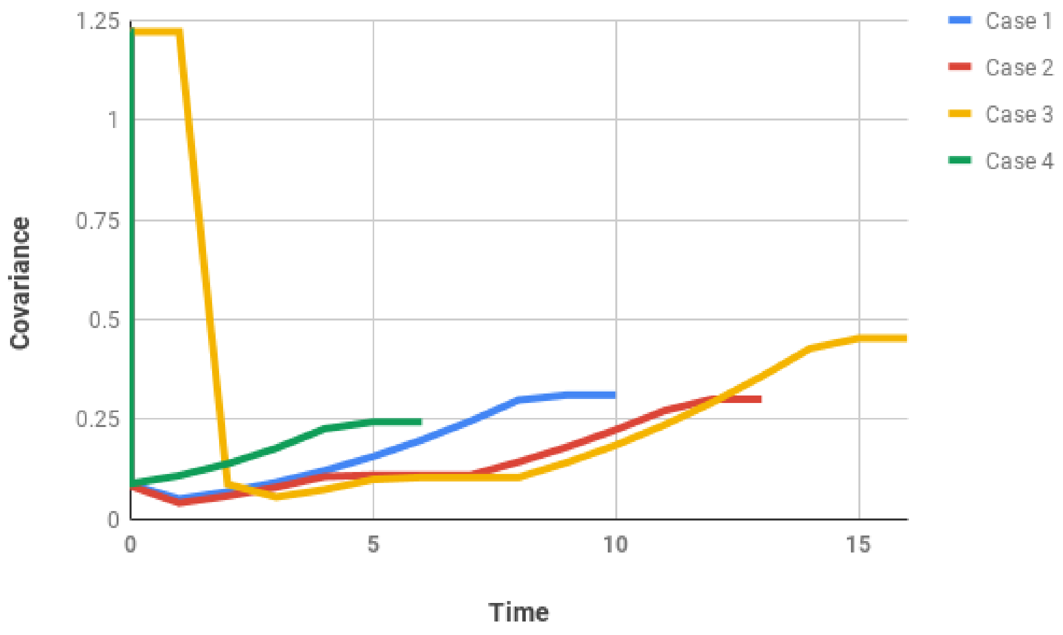
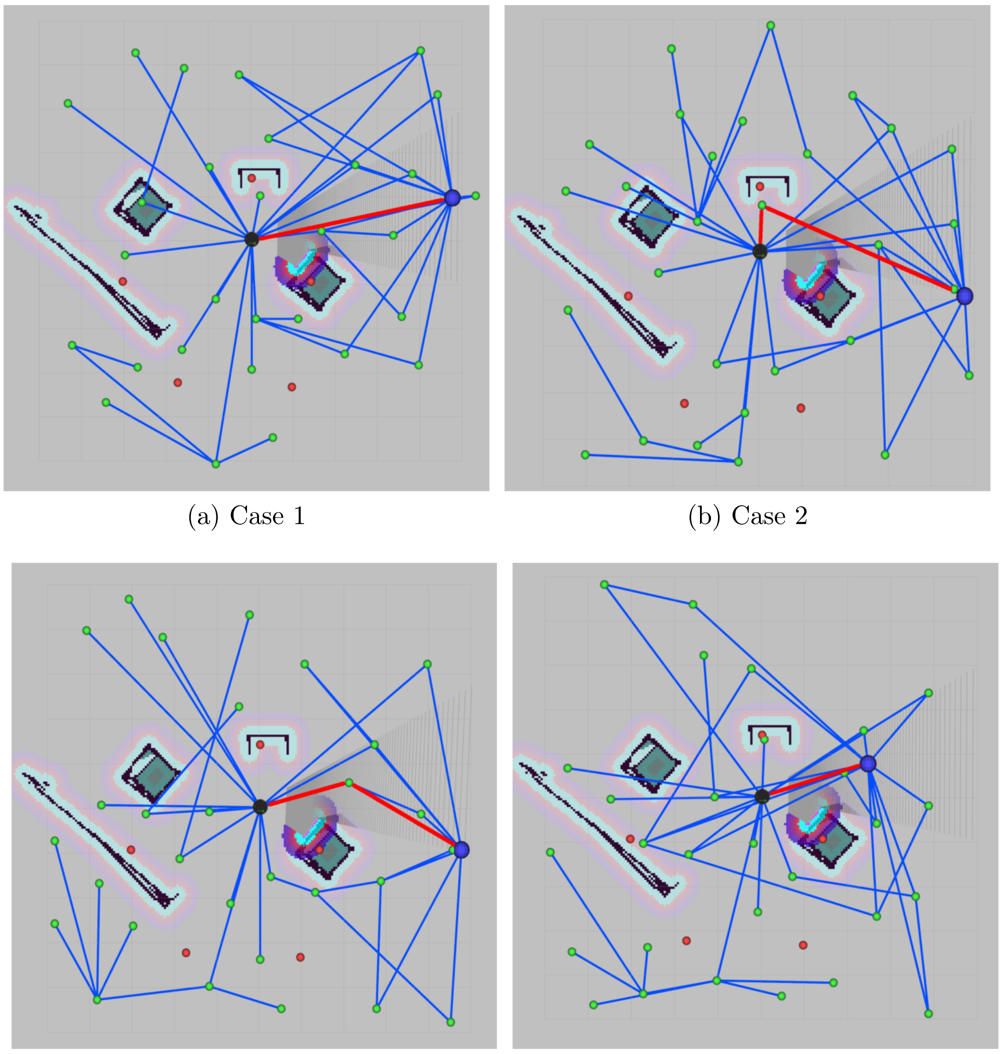


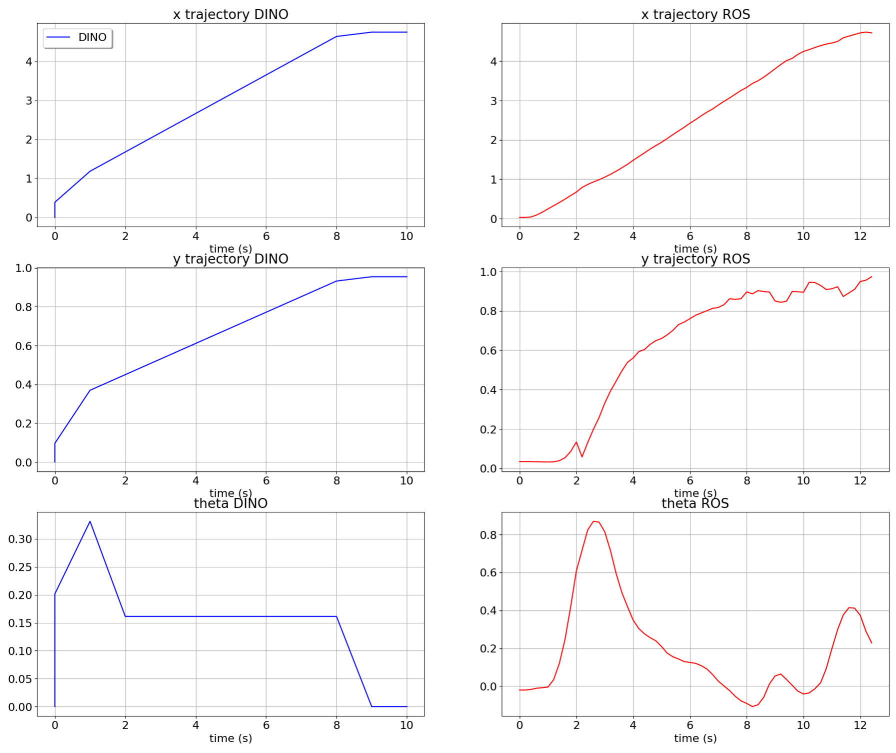
Figure 4.5: Covariance evolution for different test cases of the Playground world.



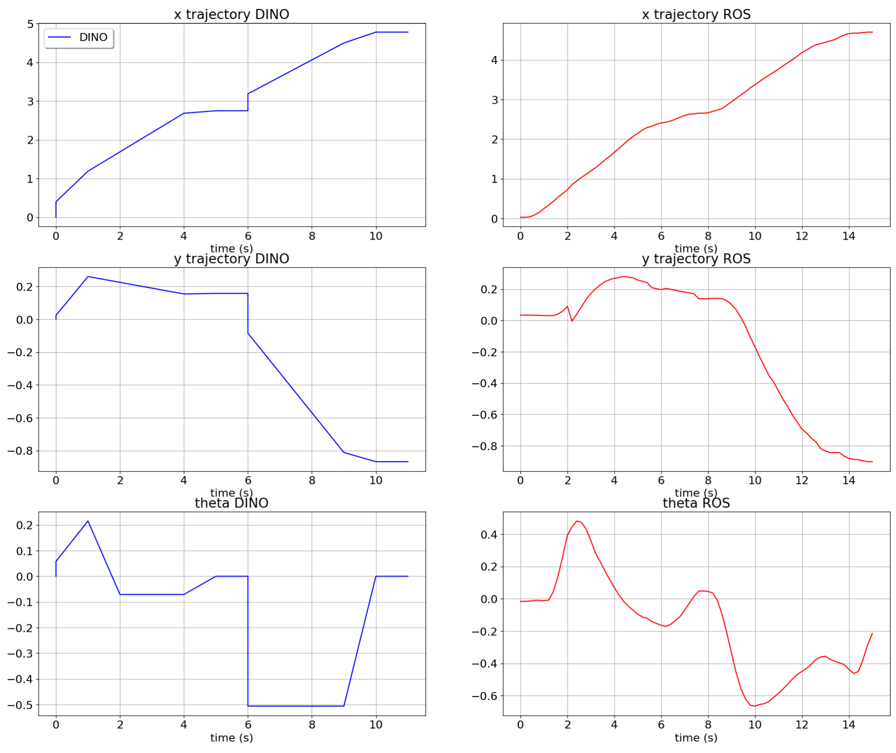
(a) Case 1 (b) Case 2

(c) Case 3 (d) Case 4

Figure 4.6: Playground test case.

