

A graph-based search approach for planning and learning

An application to planar pushing and navigation tasks

SC52010: S&C Literature Research

G.S. Groote

page intentionally left blank.

A graph-based search approach for planning and learning

An application to planar pushing and navigation tasks

by

G.S. Groote

Student Name	Student Number
--------------	----------------

Gijs S. Groote	4483987
----------------	---------

Supervisors: C. Smith, M. Wisse

Daily Supervisor: C. Pezzato

Project Duration: Nov, 2021 - Feb, 2023

Faculty: Faculty of Cognitive Robotics, Delft

Cover: Simulation environment used during the thesis [25].

Style: TU Delft Report Style, with modifications by Daan Zwaneveld



Delft Center for
Systems and Control



Cognitive
Robotics

Abstract

todo: this abstract

Create abstract when main content + results are finished

BTW: summerised feedback from Martijn + Corrado: abstract should be a stand alone text, a mini version with a small intro (clarify 'gap' in current research), explain where the proposed method is based upon and how it fills the 'gap', discuss results which mostly is compare with existing literure, how is the proposed method comparing?

*G.S. Groote
Delft, January 2023*

Contents

Abstract	i
1 Introduction	1
1.1 Research Question	3
1.2 Problem Description	3
1.2.1 Task	4
1.2.2 Assumptions	4
1.2.3 Challenges	6
1.3 Report Structure	7
2 Required Background	8
2.1 System Identification and Control Methods	8
2.1.1 System Identification	8
2.1.2 Control Methods	9
2.2 Estimating Path Existence	9
2.3 Planning	10
2.3.1 Motion Planning	10
2.3.2 Manipulation Planning	15
2.4 Fault Detection	15
2.4.1 Monitoring Metrics	15
3 The Hypothesis and Knowledge graph	16
3.1 Hypothesis Graph	17
3.1.1 Definition	18
3.1.2 The Search and the Execution loop	20
3.1.3 Example	24
3.2 Knowledge Graph	24
3.2.1 Definition	24
3.2.2 Example	25
3.2.3 Edge Metrics	25
4 Results	26
4.1 Proposed Method Metrics	26
4.2 Benchmark Tests	27
4.3 Comparison with related papers	27
4.4 Randomisation	27
4.5 Knowledge Graph On/Off	27
4.6 Discussion	27
5 Conclusions	28
5.1 Drawbacks	28
5.2 Future work	29
Glossary	30
References	31
6 Appendix	34

List of Figures

1.1	Robots used for testing the proposed method	5
1.2	Various objects in the robot environment	6
2.1	Visualisation of the Double tree RRT* motion planner that adds a single sample to the connectivity graph. The colour of the box surrounding subfigures corresponds to the coloured sections in algorithm 1. 3-dimensional configuration space displayed as 2-dimensional configuration space (x and y are visible, θ is not visible).	13
2.2	Comparing schematic example to a visualisation of the real algorithm.	14
2.3	Schematic view of the proposed double RRT* tree taking movable and unknown objects into account with cost to reach a sampled configuration displayed.	14
3.1	Simplified flowchart representation of the proposed method. The thesis focus could be augmented with an ontology and high-level planner as displayed. Such an augmentation would create a framework capable of completing high-level tasks such as cleaning or exploring.	16
3.2	FSM displaying the state of an identification edge	19
3.3	FSM displaying the state of an action edge	19
3.4	The search and execution loop.	20
3.5	Flowchart displaying the hypothesis graph's workflow.	22
3.6	Pushing task through a blocked corridor with the point robot, a green cube to push toward the target ghost state and a red blockade.	24

List of Tables

- 2.1 Compatibility between control and system identification methods for drive actions. . . . 9
- 2.2 Compatibility between control and system identification methods for push actions. . . . 9
- 2.3 Functions used by the algorithm 1 11
- 2.4 Monitor metrics used to monitor if a fault occurred during execution of an edge 15

- 3.1 Terminology of terms used 17
- 3.2 Functions used by the algorithm 1 23
- 3.3 Edge metrics used to rank control methods from ‘good’ to ‘bad’ 25

- 4.1 Proposed method metrics used to compare the proposed method with the state-of-the art 27

create list of symbols, summerising every simbol used (and making sure no symbol has 2 different meanings)

1

Introduction

This chapter narrows the broad field of robotics down to a largely unsolved problem. After an introduction of that problem, it is defined, associated challenges are listed and state-of-the-art methods are discussed.

For robots, it remains a hard problem to navigate and act in new, unseen environments. Most approaches controlling the robot and acting as the robot brain fall into one of two categories. The bulk falls into hierarchical approaches [11, 23, 15]. A hierarchical structure generally consists of a high-level and a low-level component. The high-level task planner tries to understand and model the environment, and the gained knowledge is used to generate action sequences toward a desired goal. The high-level planner sends actions to a low-level module, that takes care of sending input signals toward the robot actuators. The high-level planner has a prediction horizon consisting of an action sequence, a long prediction horizon compared to the low-level planner whose prediction horizon is maximal for a single action. Hierarchical structures generally provide solutions which are computationally efficient but not optimal, meaning the solutions found are the best feasible solutions in the task hierarchy they search. The quality of the solution depends on the hierarchy which is typically hand-coded and domain-specific [29]. Alternatively to hierarchical approaches, solutions can be provided by searching in a joint configuration space of the robot and objects, where the robots configuration space is augmented with environment objects configuration spaces whilst also including dynamic constraints [9, 3, 10]. Such a joint configuration space suffers from a combinatorial explosion and it is practically not possible to find a path connecting a start to the target configuration. Because an action sequence from searching in a joint configuration space is exhaustive, simplifications are leveraged, for example, a search close to the current configuration, lets the robot track this incomplete solution toward a desired goal and search the joint configuration space again close to the new current configuration. Different techniques exist to prevent searching in the entire joint configuration space.

Robots in new unforeseen environments is a broad topic, let's narrow down the scope. The focus lies on the challenging and largely unsolved problem of navigation among movable objects in unknown environments. Consider a simple robot without grippers or arms attached in an environment with movable objects. By sending input toward the wheels the robot can drive around. The environment consists of a flat ground plane, the robot and movable objects. By driving against objects the robot can manipulate objects.

This creates two different modes of dynamics (driving, pushing) where different differential constraints apply to the different modes. Because there is a discontinuity in the constraints the joint configuration space is a piecewise-analytic configuration space, which hinders motion planning considerably [29]. Whilst the robot can interact with the objects in the environment, there is no prior information provided to the robot (e.g. weight, friction coefficient) other than the shape and pose of the object. By interacting with objects, the robot can generate a system model, describing the expected trajectory of an object as a result of a push from the robot. The robot is tasked to relocate a subset of objects by pushing objects.

Applicable to such a robot environment three topics in robotics investigated, these topics are **Navigation Among Movable Objects (NAMO)**, **nonprehensile manipulation planning**, and **learning object dynamics**. Individually a considerable amount of research is done on these topics (NAMO [31, 16, 7, 14, 4, 8], nonprehensile manipulation planning [1, 17, 28, 27, 26, 2], learning object dynamics [24, 5]), combining one topic with another is less researched and combining all three topics even scarcer.

Combining learning, driving and pushing can improve the driving and pushing skills of the robot. Learning is crucial for unforeseen environments, for example during space exploration. Non-prehensile pushing compared to prehensile pushing saves weight, and for a robot with a gripper the ability for nonprehensile pushing comes in handy when the gripper is already in use, for example when a robot relocates a package with its gripper, it encounters a blocked path by an unknown object. Blocked paths and changing environments often coincide, such as a fallen package in a warehouse spilling fluid on the floor, with learning the robot can account for the slippery floor. Finally learning may improve single actions, but it may also improve long-term action planning.

Vega-Brown and Roy investigated NAMO in a piecewise-analytic configuration space to relocate objects [29]. Whilst an optimal plan is obtained with probability one with infinite samples, the algorithm does not learn the pushing relation between the robot and the objects. The ability to learn push dynamics greatly broadens the variety of objects a robot can push. Realistically, robots in warehouses, hospitals or supermarkets might encounter a variety of different objects, but learning dynamics inevitably leads to system model mismatch. A motion planning algorithm should be compatible with learned system models and take model mismatch into account. Learning dynamics is a topic untouched by Vega-Brown and Roy because the robot can rigidly grasp the box so that the robot-box pair behaves as a single rigid body as long as the box is grasped. This model avoids the complexity of real-world pushes, becoming alienated from reality.

Wang et al. takes nonprehensile manipulation planning out of the equation and only focuses on the NAMO problem and learning system dynamics. Learning interaction by embedding unknown objects with affordance information followed by planning with a contact-implicit motion planner toward a robot goal location [31]. By removing push manipulation, finding a path from a start to target configuration simplifies, only a single mode of dynamics is contained in the configuration space and the piecewise-analytic configuration space becomes a configuration space. There exist a variety of sampling-based motion planners that can find a path with properties such as probabilistic completeness, replanning in dynamic environments, planning under uncertainty or asymptotically optimality [12, 7]. Challenging problems introduced by push manipulation are removed by simply removing push manipulation.

Vega-Brown and Roy combined NAMO with nonprehensile manipulation planning, Wang et al. combines NAMO with learning system models. Both combine 2 of the 3 topics allowing for a more in-depth analysis but avoiding the problems introduced by combining all 3 topics (NAMO, nonprehensile manipulation planning and learning system models). Sabbagh Novin et al. does combine all three topics. She proposes to obtain system models by analysing and converting a limited set of robot-object test pushes using Bayesian regression to predict model parameters. Path planning is the result of solving a proposed mixed-integer convex optimization [21] which is tracked by Model Predictive Control (MPC) control, the proposed method is tested in a hospital setting [18] and was later improved upon resulting in lowering trajectory errors [22].

Since Sabbagh Novin et al. uses a gripper to manipulate objects, the research falls into the category of prehensile manipulation. Prehensile manipulation in comparison with nonprehensile manipulation is simpler because a pushed object becomes disconnected easier compared to a gripped object. Her research is specific to legged objects, which limits the set of objects to manipulate considerably.

This thesis proposes a method where the robot should learn robot and object dynamics by interaction, and perform motion and manipulation planning whilst facing a wide variety of objects, tasks and environments. The 3 topics (NAMO, nonprehensile manipulation planning and learning system models) are bundled together with a technique known as **backward induction** also known as **backward tracing**

or **backward search** (this thesis will refer to this technique with the term backward search). The proposed method in this thesis and Sabbagh Novin et al. proposed method solve comparable problems using different techniques, allowing for easy comparison between the results of Sabbagh Novin et al. and the results from the proposed method in this thesis.