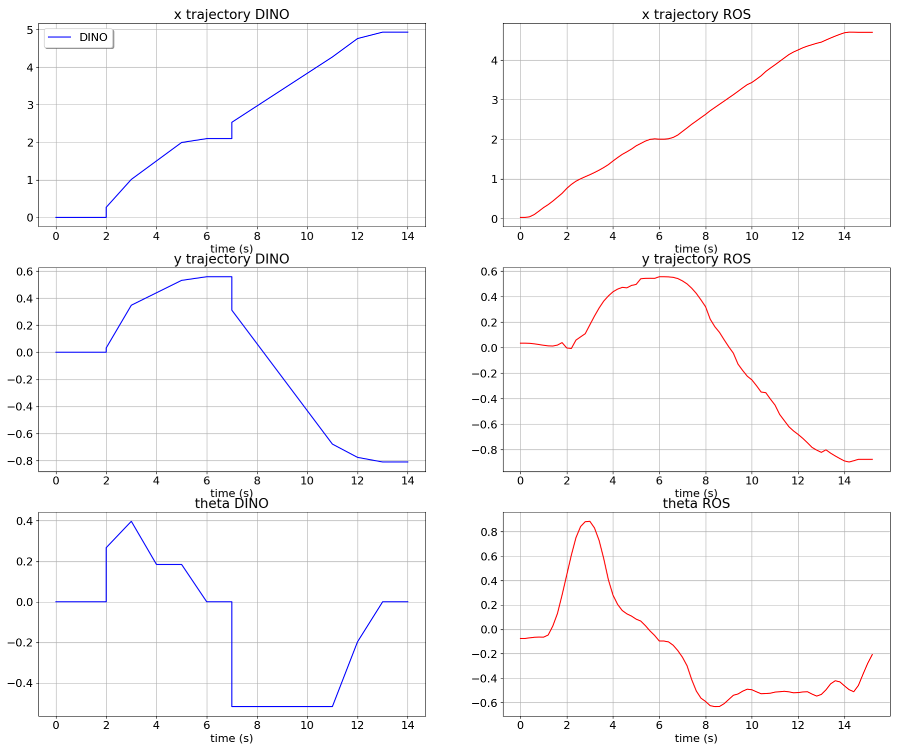


(a) Case 1

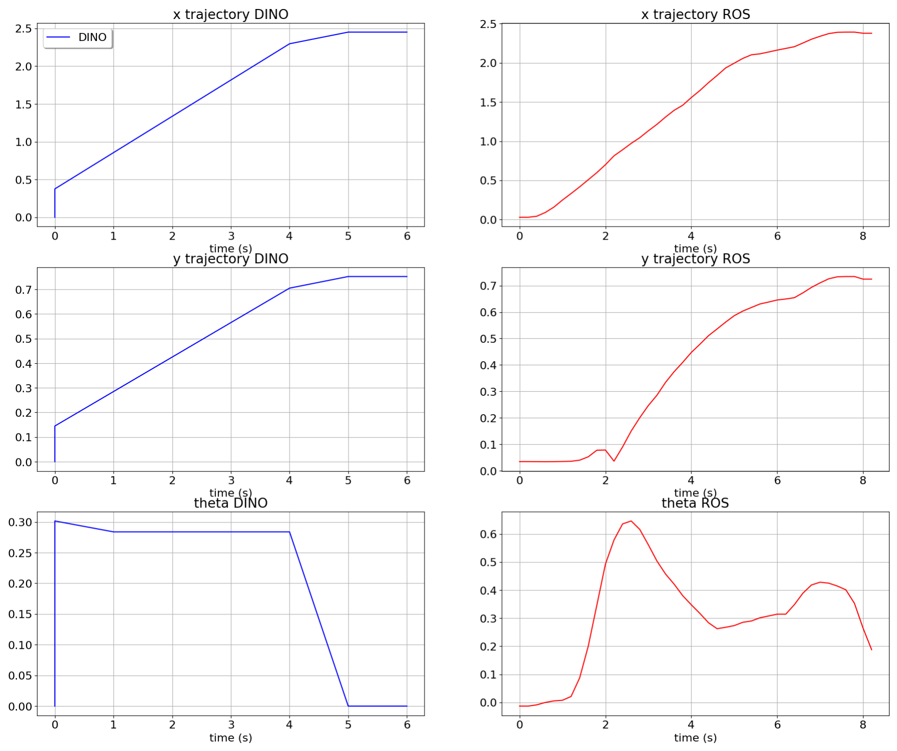


(b) Case 2

Figure 4.7: Playground trajectory test case.



(a) Case 3



(b) Case 4

Figure 4.8: Playground trajectory test case.

4.2.2 Corridor Environment

Starting from the initial waypoint (s in figure), turtlebot needs to reach near the only plug point or the goal waypoint (g in figure) to recharge the batteries. The turtlebot is initially oriented towards g. Due to the short length of the charging cable, given the mean goal pose, there is a bound on the maximum pose uncertainty the robot can afford. The cubes marked 1-4 are the landmarks in environment. The *slam_gmapping* ROS package is used to build the environment map. The resulting map of the environment is shown in Figure 4.10a. The turtlebot with its laser scanner can be seen facing the goal waypoint marked in blue.

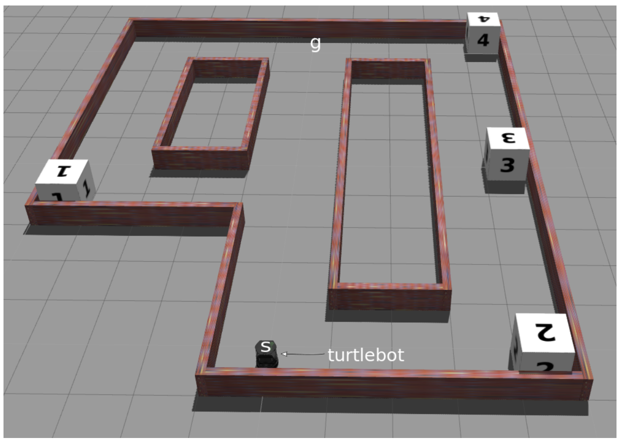
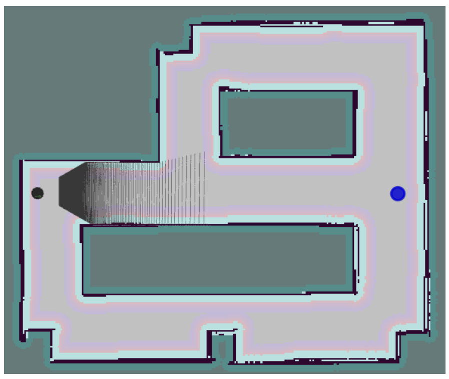
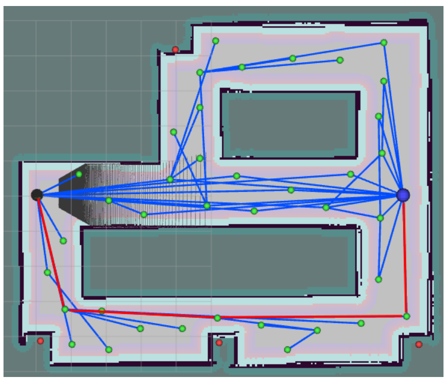


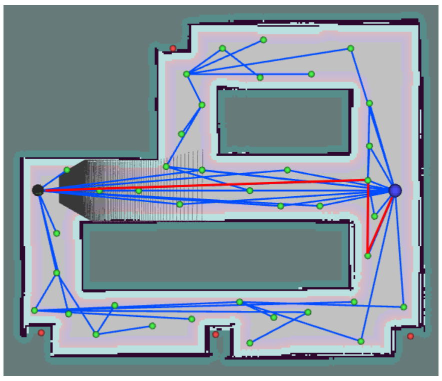
Figure 4.9: Corridor environment in Gazebo.

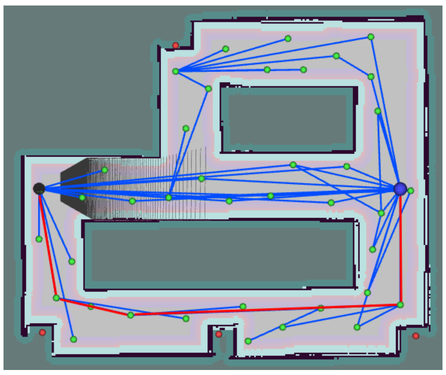
In the remainder of this section we discuss a number of test cases performed with the same initial waypoint and goal waypoint as shown in Figure 4.9. The initial variance in x, y, θ are 0.6m^2, 0.6m^2 and 0.02 rad respectively, giving $(\Sigma_0) = 1.22$. PDDL+ process discretization of $\Delta = 1$ and a temporal horizon of $T = 20$ is used but with different values of η. Motion discretization factor $dFactor = 1$, giving $\delta_{trans_k} = 1$. Performing the belief updates at each δ_{trans_k} helps in pruning the nearby waypoints and thereby reducing the state space explosion. However, in the *difficult* regions where one cannot afford to prune close-by waypoints, the updates are performed upon reaching each of these waypoints. Unless otherwise mentioned, the number of waypoints for each case $m = 40$. The different test cases are detailed below.