1.1. Research Question

To investigate the effect of learning on action selection and on action planning the following research questions have been selected.

Main research question:

How do learned objects' system models improve global task planning for a robot with nonprehensile manipulation abilities over time?

The main research question is split into two smaller more detailed subquestions. Essentially the first research subquestion asks "how does the proposed method work?", which allows to explain the proposed method. The second research subquestion essentially asks: "How does it compare to similar existing methods?", allowing to compare the proposed methods with existing state-of-the-art methods by comparing their results.

Research subquestion:

1. Can the proposed method combine learning and planning for push and drive applications with a technique known as backward search [15]?
2. How does learning system models and remembering interactions compare to only learning system models? And, how does the proposed method compare against the state of the art?

Answering the research subquestions provides a solid base to answer the main research question. The main research question is aimed to test robot abilities in a new environment, and track improvement in that a new environment. Some questions which come up are: Will the robot prefer specific strategies for certain objects? How much improvement will a robot make with some experience? Will the robot converge to a preferred strategy for an object, and will it converge to the same strategy again if its memory is wiped.

1.2. Problem Description

To answer the research questions, tests will be performed in a robot environment. A simple environment is desired because that will simplify testing, yet the robot environment should represent many real-world environments in which robots operate. For the environment, a flat ground plane is selected, since many mobile robots operate in a workspace with a flat floor, such as a supermarket, warehouse or distribution centre. The robots to test should be flat robots, which lowers the chance of tipping over. A 3-dimensional environment is selected, but with a flat floor and a flat robot can be treated as a 2-dimensional problem because the robot and objects can only change position over x and y axis (xy plane parallel to the ground plane) and rotate around the z axis (perpendicular to the ground plane).

Let's start with defining the environment.

Let the tuple $\langle \text{Origin}, \text{Ground Plane}, \text{Ob}, E \rangle$ fully define a robot environment where:

Origin Static point in the environment with a x -, y - and z -axis. Any point in the environment has a linear and an angular position and velocity with respect to the origin

Ground Plane A flat plane parallel with the Origin's x - and y - axis. Objects cannot pass through the ground plane and meet sliding friction when sliding over the ground plane.

Ob A set of objects, $Ob = (obst_1, obst_2, obst_3, \dots, obst_i)$ with $i \geq 1$, an object is a 3-dimensional body with shape and uniformly distributed mass. Examples of objects are given in figure 1.2.

E A set of motion equations describing the behaviour of objects such as gravity, interaction with the ground plane or interaction with other objects. The motion equations are equivalent to the translational dynamics.

A state consists of the linear and angular position and velocity of a point with respect to the environment's origin.

Formally, a **state**, $s_{\text{full}}(k)$ is a tuple of $\langle pos_x(k), pos_y(k), pos_\theta(k), vel_x(k), vel_y(k), vel_\theta(k) \rangle$ where $pos_x, pos_y, vel_x, vel_y, vel_\theta \in \mathbb{R}$, $pos_\theta \in [0, 2\pi)$, k indicates the time step and can be removed for simplicity if the state remains constant for all k .

1.2.1. Task

The research questions want to investigate the effect of learning system models and monitor the effect of learned knowledge over time. Thus the robot needs an incentive to learn object properties, and interactions with the objects in its environment, otherwise it would simply remain standing still in its initial location. Therefore the robot is asked to complete a task. A task is defined as a subset of all objects with associated target states:

$$\text{task} = \langle Obst_{\text{task}}, S_{\text{targets}} \rangle$$

where $Obst_{\text{task}} = (obst_1, obst_2, obst_3, \dots, obst_k) \subset Ob$, $S_{\text{target}} = (s_1, s_2, \dots, s_k)$ and $k \geq 0$

A task is completed when the robot manages to push every object to its target position within a specified error margin.

1.2.2. Assumptions

A complex robot environment is not required to answer the research questions. Therefore influences other than the robots are neglected, some real-world dynamics which are generally negligible are neglected, a workaround is found to measure uncertain properties and the objects in the environment will not tip over. To simplify the pushing and learning problem, several assumptions are taken, which are now listed.

Closed-World Assumption: Objects are manipulated, directly or indirectly only by the robot. Objects cannot be manipulated by influences from outside the environment.

Quasi-Static Assumption: Velocities are small enough that inertial forces are negligible [27].

Perfect Object Sensor Assumption: the robot has full access to the poses and geometry of all objects in the environment at all times.

Tasks are Commutative Assumption: Tasks consist of multiple objects with specified target positions. The order in which objects are pushed toward their target position is commutative.

Objects do not tip over Assumption: Movable objects slide if pushed.

The assumptions taken serve to simplify the problem of task completion. Note that in section 5.2 insight is given to remove all assumptions except the quasi-static assumption. By removing assumptions completing tasks becomes a harder problem, but a more realistic problem closer to real-world applications.

Assumptions might have certain implications, which are now listed. The **closed-world assumption** implies that objects untouched by the robot and with zero velocity component remain at the same position. Completed subtasks are therefore assumed to be completed for all times after completion time.

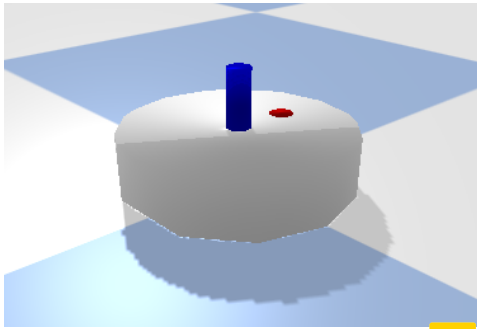
The **quasi-static assumption** allows neglecting complex dynamics, which in many cases are negligible. To ensure that complex dynamics do not become significant during testing, a maximum robot speed or acceleration is enforced.

The **perfect object sensor assumption** simplifies a sensor setup, it prevents Lidar-, camera setups and tracking setups with aruco or other motion capture markers. The existence of a single perfect measurement wipes away the need to combine measurements from multiple sources with sensor fusion algorithms, such as Kalman filtering [30].

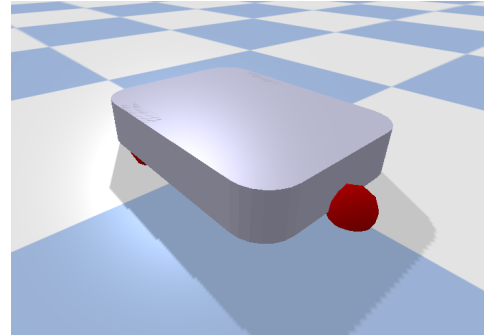
Certain tasks are only feasible if performed in a certain order (e.g. the Tower of Hanoi), the **tasks are commutative assumption** allows focusing only on a single subtask since it does not affect the completion or feasibility of other subtasks.

The **objects do not tip over assumption** ensures that objects do not tip over and suddenly have vastly different dynamics. In practice, objects will not be higher than the minimum width of the object, and spheres are excluded since rolling essentially is tipping.

Robot and Objects, an Example To get a sense of what the robots and the objects look like, see the two robots that are used during testing in figure 1.1. And among many different objects, two example objects are displayed in figure 1.2.



(a) The holonomic point robot
the 2 velocity inputs drive the robot in x and in y direction



(b) The nonholonomic boxer robot
the first velocity input drives the robot forward/backward
the second rotates the robot

Figure 1.1: Robots used for testing the proposed method

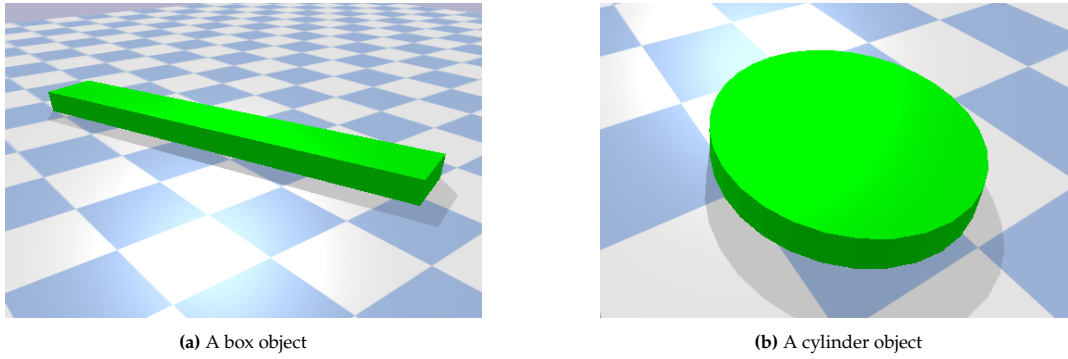


Figure 1.2: Various objects in the robot environment

For complete environments with accompanying tasks, see chapter 4.

1.2.3. Challenges

Finding a solution to a task as defined in section 1.2.1, inside a robot environment defined in section 1.2 bears some challenges. In this section, the main challenge is highlighted and other subchallenges are introduced. Finding a solution requires overcoming the main and subchallenges. Before listing the challenges that this thesis will take on, a challenge is presented that serves to show the difficulty of finding an optimal solution to a task.


Assuming the task has at least one object to place at a target state (exclude purely NP^{D} problems), finding an optimal solution to a task is non-deterministic polynomial-time hard (NP-hard) since, it can be reduced to the piano mover's problem which is known to be NP-hard [20].



Problems in class P have a solution which can be found in polynomial time, problems in non-deterministic polynomial-time (NP) are problems for which a solution cannot found in polynomial time. For problems in NP, when provided with a solution, verifying that the solution is indeed a valid solution can be done in polynomial time. NP-hard problems are a class of problems which are at least as hard as the hardest problems in NP. Problems that are NP-hard do not have to be elements of NP. They may not even be decidable [19]. This thesis or other recent studies in the references do not attempt to find an optimal solution. Instead, they provide a solution whilst guaranteeing properties such as near-optimality or probabilistic completeness.

Finding a solution to a task requires a search in the *joint configuration space* which comes with 2 problems to tackle. Before investigating a problem, let's investigate how to construct a joint configuration space which is created by augmenting the robot's configuration space with the configuration space of every object. For example, if the configuration space for both robot and objects consist of position x , y and orientation θ around the z axis, then the joint configuration space is $3n$ -dimensional, where n is the number of objects including the robot. The joint configuration space's dimensionality grows linearly, meaning the joint configuration space grows exponentially in the number of objects, which is an explosion in the number of possible combinations the environment can be in, the enormity of the joint configuration space is the root of the main challenge.

Unspecified target positions are a fine example of the curse of dimensionality, consider a blocked corridor. The object needs to be pushed to free the path but the target location of this object is unspecified, as long as the robot can drive through the corridor unhindered. A solution is, the robot drives to the object, pushes the object, and then drives through the corridor. The push action influences the free space where the robot has to drive in when the push action was completed. Even sampling-based search algorithms cannot computationally find a path between a start en target state in joint configuration space in reasonable time (orders of magnitude slower than real-time). Only by leveraging simplifications of the joint configuration space a search can be performed, such as discretization [21], factorization [29] or a heuristic function combined with a time horizon [21]. Such techniques prevent searching in configurations relatively far from the current configuration, while optimality guarantees can be given

and real-time implementations have been shown.

The  second problem is that the joint configuration space is *piecewise-analytic*. Let's elaborate, when the robot drives constraints apply due to e.g. the robot being nonholonomic. When the robot pushes an object a different set of constraints are applicable, creating 2 different modes of dynamics. A configuration space containing multiple modes of dynamics is a piecewise-analytic configuration space [29]. Motion planners have great difficulty crossing the boundary from one mode of dynamics to another.




The main challenge is to find, for a given task, a path from a starting configuration of the environment (the point in the joint configuration space that the environment starts in) to a desired target configuration in the joint configuration space, where all specified objects are at their target state as specified in the task. Whilst finding such a path requires searching an enormous joint configuration space that is piecewise analytic. The proposed solution will essentially avoid searching directly in the joint configuration space and will search in subspaces of the joint configuration space where only a single mode of dynamics  is present. Subchallenges that emerge are system identification, control methods, estimating path existence, motion and manipulation planning,  these subchallenges will be properly introduced and elaborated on in chapter 3.

1.3. Report Structure

Create the report structure when the report structure does not change anymore

2



Required Background


 This chapter presents the modules that the hypothesis graph and knowledge graph rely upon. All modules are isolated from one another. As a result, modules can easily be replaced by other modules that perform the same function, such as replace the system identification method with another system identification method. There  is a combination of modules that are non-compatible, which is displayed in table 2.2. As long as required communication between the modules are met, and the modules are compatible the hypothesis graph will try to complete the specified task. The different modules are system identification section 2.1.1, control methods section 2.1.2, path estimation section 2.2, motion planning section 2.3.1, manipulation planning section 2.3.2 and lastly fault detection section 2.4. 

2.1. System Identification and Control Methods

Driving a system toward a desired state is performed by a controller. A system model is required during control design to guarantee stable closed-loop performance. First system models and system identification methods are discussed in the upcoming subsection, and then control methods are discussed.

2.1.1. System Identification

Understanding the world is captured by models, *system models* that estimate the behaviour of real-world systems. For example what trajectory does an object take after receiving a push? The trajectory is dependent on the properties of the object (e.g. mass  inertia), and interaction with its surroundings (e.g. sliding friction). These properties and interactions are captured by a system model. Just as applicable constraints that are captured by a system model, see both robots in figure 1.1, the holonomic point robot in figure 1.1a can drive without constraints, and the boxer robot in figure 1.1b can drive forward, backwards and can rotate. A system model for robot driving for the boxer  robot should thus capture that it is not able to drive directly sideways.

This thesis encounters only 2 different robots but many different objects. Since every object can be different in type, weight and dimensions, system identification  methods are required to capture the variety of objects that the robot can encounter.

Visualise test push sequence to collect IO data

document your Prediction Error Methods (PEM) method

document your Integrated Prediction Error Methods (IPEM) method

2.1.2. Control Methods

Understanding the world is done by a system model, but to perform actions and manipulate objects control is required. A first requirement for a controller is that it should yield a stable closed-loop control, and then additional requirements are desired such as a low prediction error and low final placement error.

MPC Control The MPC controller can be tuned in many ways, for example, to steer the system to the goal aggressively or calmly by adjusting the maximum input and by putting a penalty on rapid input changes. The weights inside its internal objective function can be adjusted during execution time allowing to shift the controller focus from rapid movement to avoiding obstacle regions or converging to a target setpoint. The best feasible input is found every time step by optimising the objective function whilst respecting constraints. Solving an optimisation function to find the best feasible input generally yields robust control. It is however required that the system model is Linear Time-Invariant (LTI). System models for driving the robot can be estimated with LTI models without compromising on model accuracy.

Document the MPC method

MPPI Control A main advantage Model Predictive Path Integral (MPPI) has over MPC control is that it is compatible with nonlinear system models. Whilst drive applications can be accurately estimated by linear models, push applications are harder to estimate with a linear model. Thus MPPI is selected mainly for push applications.

Document the MPPI method

control sys. iden.	PEM	IPEM
MPC	✓	✓
MPPI	✓	✓

Table 2.1: Compatibility between control and system identification methods for drive actions.

control sys. iden.	PEM	Long Short-Term Memory (LSTM)
MPC	✓	✗
MPPI	✓	✓

Table 2.2: Compatibility between control and system identification methods for push actions.

Augment above with all implemented sys identification methods

2.2. Estimating Path Existence

Trying to accomplish an impossible task is a waste of time and resources. For that reason, a path estimation algorithm estimates the existence of a path before a planner (discussed in section 2.3) starts a dedicated search. The path estimation algorithm chosen consists of a graph-based search on a discretised version of the configuration space and can be described as follows.

“The main idea is to discretise the configuration space with a finite discretisation. The emerged cells act as nodes in the graph, cells are connected through edges to nearby cells. Graph-based planners start from the cell containing the starting pose and search for the cell containing the target pose whilst avoiding cells which lie in obstacle space.”

you still have to define the Dijkstra path estimation algorithm

use the words "occupancy map" and "Dijkstra" in the definition, and [32]

visualise the path estimation algorithm

Unfeasible solutions and an undecidable problem The path estimation algorithms does not take system constraints into account. It is thus possible that the path estimation algorithm finds a path from the start to the target configuration, but in reality, the path is unfeasible. An example is driving the boxer robot through a narrow, sharp corner. Whilst theoretically the robot would fit through the corner,

the nonholonomic property of the robot prevents it from steering through such a tight corner. It is for the motion or manipulation planner to detect that the path is unfeasible.

The path estimator suffers from another drawback than potentially providing unfeasible solutions, finding proof that there exists a path that is undecidable. Due to the chosen cell size during discretising the configuration space and the alignment of cells. An example is a quoridor that has exactly the width of the robot. The robot would exactly fit through such a quoridor, but detecting this narrow quoridor requires a number of connected cells that lie exactly in the centre line of the quoridor. Path non-existence on the other hand can be proven easily, because effects such as caging can be detected [4].

Compared to planners which will be discussed in the next section, estimating the existence of a path is fast. Potentially it can even speed up planning because the estimated path can be converted to samples for the motion/manipulation planner, a set of initial samples is referred to as a *warm start*. But let's not get ahead, section 2.3 is dedicated to planners.

Thus in exceptional cases the path estimation algorithm could yield unfeasible paths and could fail to detect existing paths. Generally, it is used to detect if there exists no path from start to target, and to provide a "warm start" to the planner.

2.3. Planning

The planning problem arises when a path from the start to the target configuration for an object must be found. Such a path cannot cross through obstacle space and must be feasible for the system to track. Motion and manipulation planning are split up because there are some major differences. Motion planning only plans for the robot whilst manipulation planning plans for the robot and some objects. Manipulation planning bears more constraints than motion planning because constraints for the robot, the obstacle and constraints between the robot and the obstacle must be respected. A system model discussed in section 2.1.1 acts as a local planner and validates if a path is feasible.

2.3.1. Motion Planning

Controllers discussed in section 2.1 can track a path from start to target. Providing a path is the motion planners' responsibility, motion planners seek inside the configuration space for a path from the start to the target configuration. A practical example of such a path is a list of successive robot poses, from starting pose (coinciding with the starting configuration) toward the target pose, where the successive poses lie close together (reachable for the robot in ~ 20 time samples). Seeking a path from start to target inside a configuration space whilst avoiding obstacles for the robot to track is referred to as *motion planning*. Finding a path between the start and target configuration for pushing applications whilst avoiding collisions is referred to as *manipulation planning*. First, this subsection presents motion planning, and the next subsection section 2.3.2 dedicates itself to manipulation planning. For both motion and manipulation planning and sampling-based methods are used, which can be described as.

"The main idea is to avoid the explicit construction of the object space, and instead conduct a search that probes the configuration space with a sampling scheme. This probing is enabled by a collision detection module, which the motion planning algorithm considers as a "black box." [16]"

Generally, the configuration space motion planners plan consists of 2 subspaces, free and obstacle space. The configuration space in this thesis consists of 4 subspaces, namely free, obstacle, unknown and movable space. To solve motion planning problems for such a configuration space a dedicated motion planning algorithm has been developed that extends the existing algorithm extends the existing double tree optimised version of Rapidly-exploring Random Tree (RRT*) algorithm [4]. The motion planner consists of.

- V : A set of nodes
- E : A set of edges
- P : A set of paths

The start connectivity tree consists of the nodes connected by edges containing the starting node, and vice versa for the target connectivity tree containing the target node. The algorithm grows the two *connectivity trees* by randomly sampling configurations and adding them to the start or target connectivity tree. The algorithm explores configuration space by growing these connectivity trees. When the start connectivity tree meets the target connectivity tree a path from start to target is found.

Newly sampled configurations are added in a structural manner that guarantees an optimal path is found with infinite sampling. Where optimality is defined as the path with the lowest cost. The cost is defined as a sum of a distance metric and a fixed penalty for paths that cross unknown or movable subspaces. Incentivising the algorithm to find a path around unknown or movable obstacles over a path crossing through unknown or movable obstacles.

The algorithm takes in 2 arguments, first the *step size*, the maximal normalised distance between connected samples in the connectivity trees. Second, the *search size* is a subspace around newly sampled samples, inside this subspace a parent node is sought that results in the lowest cost, rewiring of closely nodes happens and the other connectivity tree is searched to detect a full path from start to target.