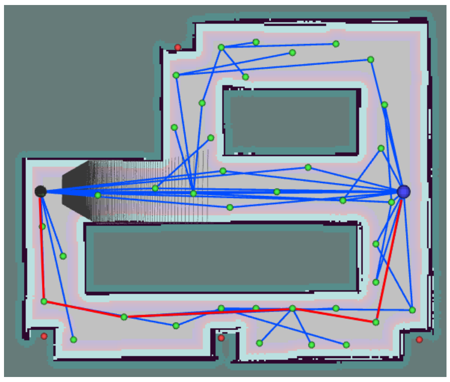
Case 1: For this case $\eta = 0.3$. The plan generated is shown via the red path in Figure 4.10b, starting from initial state to the goal state. This plan is quite expected due to the landmark rich nature of the path and the tight bound on the final trace.


(a) Initial configuration


(b) Case 1


(c) Case 2


(d) Case 3


(e) Case 4

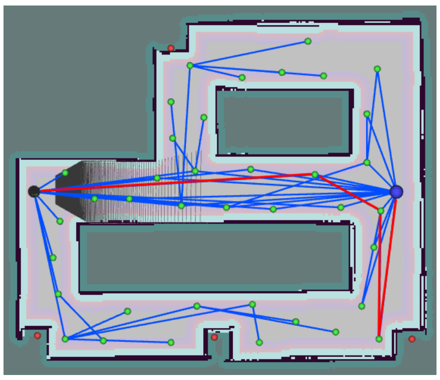
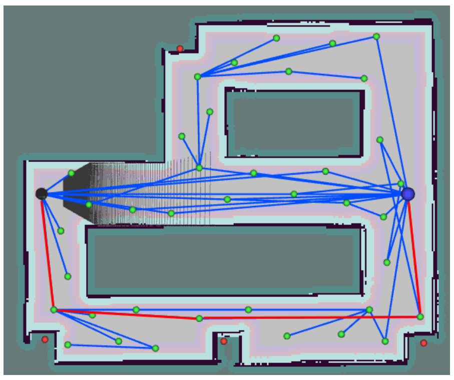
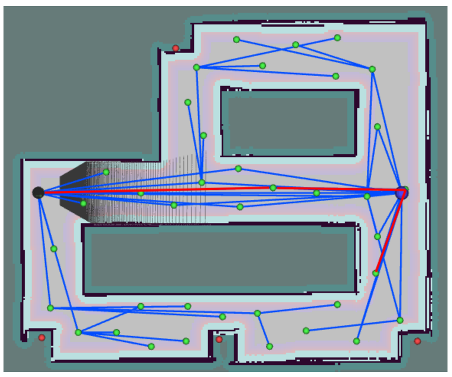
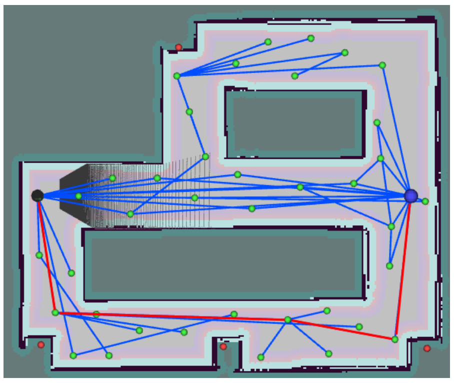

(f) Case 5

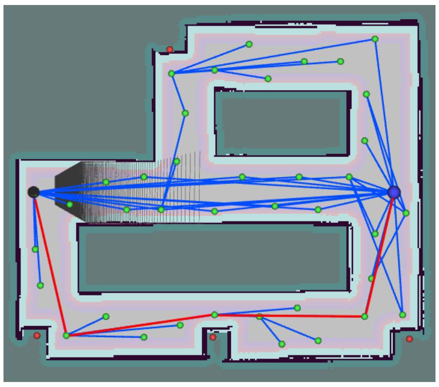
Figure 4.10: Initial configuration and the trajectories for the different test cases.

Case 2-5: For Case 2 $\eta = 0.6$. The plan generated is the one following the red path shown in Figure 4.10c. Case 3 has the same $\eta = 0.6$, however, the path followed is along the one with 3 landmarks (Figure 4.10d). This is due to the


(g) Case 6


(h) Case 7


(i) Case 8


(j) Case 9

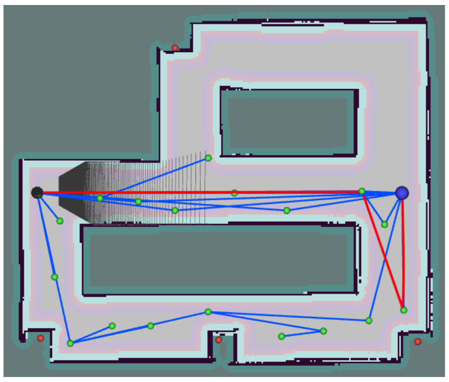

(k) Case 10

Figure 4.10: Trajectories for the remaining test cases.

randomness of the potential field based RRT motion planner and the bound on the trace being less tight. Since the objective function requires the planning time to be minimized, the planner returns the path which minimizes the makespan

Test cases	States expanded	Plan time (s)
Case 1	6184	116.62
Case 2	6934	156.62
Case 3	33769	744.36
Case 4	2710	60.64
Case 5	4847	99.74
Case 6	7646	190.44
Case 7	7289	143.04
Case 8	56544	1119.00
Case 9	28595	684.60
Case 10	388	4.60

Table 4.1: Different test case with number of states expanded in each case and the corresponding planning time in seconds.

which in turn is determined by the waypoint locations in the environment. As seen in Table 4.1, the states expanded and the planning time is much larger than in Case 2 (around 4.8 times). This is due to the large number of connected waypoints. Case 4 has $\eta = 0.6$ and the path followed (see Figure 4.10e) is similar to the one in Case 3. However, as evident from Figures 4.10e, in this case the number of connected waypoint is much less, resulting in a much lesser planning time (see Table 4.1). Case 5 (see Figure 4.10f) is similar to the Case 2 but with differences in the number of connected waypoints. This is reflected in the states expanded and the planning times as seen in Table 4.1.

Case 6-8: Cases 6, 7 and 8 have $\eta = 0.9$. However as discussed previously the sampling of waypoints affects the plan as can be seen in Figures 4.10g- 4.10i. In Figure 4.10g the plan generated is such that the trajectory upon reaching a waypoint near the goal is extended to a waypoint near landmark 4, from which it can be observed. Case 8 has the highest number of connected waypoints and this is reflected in the corresponding row in Table 4.1.

Case 9-10: Case 9 is similar to Case 1 but with higher number of connected waypoints. In Case 10, $m = 20$ and $\eta = 0.6$. The reduction in waypoint significantly reduces the state space explosion and the planning time, as can be seen in the last row of Table 4.1. However it can be seen in Figure 4.10k that the upper path has no waypoints sampled. Though it is true that for all the cases discussed so far this doesn't cause any alarm, in general a lesser number of waypoints might result in no plans being produced, even though there exists one. For example, changing the starting or the goal location in our experiments can lead to a path via landmark 1. Finding such a minimum number of waypoints is another planning problem by itself.

Furthermore, test cases have been carried out to check the trajectory between the planner and the ROS execution. The trajectory generated by the planner follows the odometry model in EKF while the ROSPlan uses DWA planner [42] for generating trajectory. The trajectory model followed by the ROSPlan is a velocity model. It can be observed in Case 1 4.11a, the position of the robot follow similar trajectory in x and y direction. However we can see a sharp change in the theta parameter. The same can be observed in Case 3 4.12a and Case 8 4.14b. In all the three cases, the robot has taken a sharp 90 degree turn to go to the goal. It can be observed that there is a slight jerk in the ROS trajectory, this is because the robot is trying to re-align its position based on SLAM and it is acceptable to have certain degree of error and correction in such realistic mapping.

In all cases, we can see a smoother trajectory in the ROS section because of the higher time-step and better planner. The local planner used by ROSPlan reduces the search space to the dynamic window, which consists of the velocities reachable within a short interval of time. In Case 2 4.11b, we see that a sharp change in consecutive orientation value because of the shift in direction from waypoint lower than the endpoint, and back to the endpoint. In such a case, the local planner is intelligently correcting such errors and building a smoother trajectory.

The similarity of our planner's trajectory with the trajectory from ROS further ascertains that the odometry model we assumed while planning the path is close to realistic velocity model and that our plans are valid. The ROS trajectory takes a longer time because of the AMCL algorithm used simultaneously with the local planner for visual feedback. In case of corridor world, the particle filter used is not able to differentiate the repetitive pattern of walls properly when in a forward motion 3.6.

We can see the change in covariance in the planner from Figure 4.16. Since different complexity of problem and plan has been tested for the corridor world, the generated pattern of covariance are in all diverse. We see that in all those cases where robot almost reaches way point but has not passed thorough any landmark diverts to a close by landmark to reduce covariance. In these cases, covariance rises higher than 3 until it visits a landmark which reduces it below the required threshold.

4.3 Performance Analysis

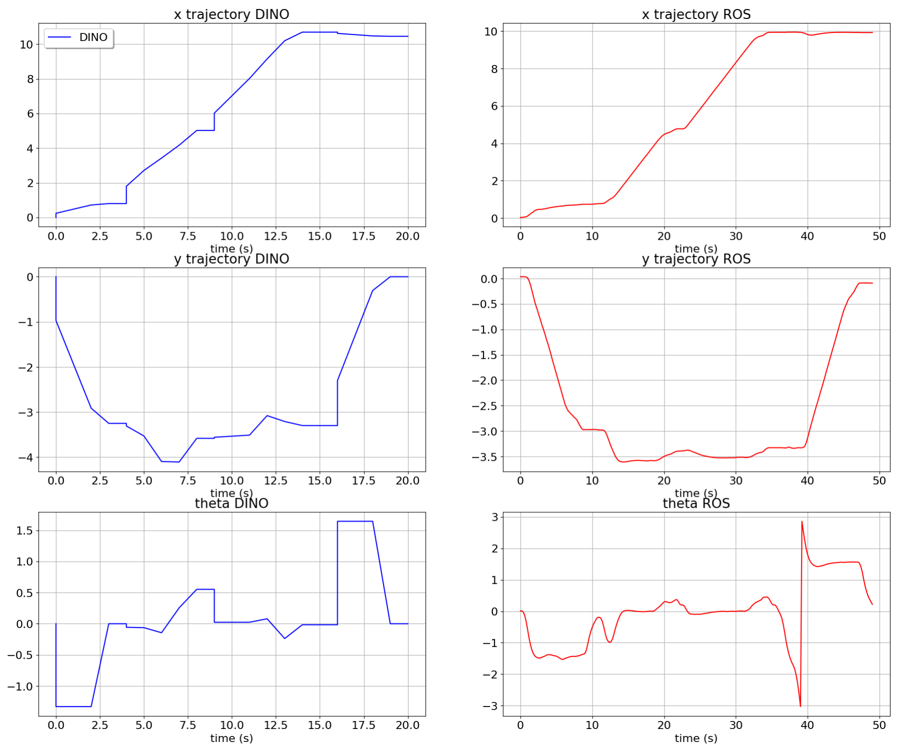
To analyze the performance based on the temporal planning horizon and the PDDL+ process discretization, we perform different simulations using Case1 in Section 4.2 as our base case. We use different discretization Δ and time horizon T

and analyze the states explored, computation/plan time t and the goal covariance trace (Σ_g).

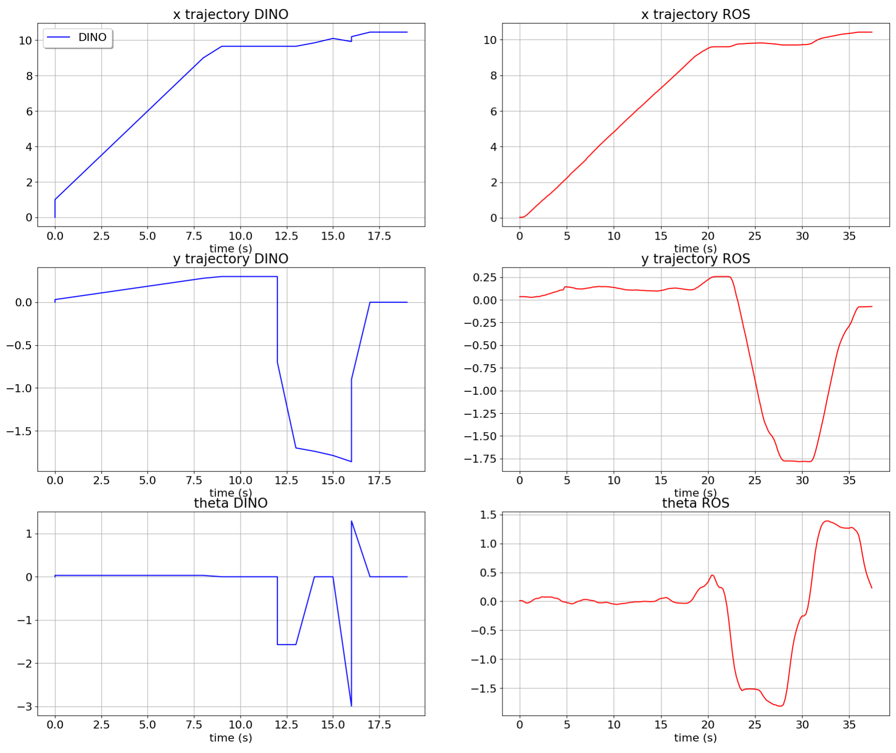
Δ	T	States expanded	t (s)	(Σ_g)
0.5	20	47500	1231.60	0.076
0.5	25	112601	N/A	N/A
0.5	30	112601	N/A	N/A
0.5	35	112601	N/A	N/A
1.0	20	6184	121.88	0.153
1.0	25	12788	249.14	0.156
1.0	30	23533	566.44	0.143
1.0	35	41682	1092.96	0.181
2.0	20	750	17.56	0.267
2.0	25	1290	39.66	0.273
2.0	30	995	37.98	0.290
2.0	35	967	47.78	0.255

Table 4.2: Performance parameters for different temporal planning horizons and PDDL+ process discretization.

From table 4.2 we can observe that the planning time is inversely proportional to the discretization Δ. This is because we perform belief updates based on $\delta_{trans_k} = \Delta * dFactor$. Since $dFactor = 1$, the number of updates is directly proportional to $Delta$, thereby increasing the states expanded and hence the plan time. It is seen that the temporal horizon and total states searched are correlated, directly affecting the total plan time. It can be observed from table 4.2 that for a longer temporal horizon T and a larger Δ, the goal condition is satisfied by expanding fewer states. Larger the value of T, the more deeper SRPG is built resulting in better heuristic. However, for 2 given waypoints, lower value of Δ implies additional belief updates, leading to more states being expanded. Conversely, lower discretization and a larger time horizon will not result in successful plans. Longer T can lead to more states being expanded and a large Δ can lead to unvalidated plans. Hence an efficient plan requires a trade-off between the two.

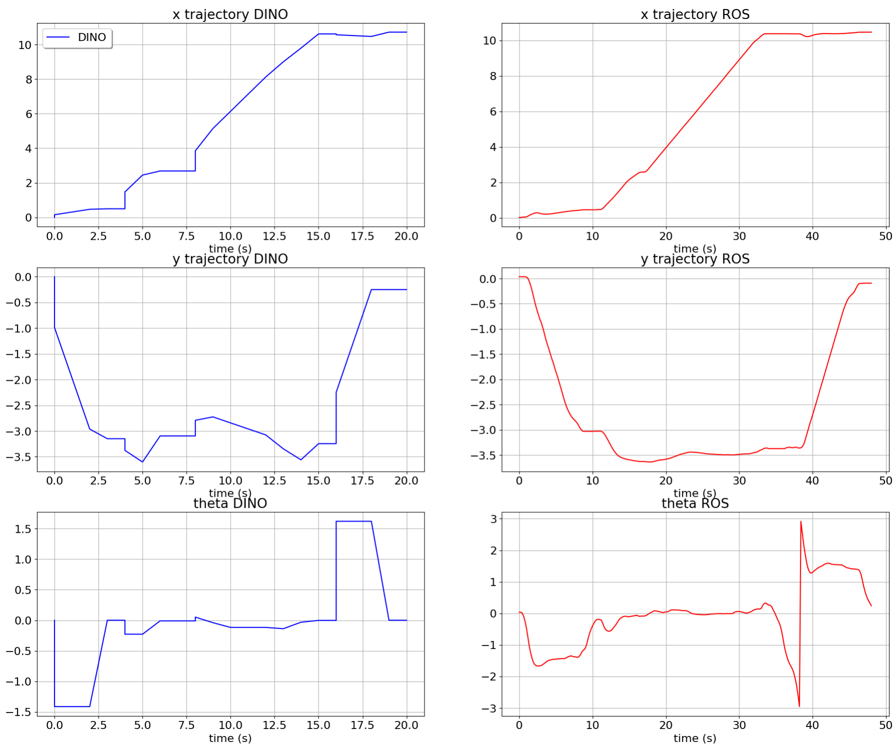


(a) Case 1

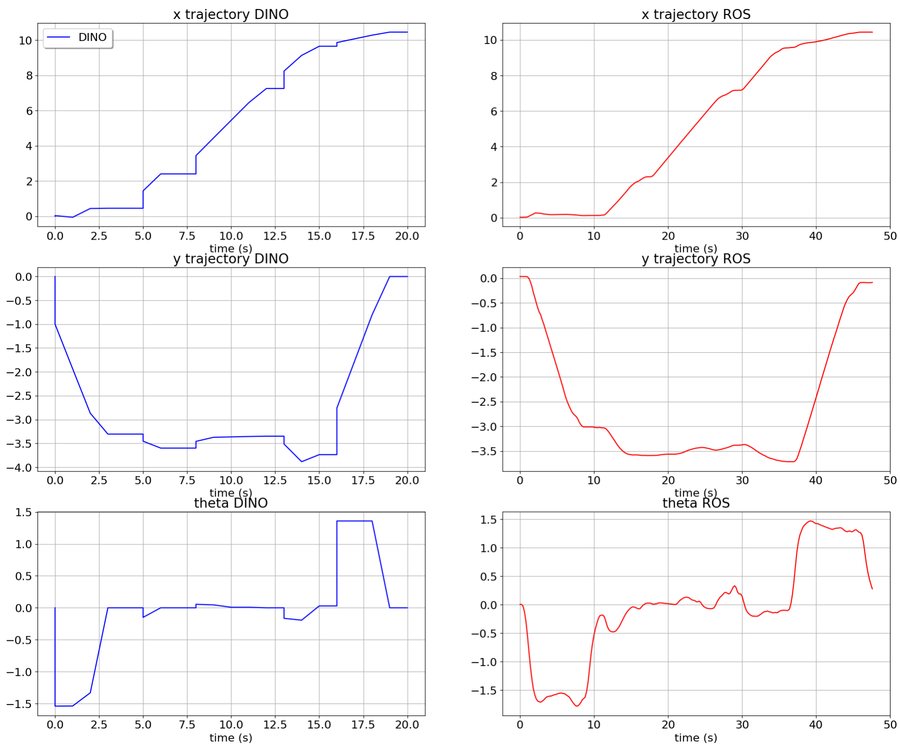


(b) Case 2

Figure 4.11: Trajectory generated by the planner and trajectory executed by ROSPlan.