Now pseudocode of the proposed algorithm is provided in algorithm 1, and functions used are elaborated on in table 2.3. The coloured sections inside algorithm 1 correspond to the surrounding coloured box around subfigures in figure 2.1. The following definitions are used by the proposed algorithm.



x :	A node containing a point in configuration space
x_{init} :	Creates a start and target node
<i>NotReachStop</i> :	True if the stopping criteria is not reached
<i>Sample_{random}</i> :	Creates a random sample in free-, movable- or unknown space
<i>Nearest</i> (x, V):	Returns the nearest nodes from x in V
<i>NearestSet</i> (x, V):	Returns set of nearest nodes from x in V
<i>Project</i> (x, x'):	Project x toward x'
<i>CollisionCheck</i> (x):	Returns true if x is in free-, movable- or unknown space
<i>ObjectCost</i> (x', x):	Returns a fixed additional cost if x enters movable- or unknown space from x' , otherwise returns 0
<i>Distance</i> (x, x'):	Returns the distance between sample x and x'
<i>CostToInit</i> (x):	Find the total cost from x to the initial node
<i>LocalPlannerCheck</i> (x, x'):	Return true if a local planner is able to connect x and x' , otherwise return false. A system model (see section 2.1) acts a local planner
<i>InSameTree</i> (x, x'):	Returns true if both x and x' are in the same tree, otherwise return false

Table 2.3: Functions used by the algorithm 1

Algorithm 1 Pseudocode for modified RRT* algorithm taking movable objects and constraints into account

```

1:  $V \leftarrow x_{init}$ 
2: while NotReachStop do
3:    $Cost_{min} \leftarrow +\infty$  ▷ Create, check and project a new random sample
4:    $x_{rand} \leftarrow Sample_{random}$ 
5:    $x_{nearest} \leftarrow Nearest(x_{rand}, V)$ 
6:    $x_{temp} \leftarrow Project(x_{rand}, x_{nearest})$ 
7:   if CollisionCheck( $x_{temp}$ ) then
8:      $x_{new} = x_{temp}$ 
9:   else
10:    Continue
11:   end if
12:    $X_{near} \leftarrow NearestSet(x_{new}, V)$  ▷ Find and connect new node to parent node
13:   for  $x_{near} \in X_{near}$  do
14:      $Cost_{temp} \leftarrow CostFromInit(x_{near}) + Distance(x_{near}, x_{new}) + ObjectCost(x_{near}, x_{new})$ 
15:     if  $Cost_{temp} < Cost_{min}$  then
16:       if LocalPlannerCheck( $x_{new}, x_{near}$ ) then
17:          $Cost_{min} \leftarrow Cost_{temp}$ 
18:          $x_{minCost} \leftarrow x_{near}$ 
19:       end if
20:     end if
21:   end for
22:   if  $Cost_{min} == \infty$  then
23:     Continue
24:   else
25:      $V.add(x_{new})$ 
26:      $E.add(x_{minCost}, x_{new})$ 
27:   end if
28:    $Cost_{path} \leftarrow +\infty$  ▷ Check if the newly added node can lower cost for nearby nodes
29:   for  $x_{near} \in X_{near}$  do
30:     if InSameTree( $x_{near}, x_{new}$ ) then
31:        $Cost_{temp} \leftarrow CostFromInit(x_{new}) + distance(x_{new}, x_{near}) + ObjectCost(x_{new}, x_{near})$ 
32:       if  $Cost_{temp} < CostFromInit(x_{near})$  then
33:         if LocalPlannerCheck( $x_{new}, x_{near}$ ) then
34:            $E.rewire(x_{near}, x_{new})$ 
35:         end if
36:       end if
37:     else ▷ Add lowest cost path to list of paths
38:        $Cost_{temp} \leftarrow CostFromInit(x_{new}) + distance(x_{new}, x_{near})$ 
39:        $\quad + CostFromInit(x_{near}) + ObjectCost(x_{new}, x_{near})$ 
40:       if  $Cost_{temp} < Cost_{path}$  then
41:         if LocalPlannerCheck( $x_{new}, x_{near}$ ) then
42:            $Cost_{pathMin} \leftarrow Cost_{temp}$ 
43:            $x_{pathMin} \leftarrow x_{near}$ 
44:         end if
45:       end if
46:     end if
47:   end for
48:   if  $Cost_{pathMin} == \infty$  then
49:     Continue
50:   else
51:      $P.addPath(x_{new}, x_{pathMin}, Cost_{pathMin})$ 
52:   end if
53: end while

```

Now an example is provided of the proposed algorithm that creates and adds one sample in figure 2.1. After adding the newly sampled sample is added to the start connectivity tree, a sample is rewired then the target connectivity tree is connected to the start connectivity tree. The resulting path found can be visualised in section 2.3.1.



Figure 2.1: Visualisation of the Double tree RRT* motion planner that adds a single sample to the connectivity graph. The colour of the box surrounding subfigures corresponds to the coloured sections in algorithm 1. 3-dimensional configuration space displayed as 2-dimensional configuration space (x and y are visible, θ is not visible).

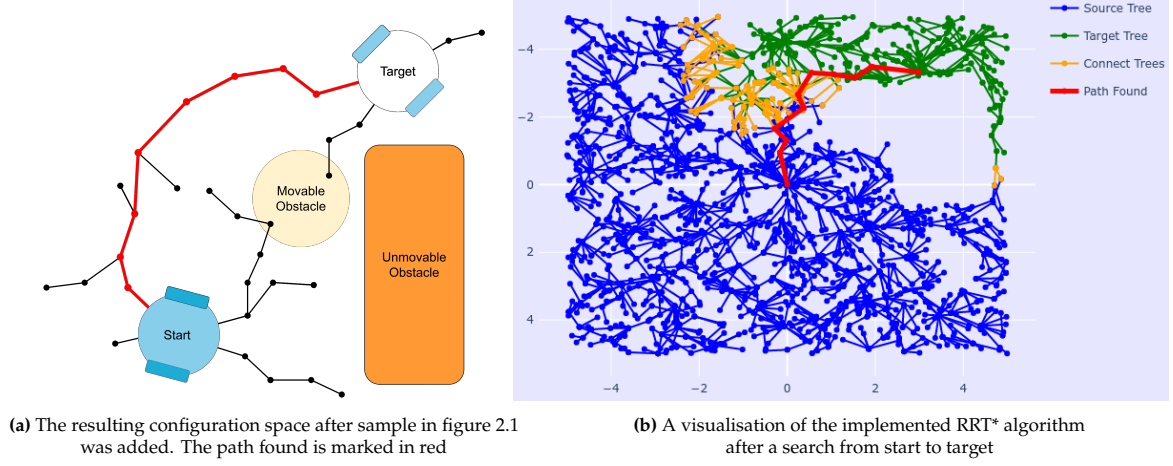


Figure 2.2: Comparing schematic example to a visualisation of the real algorithm.

The result of adding an extra penalty for crossing unknown or movable subspace is that such subspaces are avoided if possible. If it is not possible to find a valid path, then movable or unknown subspaces are crossed, displayed in figure 2.3. A path cannot be tracked by a controller if it crosses movable or unknown spaces, first, the object must be moved, and then the original path can be tracked. In figure 2.3 it can be seen that the motion planner cannot find a path around the movable object and is forced to add the cost to move the object. The added fixed cost for a path crossing through a movable or unknown object motivates the motion planner to find the shortest path around objects but prefers moving an object over making a large detour.

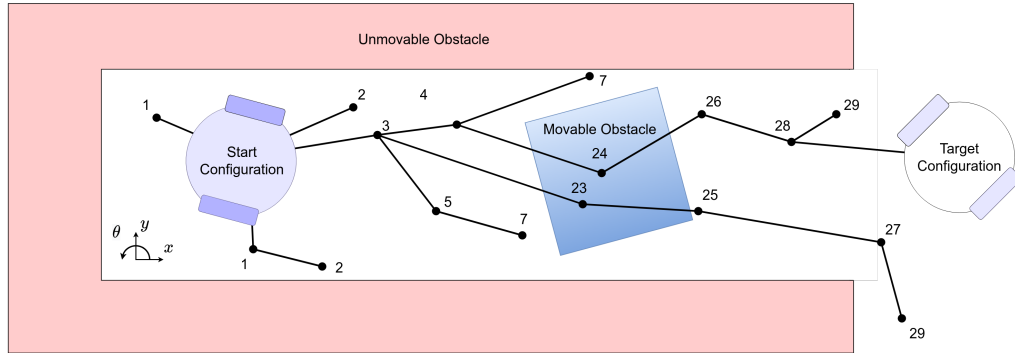


Figure 2.3: Schematic view of the proposed double RRT* tree taking movable and unknown objects into account with cost to reach a sampled configuration displayed.

The proposed motion planning algorithm searches the configuration space from the start connectivity tree and the target connectivity tree. Exploring faster compared to the single tree RRT* algorithm. The proposed algorithm rewires nodes, resulting in lowering cost for existing paths. The proposed algorithm finds the optimal lowest-cost path with infinite sampling because of its ability to rewire nodes. The *LocalPlannerCheck* provides feasible paths, such that the proposed algorithm yields paths that respect the system constraints. After all later on the system will be controlled to track the path. Now motion planning is discussed, manipulation will be discussed.

2.3.2. Manipulation Planning

- explain the manipulation planning extensions with a figure, no one is going to understand you otherwise.
- create manipulation planning subsection after implementation, code first, then write. Because writing is harder than coding

2.4. Fault Detection

During execution time, the hypothesis graph (hgraph) is unable to perform any action, such as motion planning or finding new hypotheses. This blocking behaviour has some implications, especially when an edge steers the system toward a target state and is blocked, for example, the controller steers the system to a target state, and meets an unmovable object. In such cases, the controller will never reach the target state and the system remains in a local minimum forever. Giving complete control to an edge is thus not desired because of the blocking behaviour of edges. More examples can be given such as system identification or push manipulation on an object that is cornered can be provided. However, instead of thinking of all possible options for how the robot could get stuck a more automated approach is sought. A central coordinator who can step in and terminate the edge is desired. Detecting controller faults is a large robotic topic [13] properly implementing a fault detection and diagnosis module is out of the scope of this thesis. Instead, the two simple metrics will be monitored during execution. Prediction Error (PE) where the predicted position is compared with the measured position of the system, and Tracking Error (TE) where the path provided for the robot to track is compared with the trajectory that the robot made during tracking such a path. Definitions of PE and TE are now given, in table 2.4 insight is provided why a metric would catch certain faulty behaviour.

- define PE
- define TE

2.4.1. Monitoring Metrics

Prediction Error (PE)	During executing a sudden high PE indicates unexpected behaviour occurs, such as when the robot has driven into an object which is was not expecting. A high PE, which persists indicates that the robot is continuously blocked. Single collisions are allowed, but when the PE exceeds a pre-defined threshold and persists over a pre-defined time, the hgraph concludes that there was an error during execution and the edge failed.
Tracking Error (TE)	The system should not diverge too far from to path it is supposed to track, if the robot diverges more than a pre-defined threshold the hgraph concludes that there was an error during execution and the edge fails.

Table 2.4: Monitor metrics used to monitor if a fault occurred during execution of an edge

3

The Hypothesis and Knowledge graph

This chapter is dedicated to introducing and defining the proposed method. The **hypothesis graph**, defined in section 3.1.1 is responsible for searching the joint configuration space for action sequences that complete a given task. The hgraph additionally is responsible for collecting new knowledge of the environment. The **knowledge graph**, defined in section 3.2.1 is responsible for storing collected knowledge. Collected knowledge is rated using various metrics to create an ordering within collected knowledge. The knowledge graph (kgraph) is queried for action suggestions when a new action sequence is generated by the kgraph. Figure 3.1 presents a schematic overview of the interconnection of the knowledge-, hypothesis graph and the robot environment. A more in depth analysis of the hypothesis graph is given in section 3.1, and for the knowledge graph in section 3.2.

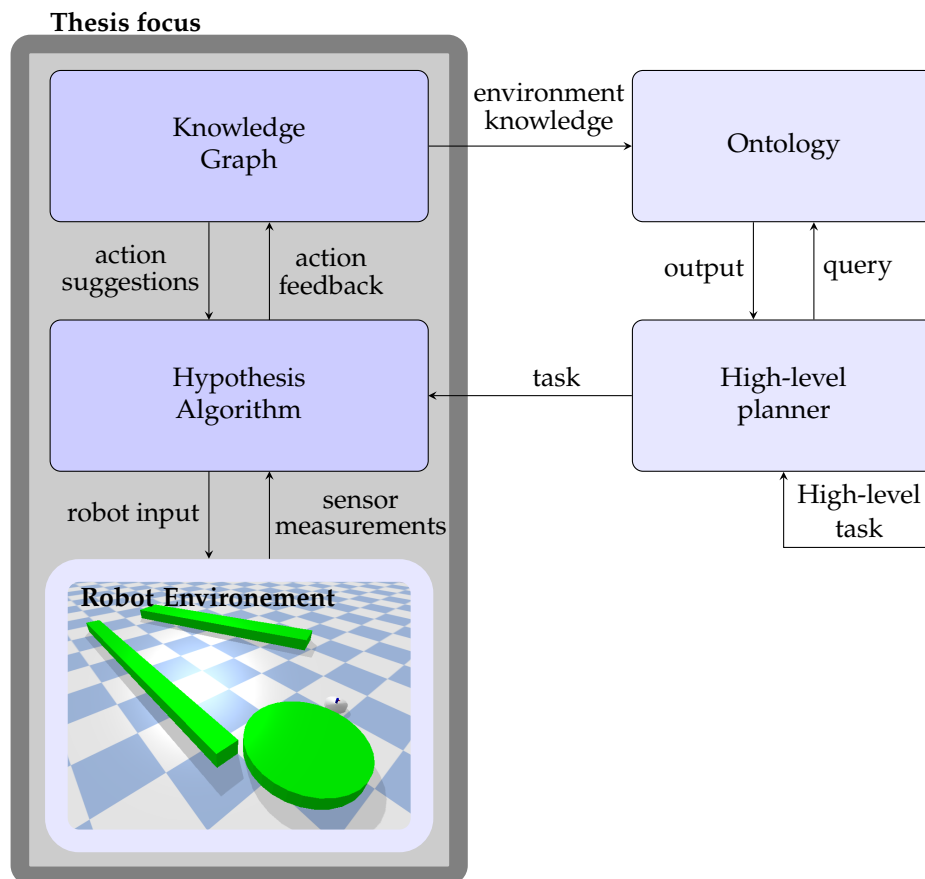


Figure 3.1: Simplified flowchart representation of the proposed method. The thesis focus could be augmented with an ontology and high-level planner as displayed. Such an augmentation would create a framework capable of completing high-level tasks such as cleaning or exploring.

a task consists of objects with corresponding target states. A summary of the terminology (e.g. task) used is presented and elaborated on in table 3.1. This chapter uses some terms that might confuse the reader, the following table conveniently groups the terminology used.

Task:	Tuple of objects and target states. $\text{task} = \langle \text{Obst}_{\text{task}}, S_{\text{targets}} \rangle$
Object Class	Classification assigned to an object, initially all objects are placed in the <i>Unknown</i> class, and objects class can be updated to either the <i>Obstacle</i> or the <i>Movable</i> class $\text{object class} = (\text{Unknown}, \text{Obstacle}, \text{Movable})$
Subtask:	A single object, and a single target state. $\text{subtask} = \langle \text{obst}_{\text{subtask}}, S_{\text{target}} \rangle$
Node:	Tuple of an status, object and a state. A node in the hgraph, represents an object being at a state in the robot environment. Status of a node can be either <i>Initialised</i> , <i>Completed</i> or <i>Failed</i> . $\text{node} = \langle \text{status}, \text{Obst}, s \rangle$
Edge:	Edge from a node to another node in the hgraph or kgraph, represent movement by the robot in the robot environment. $\text{edge} = \langle \text{status}, id_{\text{from}}, id_{\text{to}}, \text{verb}, \text{controller}, \text{dynamic model}, \text{path} \rangle$ where status is the status of the edge elaborated on in figure 3.2, see section 3.1.1 for information on the edge arguments.
Hypothesis:	Sequence of successive edges in the hgraph, an idea to put a object at it's target state. If executed in order without any error, completes a subtask. $\text{hyp} = \langle \text{edge}_1, \text{edge}_2, \text{edge}_3, \dots, \text{edge}_m \rangle$
Hypothesis Graph:	Collection of nodes and edges. A path of edges from a starting node to a target node is an hypothesis. If every subtask has an succesfully completed accompanying hypothesis, the hgraph completed the task $\text{HGraph} = \langle V, E \rangle$
Knowledge Graph:	Collection of nodes and edges. If an edge stoppes executing, feedback on the performance of the edge is stored in the kgraph $\text{KGraph} = \langle V, E \rangle$

Table 3.1: Terminology of terms used

3.1. Hypothesis Graph

The hgraph is responsible for generating action sequences, called hypothesis (consisting of a list of successive edges from start to target node in the hgraph) that complete a single subtask. When all subtasks in a task are completed, the hgraph halts and concludes the task successfully completed. A search in the joint configuration space is avoided because an edge only operates in a single mode of dynamics, such as driving or pushing. When an object cannot directly be steered toward its target location new nodes are generated which need to be completed before the original object can be steered toward its target location. An example of when an object cannot directly be steered toward its target state is because the path is blocked by another object. A hypothesis, consisting of a list of edges that represent actions in the robot environment might succeed or fail. Figure 3.5 displays a flowchart explaining how new nodes and edges are generated in the hgraph. Succesfully completed edges eventually result in

completed subtasks, failed edges trigger replanning that will restart the search to a hypothesis.

Section 3.1.1 defines the hgraph and components, then section 3.1.2 elaborates how the hgraph searches for a solution in the joint configuration space. The section is concluded with an extensive example.

3.1.1. Definition

Before defining the hgraph, some definitions are defined on which the hgraph depends. First, recall the **state** defined in the section 1.2.

An object holds the information about an object.

Formally, a **object**, $obst_{id}(k) = \langle s(k), shape \rangle$

where $shape$ is linked to a 3D representation of the object which is used to construct the configuration space.

An object node represents an object in a state.

Formally, a **objectNode**, $V_{id}^{obst} = \langle status, obst(k) \rangle$

where status indicates if the node has been visited in the hgraph. $status = (Initialised, Completed, Failed)$

An edge describes the details of how a node transitions to another node in the hgraph. In the robot environment, an edge represents a change of state for an object. System identification and performing an action such as pushing or driving both change the state of objects in the robot environment, but because are very different, the edges are split into 2 categories. IdentificationEdges that collect system Input-Output (IO) data and convert that into a system model. And actionEdges that plan and track a motion from a start to a target state. Formally:

A **identificationEdge**,

define identificationEdge, currently hard coded models are used in the implementation

A **actionEdge**, $\tau_{(from,to)} = \langle status, id_{from}, id_{to}, verb, controller, dynamic\ model, path \rangle$

with id_{from} and id_{to} indicating the node id of the node in the hgraph where the edge start from and point to respectively, $verb$ an English verb describing the action the edge represents, the controller contains the controller used for driving the robot, the dynamic model is the dynamic model used by the controller, path a list of configurations indicating the path connecting a start- to target node.

A $verb = \{driving, pushing\}$.

Now the nodes and edges have been defined, the hgraph can be defined.

Formally, a **hypothesis graph**, $G^{hypothesis} = \langle V, E \rangle$
comprising $V = \{V_i^{ob}\}$, $E \in \{\tau_{(i,j)} | V_i, V_j \in \{V^{ob}\}, i \neq j\}$.

Most hgraph components have now been defined. The status of an identification edge or action edge still remains undefined and requires some further explanation.

Status, Types and Lifetime of edges Because system identification and tracking a path are so very different, the edges are split into two categories, identification edges and action edges. An identification edge, which is responsible for sending an input sequence to the system and recording the system output. That input/output sequence and assumptions on the system are the basis for system identification, techniques on various system identification methods are discussed in section 2.1. The goal is to create a dynamical model which is augmented with a corresponding controller is closed-loop stable.

An identificationEdge, the status can be visualised in figure 3.2.

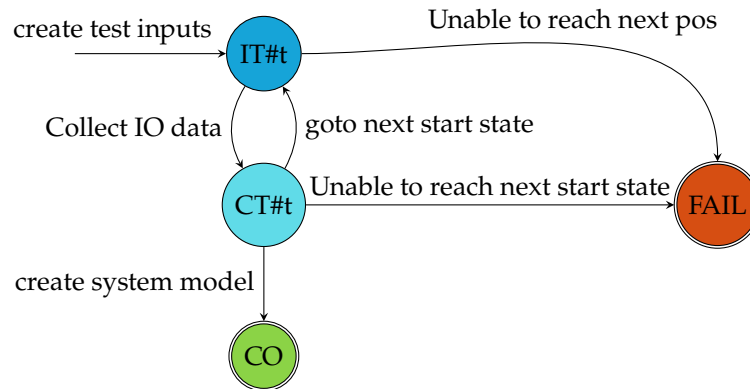


Figure 3.2: FSM displaying the state of an identification edge

some explainer on this status of iden edge

The second type of edge is an actionEdge, containing a drive or push action. An actionEdge ready for execution contains all the necessary information to send input to the robot resulting in an object being steered toward it's target state. Before an edge is ready for execution it should be initialised properly, more specifically: initialised, path estimated should be performed, a system model must be initiated and path planning must be performed. Then finally the edge is ready to be executed and send input toward the robot, an Finite State Machine (FSM) of the actionEdge's status can be visualised in figure 3.3.