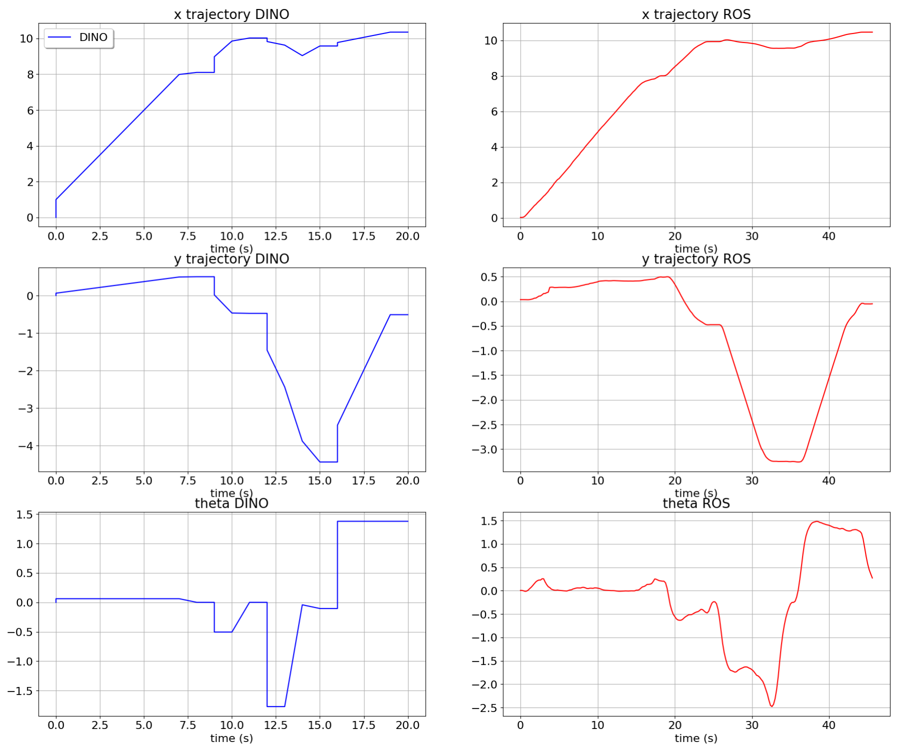


(a) Case 3

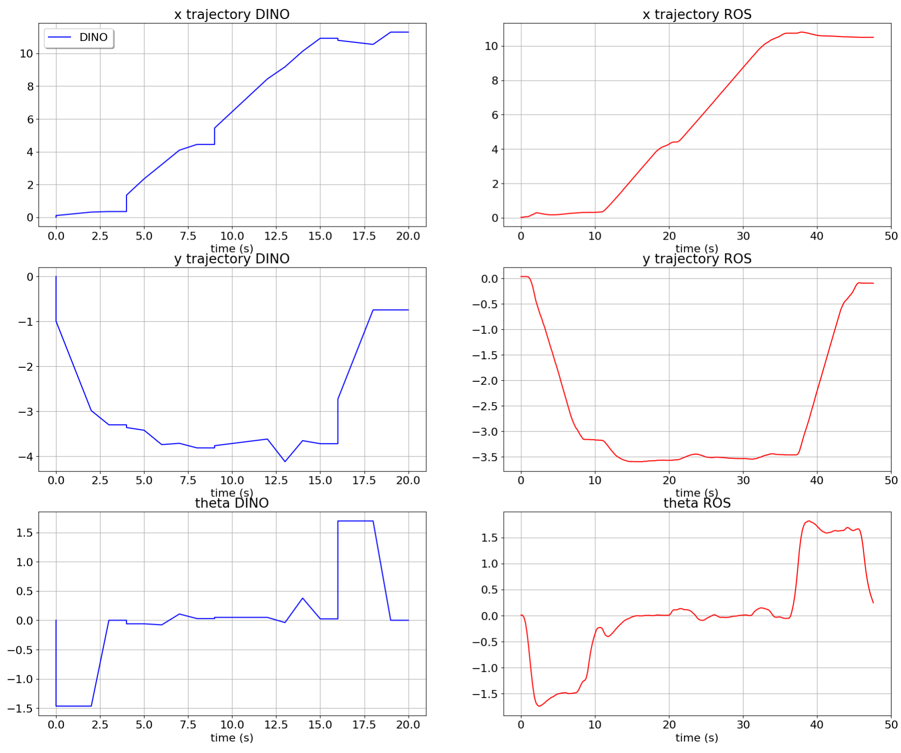


(b) Case 4

Figure 4.12: Trajectory generated by the planner and trajectory executed by ROSPlan.

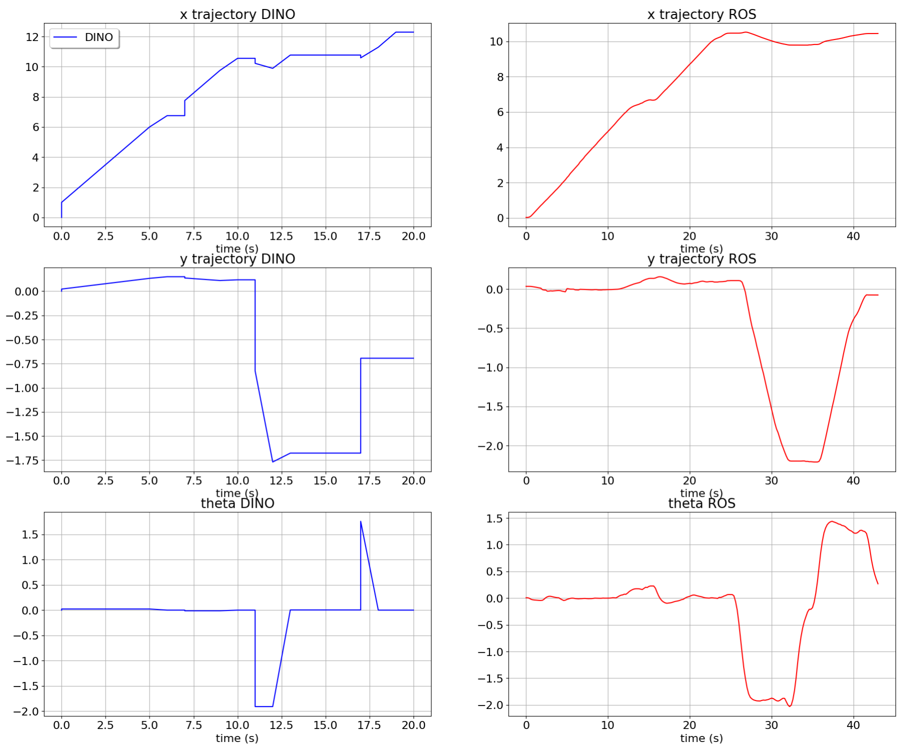


(a) Case 5

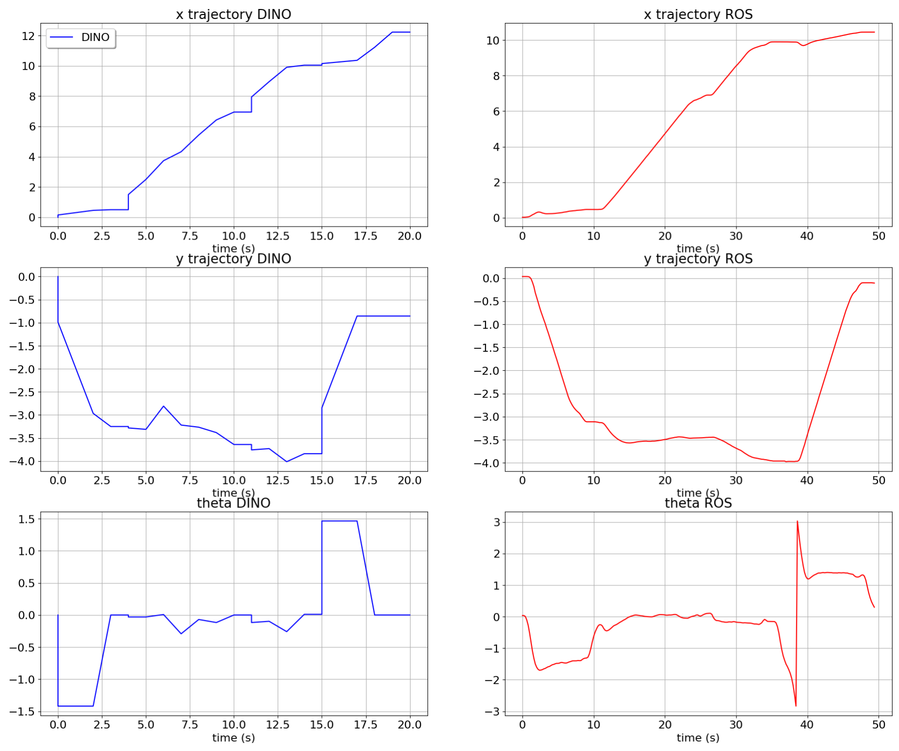


(b) Case 6

Figure 4.13: Trajectory generated by the planner and trajectory executed by ROSPlan.