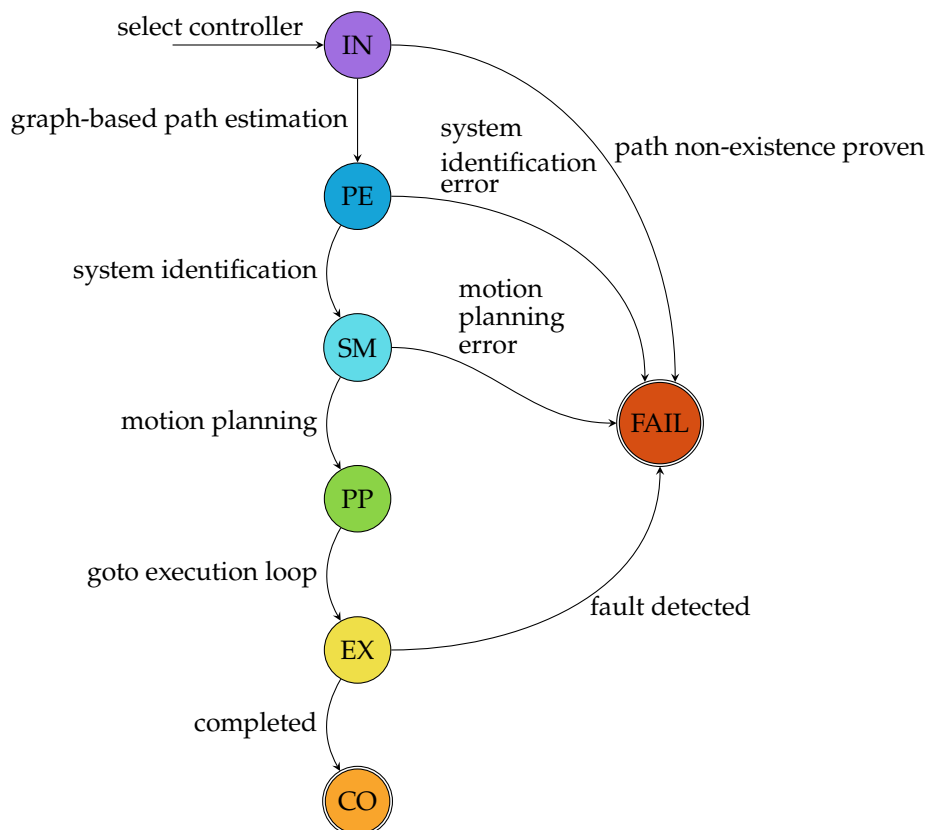


Figure 3.3: FSM displaying the state of an action edge

- INITIALISED (IN)** The edge is created with a source and target node which are present in the hgraph. A choice of controller is made.
- PATH EXISTS (PE)** A graph-based search is performed to validate if the target state is reachable assuming that the system is holonomic.
- SYSTEM MODEL (SM)** A dynamics system model is provided to the controller residing in the edge.
- PATH PLANNED (PP)** Resulting from a sample-based planner, a path from start to target state is provided.
- EXECUTING (EX)** The edge is currently receiving observations from the robot environment and sends back robot input.
- COMPLETED (COMPL)** The edge has driven the system toward its target state and its performance has been calculated.
- FAILED (FAIL)** An error occurred, yielding the edge unusable.

Figure 3.3 shows that many steps must successfully be completed before the robot can start executing. The performance of an edge during execution, measured in various metrics (section 4.1 is dedicated to metrics) is dependent on many aspects. Such as the choice of controller, the path estimation, the system model yielded by the identification edge and the path yielded by motion planning. Now that the hgraph is defined, let's see how it is generated in the upcoming section.

3.1.2. The Search and the Execution loop

Now that hgraph is defined let's find out how it reacts to various behaviour. The reaction of the hgraph can be visualized in a flowchart of the hgraph provided in figure 3.5. In figure 3.5 two loops can be identified, clarified in Figure 3.4. In the search, loop hypotheses are formed and the execution loop tests the hypothesis.

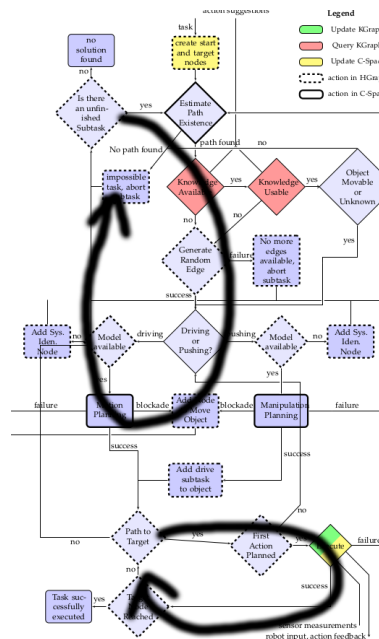


Figure 3.4: The search and execution loop.

Update figure above, professionalise arrows

While the hgraph resides in the search loop, hypotheses are formed. Forming a hypothesis, generating edges, nodes and progressing their status as described in figures 3.2 and 3.3 respectively. The hgraph operates synchronously, thus at any point in time, the hgraph resides in a single block within figure 3.5. The result is that the robots cannot operate whilst the hgraph resides in the search loop, and

during execution, no hypothesis can be formed or updated. Assumption 1.2.2 guarantees that the robot environment does not change causing existing hypotheses to be outdated.

While the hgraph resides in the execution loop, *an edge is being executed*. The phrase “an edge is being executed” seems odd at first because every aspect of closed-loop control resides in an edge, edges can be executed. Recall that nodes in the hgraph represent objects in a state, and edges represent actions. Now a flowchart of the behaviour of the hgraph is presented.

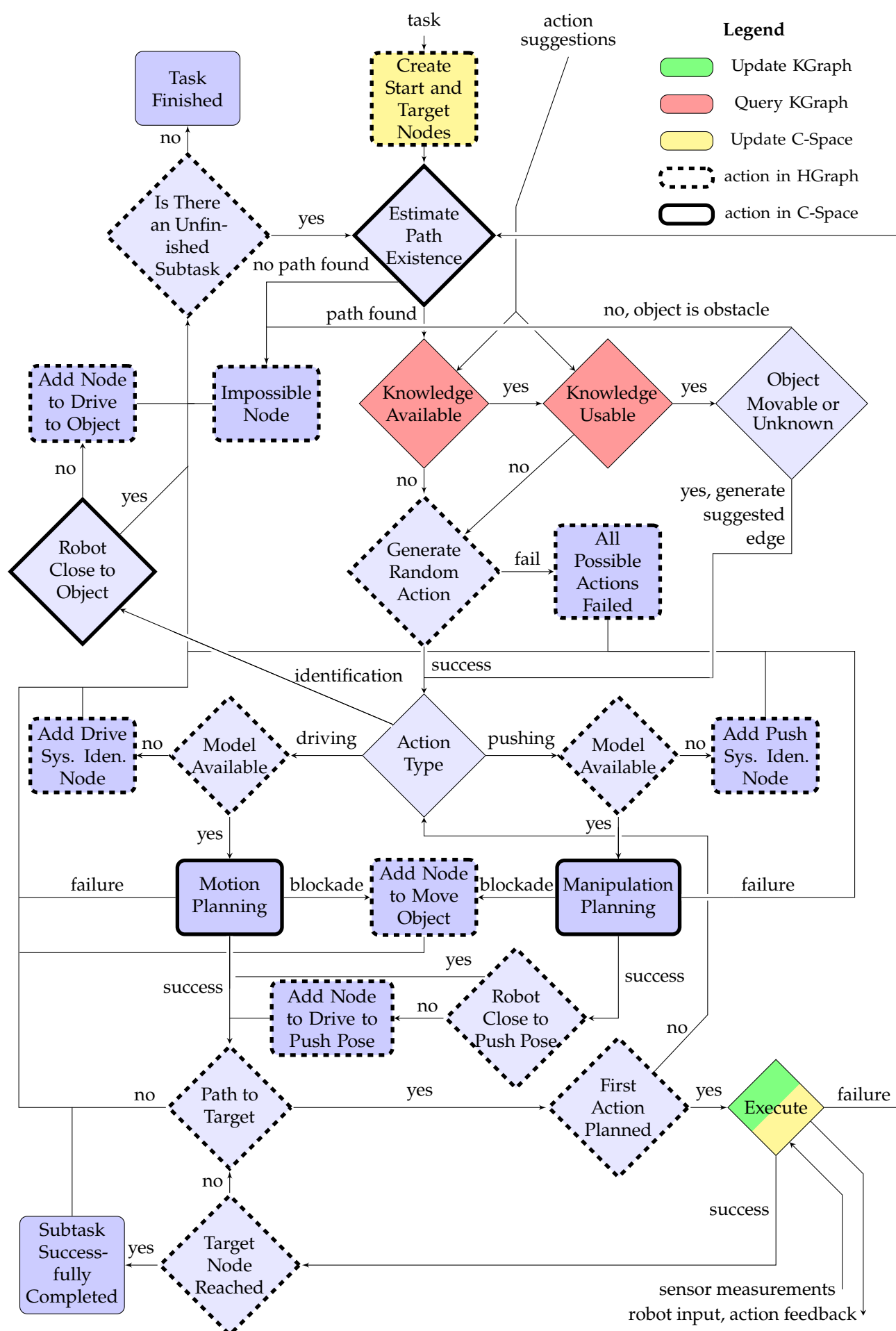


Figure 3.5: Flowchart displaying the hypothesis graph’s workflow.

Table 3.2 provides important information that does not fit on the flowchart above.

Node name	Description of actions taken
task finished	Calculate and log all metrics for every hypothesis in the hgraph.
create start and target nodes	generate a start and target node for every subtask in the task.
is there an unfinished subtask	
estimate path existence	
add node to drive to object	
impossible node	
knowledge available	
knowledge usable	
object movable or unknown	
robot close to object	
generate random action	
all possible actions failed	
add drive system	
identification node	
model available	
action type	
model available	
add push system identification node	
motion planning	
add node to drive to push pose	
robot close to push pose	
path to target	
first action planned	
execute	
subtask succesfully completed	
target node reached	

Table 3.2: Functions used by the algorithm 1

add RHS' in the table above

After initialising the hgraph with a task, there is only a single access point toward the hgraph. A function *respond(observation)* that returns control input toward the robot. As argument function *respond(observation)* takes an observation of the environment, such an observation contains measurements of every object's latest measured state. Recall that the perfect-sensor assumption, assumption 1.2.2 provides access to the exact states of every object.

it is not clear to the reader that the text above only focusses on a single subtask AKA no global optimum for the task is sought

Visualise this in hgraph: adding an edge for a drive action including sys iden. use "backward search"

A hypothesis is formed when the robot node is connected through a set of successive edges to the subtasks target node.

visualise this in hgraph: Executing a drive action

Visualise this: adding an edge for a push action including sys iden.

visualise this in hgraph: Executing a push action

emphasise recursiveness of the hgraph, thus want to move box A, but B is in the way, move box B first, but box C is in the way, move box C first becomes: move(C) -> move(B) -> move(A)

introduce the blacklist: a set of edges that cannot be added to a node because they failed in the past. The blacklist prevents regenerating edges which already failed in the past

section about fault/monitoring metrics, that can halt and fail an executing edge

What by now hopefully became clear to the reader is that the hgraph autonomously searches for hypotheses to solve the task, one subtask at a time. The hgraph switches between the search and execution loop. Switching from the search loop toward the execution loop when a hypothesis is found, and switching back when a hypothesis is completed or an action failed to complete.