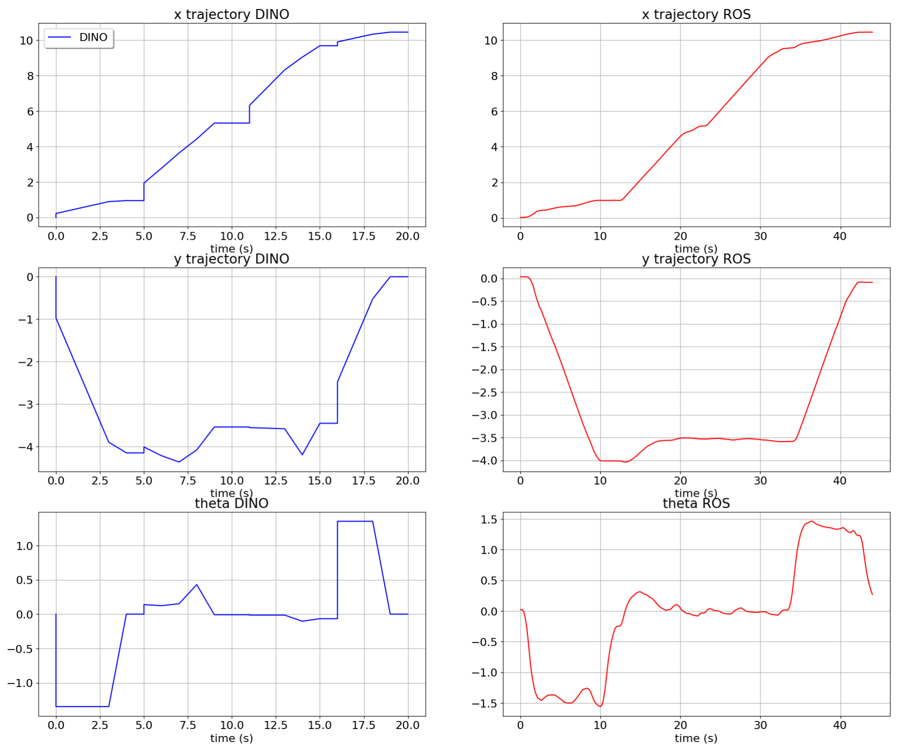


(a) Case 7

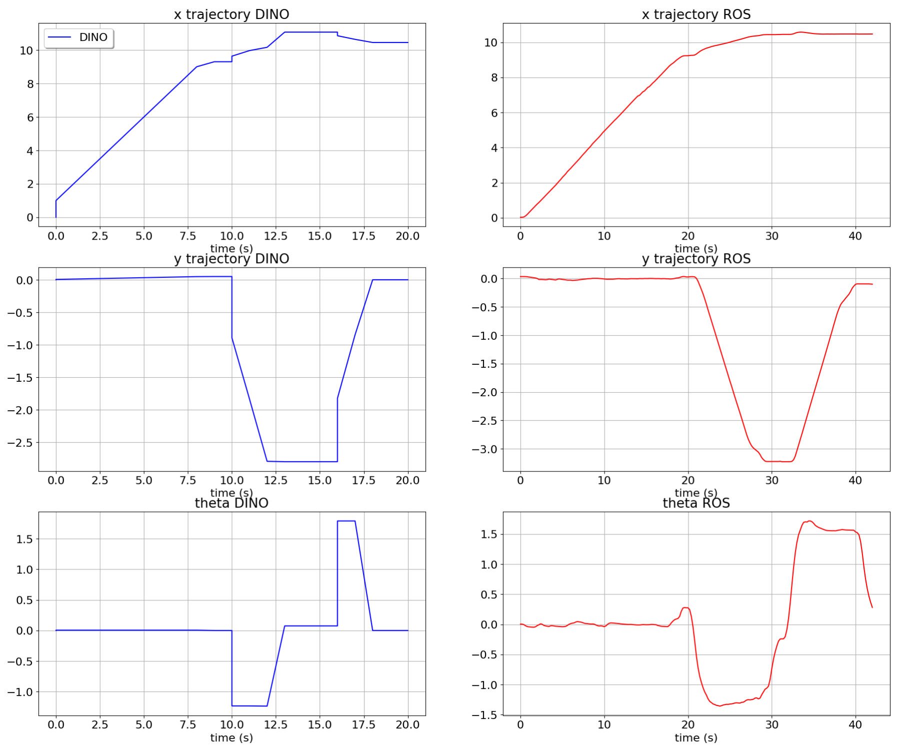


(b) Case 8

Figure 4.14: Trajectory generated by the planner and trajectory executed by ROSPlan.



(a) Case 9



(b) Case 10

Figure 4.15: Trajectory generated by the planner and trajectory executed by ROSPlan.

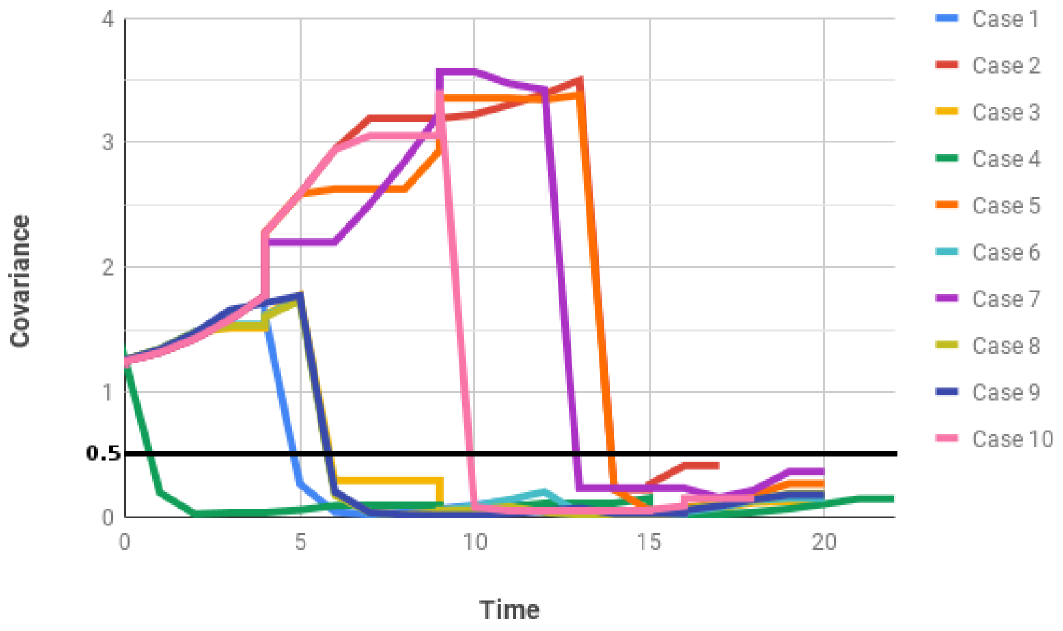


Figure 4.16: Covariance for corridor case 1 - 10

Chapter 5

Conclusion and Future Work

We have developed a combined Task and Motion Planner capable of reasoning in belief space that produces a viable plan. Using a high level PDDL+ planning framework we are able to work in discrete and continuous non-linear systems, handling the predicates and rules defined in each domain. Furthermore we integrated an external library to perform semantic attachments that computes the complex non-linear state space evolution, generating a staged relaxed plan graph utilizing breadth first search module to thoroughly search for optimal goal.

Expressive power of PDDL+ combined with heuristic base semantic attachments simulate the belief evolutions given an action sequence and the corresponding expected future observations. The underlying methodology of the hybrid planner has been discussed, validating the approach using realistic synthetic simulations in Gazebo. For the probabilistic completeness, we have implemented a potential field based RRT algorithm for efficient waypoint sampling. This method quantitatively produced more robust distribution of waypoints while reducing computational complexity by making only necessary waypoints connected. It also considered distributions around landmark through potential field which evidently helped in reducing system uncertainties and giving optimum plan. Our use of ROSPlan for seamless integration of planner, semantic attachment, problem generation and execution was also successful in completing the task for different test cases in a single command. This also gives us an option to plug any other planner or executor easily without affecting rest of the framework.

While this is a novel work, there is scope for further improvements with multiple upgrades in all the software and algorithmic sections. To being with, the current version of planner can be improved to include metric covariance, so that an option to get best plan with least covariance in concurrence with time can be generated. To utilize the model checking feature of the planner, advanced algorithm for multi-robot simulation can be implemented.

Similarly, the RRT model with potential field currently takes the center of an object as the origin and perceives a potential field around it. However, contrary to current implementation, the field magnitude should have peaks only around the view points from which landmarks can be observed facilitating a perception-aware model. This can be performed in concurrence with SLAM for better map building.

In our test cases we use the default AMCL for navigation. The particle filter used in AMCL can be modified according to the environment or the module can be improved with better sensors for localization and mapping. One of the problems of AMCL in our algorithm is its limitation to 2D mapping. For multi-robot heterogeneous system, a 3D reconstruction of environment is essential. Hence, navigation module should be changed to a 3D based SLAM type module.