The limited number of possible edges (every combination of a system identification method with a compatible control method) guarantees that the robot tries to connect 2 nodes, but concludes that it cannot reach a node if all possible edges have failed. Eventually running out of nodes to connect and conclude that a subtask cannot be completed.

In the next section, the edges that are executed will be reviewed and stored in a knowledge base. The knowledge base will suggest edges when faced with similar nodes to connect.

3.1.3. Example

here are some example hgraph's required

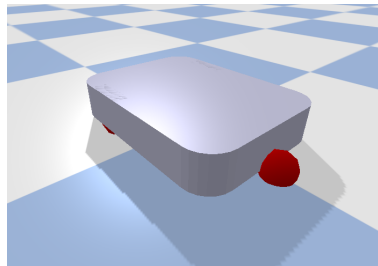


Figure 3.6: Pushing task through a blocked corridor with the point robot, a green cube to push toward the target ghost state and a red blockade.

3.2. Knowledge Graph

this should get a intro, what does that kgraph do

3.2.1. Definition

instinctively explain what the heck the kgraph does

collect feedback -> orders feedback per obstacle -> recommends controller/dyn model

why are there 2 types of nodes? explain why a changeOfStateNode here is required

3.2.2. Example

3.2.3. Edge Metrics

The kgraph keeps an ordered list of ‘good’ and ‘bad’ edge arguments (controller and system model). ‘Good’ and ‘bad’ are defined by edge metrics, these metrics are created after the completion of an edge, regardless of whether the edge was successfully completed or failed. An indication is given on why certain metrics matter in table 3.3.

Prediction Error (PE)	To better compare prediction errors the PE is summarised and average PE. The average PE is an indicator of an accurate system model but can give misleading results since PE is also an indicator of unexpected collisions. Prediction error should thus only be used if there are no collisions detected. The average PE comes with more flaws since the average is mostly determined by outliers, some unfortunate outliers in the PE might for the largest part determine the average PE. The average PE will thus not be used because it is not robust enough.
Tracking Error (TE)	For a low TE the system model must be close to the real motion equations to yield a feasible path, the controller must be well tuned to be able to track that path and the controller and system model must be in collaboration, because the controller uses the system model to calculate system input. A low TE tells multiple things, whilst a high TE would indicate improvements could be gained in the controller, the system model or their collaboration.
ratio #successfully completed edges and #total edges	Over time the kgraph can recommend the same edge arguments multiple times. Logging the ratio of succeeding edges vs total edges builds an evident portfolio. Still, this metric has to be taken with a grain of salt because edges with equal edge arguments perform similar actions e.g. pushing an object through a wide corridor is compared to pushing the same object through a narrow corridor. One could say “comparing apples with pears”.
the final position and displacement error	The quality of the result is measured in the final position and displacement error. The importance should thus be stressed when ordering edge arguments.
planning time	With system identification, path estimation, motion or manipulation planning the planning time can vary in orders of magnitude between simple or more complex approaches. Planning time mainly serves to rank the slowest planners low, whilst not influencing the rank of fast and average planners.
runtime	Also known as execution time would be a quality indicator if start and target states would be equal. Edges are recommended to solve similar tasks, where the path length between the start and target state is different. Thus planning time is not of any use to rank edges.
completion time = runtime + planning time	With the same argumentation as runtime, completion time is not of any use to rank edges.

Table 3.3: Edge metrics used to rank control methods from ‘good’ to ‘bad’

4

Results

This chapter does this or that todo A.

The Simulation Environment Testing in a simulation environment has been done using the URDF Gym Environment [25], a 100% python environment build upon the PyBullet library [6]. The code created during the thesis can be found on GitLab and GitHub. Experiments ran on standard TU Delft laptop: HP ZBook Studio x360 G5, running OS: Ubuntu 22.04.1 LTS x86_64, CPU: Intel i7-8750H (12) @ 4.100GHz, GPU: NVIDIA Quadro P1000 Mobile. The simulation environment provides many different

robots, 2 simple robots are selected to perform tests, they are displayed in figure 1.1, and various objects are displayed in figure 1.2.

The results will be made in a while, wait for a moment while we try to make sense of the almost-existing results

4.1. Proposed Method Metrics

Most interesting is the progression of metrics over time. It is expected that the effect of learning can be measured by investigating various metrics and tracking their development over time. Furthermore, metrics will be used to compare the proposed method to relative state-of-the-art papers. All metrics including argumentation of why the metric is relevant are presented in table 4.1.

Corrado: # is that nicely replacing number of?

Total Average Prediction Error	The total average PE is created by averaging over every hypothesis' average PE in a hgraph. Since the PE is high when unexpected behaviour occurs, seeing the total average PE lower would indicate the robot encounters less unexpected behaviour, indicating the robot is learning.
Total Average Tracking Error	The total average TE is created by averaging over every hypothesis' average TE in a hgraph. Seeing the total average TE lower over time would indicate the robot is selecting better suitable controllers and system models, indicating the robot is learning.
Final positions and displacement errors	The final position and displacement error is a metric which how a controller performs. This thesis does not create or investigate controllers, but it is interesting to see why different controllers are preferred for different objects. The final position and displacement error could be the cause.
The ratio between #hypothesis and #tasks	Expected is that whilst learning system models, the hypothesis created will be more effective. Thus the ratio between the total number of hypotheses and the total number of tasks is expected to lower with new knowledge.
The ratio between #successfull and #total edges in kgraph	When the kgraph improves recommending a controller and system model, the ratio between successful edges and total edges is expected to increase because, with better recommendations, more edges will be succesfully completed.
task completion time = runtime + planning time	If equal tasks are given multiple times, the total task completion time should drop pretty drastically. Multiple factors help to lower the task completion time, firstly system identification has to be performed only once, and there is no need to lose time on redoing system identification. Secondly, the hgraph is expected to improve generated hypothesis, or better said, the same mistake should not be made multiple times, resulting in fewer failing hypotheses and lowering task completion time.

Table 4.1: Proposed method metrics used to compare the proposed method with the state-of-the art

4.2. Benchmark Tests

4.3. Comparison with related papers

The papers to compare with:

A Model Predictive Approach for Online Mobile Manipulation of Non-Holonomic Objects Using Learned Dynamics

[22],

See the paper for tests to reproduce with my hgraph

Dynamic Model Learning and Manipulation Planning for Objects in Hospitals Using a Patient Assistant Mobile (PAM) Robot

[18] The following [18] citation is here because it is a referred to from [22] many times

4.4. Randomisation

4.5. Knowledge Graph On/Off

4.6. Discussion

5

Conclusions

This chapter todo A.

Recall the main research question.

“How do objects’ system models learn by a nonprehensile manipulation robot during task execution improve global task planning?”

Before answering the main research question, the two research subquestions are answered.

Recall first research subquestion:

“How does a backward search [15] while remembering interactions with unknown objects compare to not remembering learning interactions over time?”

The backwards search technique used by the hypothesis graph has been shown to successfully handle uncertain action sequences toward task completion. Replanning occurs when edges fail to complete. To obtain a clear overview of the effect of the learned system model, the effectiveness of the hgraph has been monitored over time in various metrics. Two cases have been researched, task execution whilst remembering interaction in the knowledge graph versus task execution without a knowledge graph. The main difference was the time difference because the system identification procedure was executed for every edge in the no KGraph case. Another difference was spotted, over time the KGraph case generates hypotheses which completed more often compared to the no KGraph case.

Recall the second research subquestion:

“Can the proposed method combine learning and planning for push en drive applications? Can the proposed method complete tasks, and how does it compare against the state of the art?”

The proposed hypothesis graph with the associated knowledge graph was able to perform as well or better compared to teh state of the art. The proposed method was tested against existing methods which only a subset of tasks, or 2 of the 3 topics. In comparison with with[22] the proposed method was shown to perform similarly in prediction errors, and final displacement error. The proposed method is not designed to optimise a global plan, thus unnecessary driving and pushing can occur.

anser the main research question

Now the main question can be answered...

5.1. Drawbacks

[29] optimises a global path (let’s say shortest path to complete a task), the proposed method takes a random subtask which does not even try to converge toward a global optimal path.

5.2. Future work

how could the closed-world assumption be removed to resample the real world more?

how could the perfect object sensor assumption be removed to resample the real world more?

how could the task as commutative assumption be removed to resample the real world more?

how could the objects do not tip over assumption be removed to resample the real world more?

The hgraph could think of new hypotheses during execution time if it would operate async, which would greatly improve it.

list metric

find the metrics used in relevant comparable papers

create simple sections + bullet lists what can be in them

create expected results

Glossary

List of Acronyms

MPPI Model Predictive Path Integral	9
MPC Model Predictive Control	2
NAMO Navigation Among Movable Objects	2
NP-hard non-deterministic polynomial-time hard	6
NP non-deterministic polynomial-time	6
FSM Finite State Machine	19
hgraph hypothesis graph	15
kgraph knowledge graph	16
IO Input-Output	18
PE Prediction Error	15
TE Tracking Error	15
RRT* optimised version of Rapidly-exploring Random Tree	10
LTI Linear Time-Invariant	9
LSTM Long Short-Term Memory	9
PEM Prediction Error Methods	8
IPEM Integrated Prediction Error Methods	8