The external module still relies on the odometry model for mobile robot to perform localization using EKF estimates. This is not robust to handle the physical limitation of the mobile robot. A better alternative would be to use velocity model of the robot similar to the one used by ROSPlan framework which takes into account the dynamics of the robot's motion for better plan calculation.

Bibliography

[1] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL- The Planning Domain Definition Language. In *AIPS-98 Planning Competition Committee*, 1998. 1, 2, 8

[2] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Robotics and Automation (ICRA), IEEE International Conference on*, pages 639–646. IEEE, 2014. 1, 7, 15

[3] Stéphane Cambon, Rachid Alami, and Fabien Gravot. A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research*, 28(1):104–126, 2009. 1, 7

[4] Leslie P Kaelbling and Tomás Lozano-Pérez. Integrated robot task and motion planning in the now. Technical Report 2012-018, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2012. 1, 7

[5] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic Attachments for Domain-Independent Planning Systems. In *Towards Service Robots for Everyday Environments*, pages 99–115. Springer Berlin Heidelberg, 2012. 1, 9

[6] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015. 1, 7

[7] Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research*, 0(0):0278364918761570, 0. 1, 7

[8] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998. 2

[9] SJ Russell and P Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2002. 2

[10] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971. 2, 6

[11] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001. 3, 27

[12] Henry Kautz and Bart Selman. Unifying sat-based and graph-based planning. In *IJCAI*, volume 99, pages 318–325, 1999. 3

[13] Maria Fox and Derek Long. Modelling Mixed Discrete-Continuous Domains for Planning. *Journal of Artificial Intelligence Research*, 27(1):235–297, 2006. 3, 9

[14] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991. 3

[15] Lydia E Kavraki and Steven M LaValle. Springer handbook of robotics, 2008. 4

[16] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011. 4

[17] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996. 4

[18] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000. 4

[19] Nils J Nilsson. Shakey the robot. Technical Report 323, Airtificial Intellignece Center, SRI International, Menlo Park, California, 1984. 6

[20] Christian Dornhege, Marc Gissler, Matthias Teschner, and Bernhard Nebel. Integrating symbolic and geometric planning for mobile manipulation. In *Safety, Security & Rescue Robotics (SSRR), IEEE International Workshop on*, pages 1–6. IEEE, 2009. 6

[21] Jörg Hoffmann. The Metric-FF Planning System: Translating “Ignoring Delete Lists” to Numeric State Variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003. 7, 10

[22] Wiktor Mateusz Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni, and Fabio Mercorio. Heuristic Planning for PDDL+ Domains. In *AAAI Workshop: Planning for Hybrid Systems*, Phoenix, Arizona, USA, July 2016. 7

[23] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), IEEE International Conference on*, pages 1470–1477. IEEE, 2011. 7

[24] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227, 2013. 7

[25] Camille Phiquepal and Marc Toussaint. Combined task and motion planning under partial observability: An optimization-based approach. *RSS Workshop on Integrated Task and Motion Planning*, 2017. 7

[26] Tomás Lozano-Pérez and Leslie Pack Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pages 3684–3691. IEEE, 2014. 7

[27] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 2003. 8

[28] J Hoffmann and S Edelkamp. The classical part of ipc-4: An overview. *Journal of AI Research, To appear*, 2005. 8

[29] Ji-Ae Shin and Ernest Davis. Processes and continuous change in a sat-based planner. *Artificial Intelligence*, 166(1-2):194–253, 2005. 8

[30] Giuseppe Della Penna, Benedetto Intrigila, Daniele Magazzeni, and Fabio Mercorio. Upmurphi released: Pddl+ planning for hybrid systems. In *Proceedings of the 2nd ICAPS Workshop on Model Checking and Automated Planning (MOCHAP-15)*, pages 36–40, 2015. 8, 9

[31] Amanda Jane Coles and Andrew Ian Coles. Pddl+ planning with events and linear processes. In *ICAPS*, 2014. 8

[32] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-chaining partial-order planning. In *ICAPS*, pages 42–49, 2010. 8

[33] Wiktor Mateusz Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni, and Fabio Mercorio. Heuristic planning for pddl+ domains. In *AAAI Workshop: Planning for Hybrid Systems*, 2016. 9, 27

[34] Michael Cashmore, Maria Fox, Derek Long, and Daniele Magazzeni. A compilation of the full pddl+ language into smt. In *AAAI Workshop: Planning for Hybrid Systems*, 2016. 9

[35] Franc Ivankovic, Patrik Haslum, Sylvie Thiébaux, Vikas Shivashankar, and Dana S Nau. Optimal planning with global numerical state constraints. In *ICAPS*, 2014. 9

[36] Sara Bernardini, Maria Fox, Derek Long, and Chiara Piacentini. Boosting Search Guidance in Problems with Semantic Attachments. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 29–37, Pittsburgh, PA, USA, June 2017. 10

[37] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005. 10, 13, 14

[38] Jonathan Ferrer-Mestres, Guillem Frances, and Hector Geffner. Combined task and motion planning as classical ai planning. *arXiv preprint arXiv:1706.06927*, 2017. 14

[39] Héctor Geffner. Functional strips: a more flexible language for planning and problem solving. In *Logic-based artificial intelligence*, pages 187–209. Springer, 2000. 14

[40] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1470–1477. IEEE, 2011. 14

[41] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. Rosplan: Planning in the robot operating system. In *ICAPS*, pages 333–341, 2015. 15

[42] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997. 55

Appendix A

Codes and Files

A.1 DINO

A.1.1 upm_system

https://bitbucket.org/LittleSunny/masters- thesis/src/master/DiNo/include/
upm_system.cpp

A.1.2 upm_statecl

https://bitbucket.org/LittleSunny/masters-thesis/src/master/DiNo/include/
upm_statecl.hpp

A.1.3 upm_util

https://bitbucket.org/LittleSunny/masters-thesis/src/master/DiNo/include/
upm_util.hpp

A.1.4 upm_io.cpp

https://bitbucket.org/LittleSunny/masters-thesis/src/master/DiNo/include/
upm_io.cpp

A.1.5 cpp_code.cpp

https://bitbucket.org/LittleSunny/masters-thesis/src/master/DiNo/src/DiNo/
cpp_code.cpp

A.1.6 ump_epilog.hpp

https://bitbucket.org/LittleSunny/masters-thesis/src/master/DiNo/include/upm_epilog.hpp

A.2 External Advisor

https://bitbucket.org/LittleSunny/masters-thesis/src/master/DiNo/ex/ext_lib/

A.3 ROSPlan

A.3.1 PDDLProblemGenerator.cpp

https://bitbucket.org/LittleSunny/masters-thesis/src/master/src/ROSPlan/rosplan_planning_system/src/ProblemGeneration/PDDLProblemGenerator.cpp

A.3.2 planningsystem.cpp

https://bitbucket.org/LittleSunny/masters-thesis/src/master/src/ROSPlan/rosplan_planning_system/src/PlanningSystem.cpp

A.3.3 plannerinterface.cpp

https://bitbucket.org/LittleSunny/masters-thesis/src/master/src/ROSPlan/rosplan_planning_system/src/PlannerInterface.cpp

A.3.4 DINOEsteralPlanParser.cpp

https://bitbucket.org/LittleSunny/masters-thesis/src/master/src/ROSPlan/rosplan_planning_system/src/PlanParsing/DINOEsterelPlanParser.cpp

A.3.5 interfaced_planning_system.launch

https://bitbucket.org/LittleSunny/masters-thesis/src/master/src/ROSPlan/rosplan_planning_system/launch/interfaced_planning_system.launch

A.3.6 RPRoadmapServer.cpp

https://bitbucket.org/LittleSunny/masters-thesis/src/master/src/ROSPlan/rosplan_interface_mapping/src/RPRoadmapServer.cpp

A.3.7 RPRoadmapVisualization.cpp

https://bitbucket.org/LittleSunny/masters-thesis/src/master/src/ROSPlan/rosplan_interface_mapping/src/RPRoadmapVisualization.cpp

A.3.8 rosplan_roadmap_server.launch

https://bitbucket.org/LittleSunny/masters-thesis/src/master/src/ROSPlan/rosplan_interface_mapping/launch/rosplan_roadmap_server.launch

A.3.9 toy.launch

https://bitbucket.org/LittleSunny/masters-thesis/src/master/src/ROSPlan/rosplan_demos/launch/toy.launch

A.3.10 turtlebot_explorer.bash

https://bitbucket.org/LittleSunny/masters-thesis/src/master/src/ROSPlan/rosplan_demos/scri