References

- [1] Ermanno Arruda et al. “Uncertainty Averse Pushing with Model Predictive Path Integral Control”. In: *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. 2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids). Birmingham: IEEE, Nov. 2017, pp. 497–502. ISBN: 978-1-5386-4678-6. DOI: 10.1109/HUMANOIDS.2017.8246918. URL: <http://ieeexplore.ieee.org/document/8246918/> (visited on 04/29/2022).
- [2] Maria Bauza, Francois R. Hogan, and Alberto Rodriguez. “A Data-Efficient Approach to Precise and Controlled Pushing”. Oct. 9, 2018. arXiv: 1807.09904 [cs]. URL: <http://arxiv.org/abs/1807.09904> (visited on 03/01/2022).
- [3] Dmitry Berenson et al. “Manipulation Planning on Constraint Manifolds”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009 IEEE International Conference on Robotics and Automation (ICRA). Kobe: IEEE, May 2009, pp. 625–632. ISBN: 978-1-4244-2788-8. DOI: 10.1109/ROBOT.2009.5152399. URL: <http://ieeexplore.ieee.org/document/5152399/> (visited on 12/05/2022).
- [4] Long Chen et al. “A Fast and Efficient Double-Tree RRT*-Like Sampling-Based Planner Applying on Mobile Robotic Systems”. In: *IEEE/ASME Transactions on Mechatronics* 23.6 (Dec. 2018), pp. 2568–2578. ISSN: 1083-4435, 1941-014X. DOI: 10.1109/TMECH.2018.2821767. URL: <https://ieeexplore.ieee.org/document/8329210/> (visited on 04/14/2022).
- [5] Lin Cong et al. “Self-Adapting Recurrent Models for Object Pushing from Learning in Simulation”. July 27, 2020. arXiv: 2007.13421 [cs]. URL: <http://arxiv.org/abs/2007.13421> (visited on 04/06/2022).
- [6] Erwin Coumans and Yunfei Bai. *PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning*. 2016–2021. URL: <http://pybullet.org>.
- [7] Mohamed Elbanhawi and Milan Simic. “Sampling-Based Robot Motion Planning: A Review”. In: *IEEE Access* 2 (2014), pp. 56–77. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2014.2302442. URL: <https://ieeexplore.ieee.org/document/6722915/> (visited on 11/30/2022).
- [8] Kirsty Ellis et al. “Navigation among Movable Obstacles with Object Localization Using Photorealistic Simulation”. In: July 2022. URL: https://www.researchgate.net/publication/362092621_Navigation_Among_Movable_Obstacles_with_Object_Localization_using_Photorealistic_Simulation.
- [9] Kris Hauser and Jean-Claude Latombe. “Multi-Modal Motion Planning in Non-expansive Spaces”. In: *The International Journal of Robotics Research* 29.7 (June 2010), pp. 897–915. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364909352098. URL: <http://journals.sagepub.com/doi/10.1177/0278364909352098> (visited on 12/05/2022).
- [10] Léonard Jaillet and Josep M. Porta. “Path Planning Under Kinematic Constraints by Rapidly Exploring Manifolds”. In: *IEEE Transactions on Robotics* 29.1 (Feb. 2013), pp. 105–117. ISSN: 1552-3098, 1941-0468. DOI: 10.1109/TR0.2012.2222272. URL: <http://ieeexplore.ieee.org/document/6352929/> (visited on 12/05/2022).
- [11] Leslie Pack Kaelbling and Tomas Lozano-Perez. “Hierarchical Task and Motion Planning in the Now”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011 IEEE International Conference on Robotics and Automation (ICRA). Shanghai, China: IEEE, May 2011, pp. 1470–1477. ISBN: 978-1-61284-386-5. DOI: 10.1109/ICRA.2011.5980391. URL: <http://ieeexplore.ieee.org/document/5980391/> (visited on 12/05/2022).
- [12] Sertac Karaman and Emilio Frazzoli. “Sampling-Based Algorithms for Optimal Motion Planning”. In: *The International Journal of Robotics Research* 30.7 (June 2011), pp. 846–894. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364911406761. URL: <http://journals.sagepub.com/doi/10.1177/0278364911406761> (visited on 03/15/2022).

- [13] Eliahu Khalastchi and Meir Kalech. "On Fault Detection and Diagnosis in Robotic Systems". In: *ACM Computing Surveys* 51.1 (Jan. 31, 2019), pp. 1–24. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3146389. URL: <https://dl.acm.org/doi/10.1145/3146389> (visited on 12/13/2022).
- [14] Zachary Kingston, Mark Moll, and Lydia E. Kavraki. "Sampling-Based Methods for Motion Planning with Constraints". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1.1 (May 28, 2018), pp. 159–185. ISSN: 2573-5144, 2573-5144. DOI: 10.1146/annurev-control-060117-105226. URL: <https://www.annualreviews.org/doi/10.1146/annurev-control-060117-105226> (visited on 05/06/2022).
- [15] Athanasios Krontiris and Kostas Bekris. "Dealing with Difficult Instances of Object Rearrangement". In: July 2015. DOI: 10.15607/RSS.2015.XI.045.
- [16] Steven Michael LaValle. *Planning Algorithms*. Cambridge ; New York: Cambridge University Press, 2006. 826 pp. ISBN: 978-0-521-86205-9.
- [17] Tekin Meriçli, Manuela Veloso, and H. Levent Akin. "Push-Manipulation of Complex Passive Mobile Objects Using Experimentally Acquired Motion Models". In: *Autonomous Robots* 38.3 (Mar. 2015), pp. 317–329. ISSN: 0929-5593, 1573-7527. DOI: 10.1007/s10514-014-9414-z. URL: <http://link.springer.com/10.1007/s10514-014-9414-z> (visited on 01/31/2022).
- [18] Roya Sabbagh Novin et al. "Dynamic Model Learning and Manipulation Planning for Objects in Hospitals Using a Patient Assistant Mobile (PAM) Robot". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Madrid: IEEE, Oct. 2018, pp. 1–7. ISBN: 978-1-5386-8094-0. DOI: 10.1109/IROS.2018.8593989. URL: <https://ieeexplore.ieee.org/document/8593989/> (visited on 05/05/2022).
- [19] Manoj Pokharel. "Computational Complexity Theory(P,NP,NP-Complete and NP-Hard Problems)". In: (June 2020).
- [20] John Reif and Micha Sharir. "Motion Planning in the Presence of Moving Obstacles". In: *26th Annual Symposium on Foundations of Computer Science (Sfcs 1985)*. 26th Annual Symposium on Foundations of Computer Science (Sfcs 1985). Portland, OR, USA: IEEE, 1985, pp. 144–154. ISBN: 978-0-8186-0644-1. DOI: 10.1109/SFCS.1985.36. URL: <http://ieeexplore.ieee.org/document/4568138/> (visited on 05/05/2022).
- [21] Roya Sabbagh Novin, Mehdi Tale Masouleh, and Mojtaba Yazdani. "Optimal Motion Planning of Redundant Planar Serial Robots Using a Synergy-Based Approach of Convex Optimization, Disjunctive Programming and Receding Horizon". In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 230.3 (Mar. 2016), pp. 211–221. ISSN: 0959-6518, 2041-3041. DOI: 10.1177/0959651815617883. URL: <http://journals.sagepub.com/doi/10.1177/0959651815617883> (visited on 05/05/2022).
- [22] Roya Sabbagh Novin et al. "A Model Predictive Approach for Online Mobile Manipulation of Non-Holonomic Objects Using Learned Dynamics". In: *The International Journal of Robotics Research* 40.4-5 (Apr. 2021), pp. 815–831. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364921992793. URL: <http://journals.sagepub.com/doi/10.1177/0278364921992793> (visited on 05/05/2022).
- [23] Jonathan Scholz et al. "Navigation Among Movable Obstacles with Learned Dynamic Constraints". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Daejeon, South Korea: IEEE, Oct. 2016, pp. 3706–3713. ISBN: 978-1-5090-3762-9. DOI: 10.1109/IROS.2016.7759546. URL: <http://ieeexplore.ieee.org/document/7759546/> (visited on 04/29/2022).
- [24] Neal Seegmiller et al. "Vehicle Model Identification by Integrated Prediction Error Minimization". In: *The International Journal of Robotics Research* 32.8 (July 2013), pp. 912–931. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364913488635. URL: <http://journals.sagepub.com/doi/10.1177/0278364913488635> (visited on 02/16/2022).
- [25] Max Spahn. *Urdf-Environment*. Version 1.2.0. Aug. 8, 2022. URL: https://github.com/maxspahn/gym_envs_urdf.

- [26] Jochen Stüber, Marek Kopicki, and Claudio Zito. “Feature-Based Transfer Learning for Robotic Push Manipulation”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)* (May 2018), pp. 5643–5650. DOI: 10.1109/ICRA.2018.8460989. arXiv: 1905.03720. URL: <http://arxiv.org/abs/1905.03720> (visited on 03/15/2022).
- [27] Jochen Stüber, Claudio Zito, and Rustam Stolkin. “Let’s Push Things Forward: A Survey on Robot Pushing”. In: *Frontiers in Robotics and AI* 7 (Feb. 6, 2020), p. 8. ISSN: 2296-9144. DOI: 10.3389/frobt.2020.00008. arXiv: 1905.05138. URL: <http://arxiv.org/abs/1905.05138> (visited on 02/24/2022).
- [28] Marc Toussaint et al. “Sequence-of-Constraints MPC: Reactive Timing-Optimal Control of Sequential Manipulation”. Mar. 10, 2022. arXiv: 2203.05390 [cs]. URL: <http://arxiv.org/abs/2203.05390> (visited on 04/29/2022).
- [29] William Vega-Brown and Nicholas Roy. “Asymptotically Optimal Planning under Piecewise-Analytic Constraints”. In: *Algorithmic Foundations of Robotics XII*. Ed. by Ken Goldberg et al. Vol. 13. Springer Proceedings in Advanced Robotics. Cham: Springer International Publishing, 2020, pp. 528–543. ISBN: 978-3-030-43088-7 978-3-030-43089-4. DOI: 10.1007/978-3-030-43089-4_34. URL: http://link.springer.com/10.1007/978-3-030-43089-4_34 (visited on 06/23/2022).
- [30] M. Verhaegen and Vincent Verdult. *Filtering and System Identification: A Least Squares Approach*. Cambridge ; New York: Cambridge University Press, 2007. 405 pp. ISBN: 978-0-521-87512-7.
- [31] Maozhen Wang et al. “Affordance-Based Mobile Robot Navigation Among Movable Obstacles”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Las Vegas, NV, USA: IEEE, Oct. 24, 2020, pp. 2734–2740. ISBN: 978-1-72816-212-6. DOI: 10.1109/IROS45743.2020.9341337. URL: <https://ieeexplore.ieee.org/document/9341337/> (visited on 06/28/2022).
- [32] Liangjun Zhang, Young J. Kim, and Dinesh Manocha. “A Simple Path Non-existence Algorithm Using C-Obstacle Query”. In: *Algorithmic Foundation of Robotics VII*. Ed. by Srinivas Akella et al. Vol. 47. Springer Tracts in Advanced Robotics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 269–284. ISBN: 978-3-540-68404-6 978-3-540-68405-3. DOI: 10.1007/978-3-540-68405-3_17. URL: http://link.springer.com/10.1007/978-3-540-68405-3_17 (visited on 06/16/2022).

6

Appendix

This chapter todo A.

test some stuff, do you see this Vimtex?
oh boy VimTex really is awesome!