A.4 pred_update_RRT_problem.pddl

```
(define (problem landmark_task)
(:domain landmark)
(:objects
    dummy1 dummy2 - covariance
    kenny - robot
    endpoint wp0 wp1 wp10 wp11 wp12 wp13 wp14 wp15
                wp16 wp17 wp18 wp19 wp2 wp20 wp21 wp22
            wp23 wp24 wp25 wp26 wp27 wp28 wp29 wp3
            wp4 wp5 wp6 wp7 wp8 wp9 - waypoint
)
(:init
    (connected endpoint wp0)
    (connected endpoint wp10)
    (connected endpoint wp11)
    (connected endpoint wp13)
    (connected endpoint wp17)
    (connected endpoint wp19)
    (connected endpoint wp21)
    (connected endpoint wp22)
    (connected endpoint wp23)
    (connected endpoint wp25)
    (connected endpoint wp26)
    (connected endpoint wp29)
    (connected endpoint wp6)
    (connected endpoint wp8)
    (connected wp0 wp1)
    (connected wp0 wp5)
    (connected wp0 wp6)
    (connected wp0 wp7)
    (connected wp0 wp9)
    (connected wp0 wp11)
    (connected wp0 wp14)
    (connected wp0 wp16)
    (connected wp0 wp19)
    (connected wp0 wp21)
    (connected wp0 wp22)
    (connected wp0 wp26)
    (connected wp0 wp27)
    (connected wp0 wp28)
    (connected wp0 endpoint)
    (connected wp1 wp0)
    (connected wp1 wp2)
    (connected wp1 wp3)
    (connected wp1 wp12)
    (connected wp1 wp18)
    (connected wp10 wp8)
    (connected wp10 endpoint)
    (connected wp11 wp0)
    (connected wp11 wp29)
    (connected wp11 endpoint)
    (connected wp12 wp1)
    (connected wp13 wp6)
    (connected wp13 endpoint)
    (connected wp14 wp0)
    (connected wp15 wp3)
    (connected wp16 wp0)
    (connected wp16 wp17)
    (connected wp17 wp16)
    (connected wp17 endpoint)
    (connected wp18 wp1)
    (connected wp19 wp0)
    (connected wp19 endpoint)
    (connected wp2 wp1)
    (connected wp20 wp3)
    (connected wp21 wp0)
    (connected wp21 wp24)
    (connected wp21 endpoint)
    (connected wp22 wp0)
    (connected wp22 endpoint)
    (connected wp23 wp9)
    (connected wp23 wp25)
    (connected wp23 endpoint)
    (connected wp24 wp21)
    (connected wp25 wp23)
    (connected wp25 endpoint)
    (connected wp26 wp0)
    (connected wp26 endpoint)
    (connected wp27 wp0)
    (connected wp28 wp0)
    (connected wp29 wp11)
    (connected wp29 endpoint)
    (connected wp3 wp1)
    (connected wp3 wp4)
    (connected wp3 wp15)
    (connected wp3 wp20)
    (connected wp4 wp3)
    (connected wp5 wp0)
    (connected wp6 wp0)
    (connected wp6 wp8)
    (connected wp6 wp13)
    (connected wp6 endpoint)
    (connected wp7 wp0)
    (connected wp8 wp6)
    (connected wp8 wp10)
    (connected wp8 endpoint)
    (connected wp9 wp0)
    (connected wp9 wp23)
    (observe)
    (robot_at kenny wp0)
    (= (cov) 1.22)
    (= (dFactor) 0.5)
    (= (distance endpoint wp0) 2.56223)
    (= (distance endpoint wp10) 1.69189)
    (= (distance endpoint wp11) 0.585235)
    (= (distance endpoint wp13) 3.45145)
    (= (distance endpoint wp17) 5.41872)
    (= (distance endpoint wp19) 0.756637)
    (= (distance endpoint wp21) 3.62836)
    (= (distance endpoint wp22) 4.62871)
    (= (distance endpoint wp23) 3.19531)
    (= (distance endpoint wp25) 5.82087)
    (= (distance endpoint wp26) 2.12603)
    (= (distance endpoint wp29) 1.36473)
    (= (distance endpoint wp6) 5.50273)
    (= (distance endpoint wp8) 3.45579)
    (= (distance wp0 wp1) 4.28077)
    (= (distance wp0 wp5) 3.50036)
    (= (distance wp0 wp6) 2.94364)
    (= (distance wp0 wp7) 4.5467)
    (= (distance wp0 wp9) 1.93132)
    (= (distance wp0 wp11) 1.978)
    (= (distance wp0 wp14) 1.30096)
    (= (distance wp0 wp16) 6.03013)
    (= (distance wp0 wp19) 2.78792)
    (= (distance wp0 wp21) 1.1)
    (= (distance wp0 wp22) 2.14009)
    (= (distance wp0 wp26) 4.51054)
    (= (distance wp0 wp27) 1.06888)
    (= (distance wp0 wp28) 2.60192)
    (= (distance wp0 endpoint) 2.56223)
    (= (distance wp1 wp0) 4.28077)
    (= (distance wp1 wp2) 2.70046)
    (= (distance wp1 wp3) 1.72627)
    (= (distance wp1 wp12) 1.54029)
    (= (distance wp1 wp18) 2.90259)
    (= (distance wp10 wp8) 2.77308)
    (= (distance wp10 endpoint) 1.69189)
    (= (distance wp11 wp0) 1.978)
    (= (distance wp11 wp29) 1.37295)
    (= (distance wp11 endpoint) 0.585235)
    (= (distance wp12 wp1) 1.54029)
    (= (distance wp13 wp6) 4.67467)
    (= (distance wp13 endpoint) 3.45145)
    (= (distance wp14 wp0) 1.30096)
    (= (distance wp15 wp3) 1.90394)
    (= (distance wp16 wp0) 6.03013)
    (= (distance wp16 wp17) 2.09881)
    (= (distance wp17 wp16) 2.09881)
    (= (distance wp17 endpoint) 5.41872)
    (= (distance wp18 wp1) 2.90259)
    (= (distance wp19 wp0) 2.78792)
    (= (distance wp19 endpoint) 0.756637)
    (= (distance wp2 wp1) 2.70046)
    (= (distance wp20 wp3) 3.85519)
    (= (distance wp21 wp0) 1.1)
    (= (distance wp21 wp24) 3.20975)
    (= (distance wp21 endpoint) 3.62836)
    (= (distance wp22 wp0) 2.14009)
    (= (distance wp22 endpoint) 4.62871)
    (= (distance wp23 wp9) 2.88531)
    (= (distance wp23 wp25) 2.66693)
    (= (distance wp23 endpoint) 3.19531)
    (= (distance wp24 wp21) 3.20975)
    (= (distance wp25 wp23) 2.66693)
    (= (distance wp25 endpoint) 5.82087)
    (= (distance wp26 wp0) 4.51054)
    (= (distance wp26 endpoint) 2.12603)
    (= (distance wp27 wp0) 1.06888)
    (= (distance wp28 wp0) 2.60192)
    (= (distance wp29 wp11) 1.37295)
    (= (distance wp29 endpoint) 1.36473)
    (= (distance wp3 wp1) 1.72627)
    (= (distance wp3 wp4) 1.05475)
    (= (distance wp3 wp15) 1.90394)
    (= (distance wp3 wp20) 3.85519)
    (= (distance wp4 wp3) 1.05475)
    (= (distance wp5 wp0) 3.50036)
    (= (distance wp6 wp0) 2.94364)
    (= (distance wp6 wp8) 5.64646)
    (= (distance wp6 wp13) 4.67467)
    (= (distance wp6 endpoint) 5.50273)
    (= (distance wp7 wp0) 4.5467)
    (= (distance wp8 wp6) 5.64646)
    (= (distance wp8 wp10) 2.77308)
    (= (distance wp8 endpoint) 3.45579)
    (= (distance wp9 wp0) 1.93132)
    (= (distance wp9 wp23) 2.88531)
    (= (finalTrace) 0.5)
    (= (predict_covariance) 0)
    (= (relativeD) 0)
    (= (update_covariance) 1.22)
)
(:goal (and
    (robot_at kenny endpoint)
    (< (cov) finalTrace)
))
(:metric minimize(total-time)))
```

A.5 waypoints_RRT

```
endpoint[2.45,0.75,0.0]
wp0[0,0,0.0]
wp1[-1.05,-4.15,0.0]
wp10[3.85,-0.2,0.0]
wp11[1.9,0.55,0.0]
wp12[0.45,-4.5,0.0]
wp13[-0.25,2.9,0.0]
wp14[0.05,1.3,0.0]
wp15[-4.4,-3.5,0.0]
wp16[-3.65,4.8,0.0]
wp17[-1.6,4.35,0.0]
wp18[-3.9,-4.7,0.0]
wp19[2.35,1.5,0.0]
wp2[1.65,-4.2,0.0]
wp20[-4.9,-1.25,0.0]
wp21[-1.1,0,0.0]
wp22[-1.7,-1.3,0.0]
wp23[3.55,-2.25,0.0]
wp24[-1.35,3.2,0.0]
wp25[3.85,-4.9,0.0]
wp26[3.85,2.35,0.0]
wp27[-0.2,-1.05,0.0]
```

```
wp28[-0.1,-2.6,0.0]
wp29[2.65,-0.6,0.0]
wp3[-2.75,-4.45,0.0]
wp4[-2.65,-3.4,0.0]
wp5[-3.5,-0.05,0.0]
wp6[-2.75,-1.05,0.0]
wp7[-4.5,0.65,0.0]
wp8[2.65,-2.7,0.0]
wp9[0.7,-1.8,0.0]
```

A.6 landmark_RRT

```
lm0[0,1.5,0]
lm1[1.5,-1,0]
lm2[1,-3.5,0]
lm4[-1.8,-3.4,0]
lm5[-3.2,-1,0]
lm6[2.45,0.75,0.0]
```

A.7 Initial test waypoint

```
wp0[0,0.0,0]
wp1[0,4.0,-1.57]
wp2[3,0.0,0]
wp3[2,2.0,0]
wp4[6,-2, 1.57]
wp5[5, 1, 3.14]
```

A.8 Initial test landmark

```
lm0[2,3,0]
lm1[4,6,0]
lm2[6,9, M_PI/3]
lm3[8,8, 0]
lm4[9,10, M_PI/4]
lm5[6,8, M_PI/6]
```