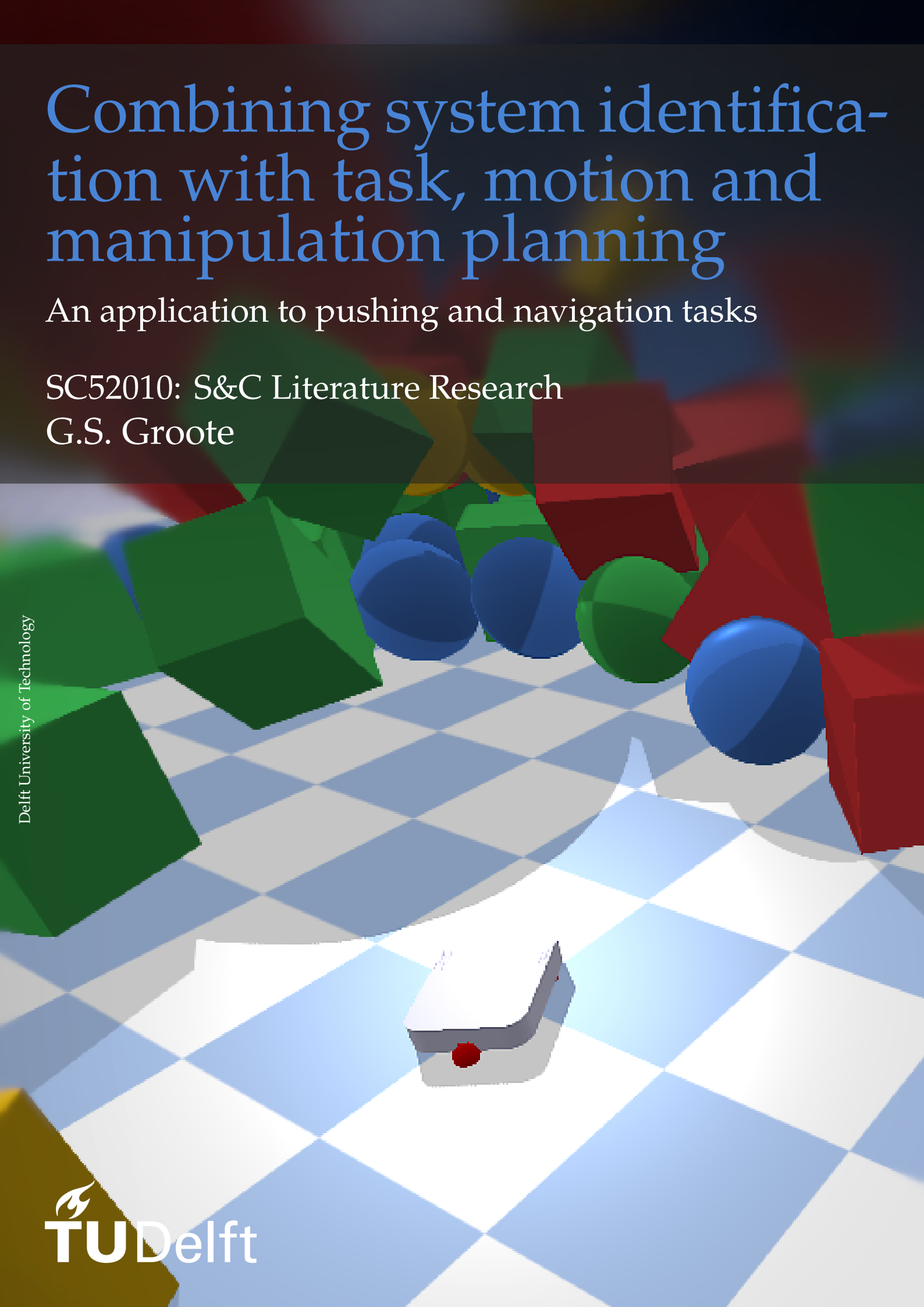
Combining system identification with task, motion and manipulation planning

An application to pushing and navigation tasks

SC52010: S&C Literature Research

G.S. Groote



Delft University of Technology

TU Delft

page intentionally left blank.

Combining system identification with task, motion and manipulation planning

An application to pushing and navigation tasks

by

G.S. Groote

Student Name	Student Number
Gijs S. Groote	4483987

Supervisor:	M. Wisse
Daily Supervisor:	C. Pezzato
Project Duration:	Nov, 2021 - Sept, 2022
Faculty:	Faculty of Cognitive Robotics, Delft

Cover: Simulation environment used during the literature study **[spahn_urdf-environment_2022]**

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

TU Delft | Delft Center for Systems and Control | THE AI FOR RETAIL LAB DELFT | Cognitive Robotics

Abstract

This literature research investigates 3 topics in robotics and combines them in a proposed solution to see if they complement and enhance each other. These topics are *learning system models* with system identification techniques, *task and motion planning* in an environment with movable obstacles and lastly *manipulation* of objects to push them to target locations. After investigating existing solutions to these individual topics, existing literature which combines 3 of these 3 topics is researched and discussed. Multiple studies combine learning system models with Navigation Among Movable Obstacles (NAMO), but leave manipulation out of the scope, studies related to object manipulation are mostly combined with learning system models, and leave NAMO out of the equation. The one study found combining both 3 topics uses an radically different approach compared to the proposed method in this literature. This allows to conclude that this literature enters new territory in the field of robotics, because the proposed method combines all 3 topics. The research question is similar to: can these 3 topics be combined? and reads: Can objects' system models be learned by a robot during task execution, and can these newly learned models improve task, motion and manipulation planning?

A solution is proposed which combines the 3 topics and answers the research question. Even though the proposed solution still has to be verified, the individual research topics have been shown to be successful many times over. The robustness of backward induction reveals the possibility for converging toward reliable, stable performance of completing tasks.

G.S. Groote
Delft, April 2023

Contents

List of Figures

List of Tables

1

Introduction

In 1961 the first industrial robot Unitmate was put to work [**noauthor_unimate_2022**]. From that point on, industrial robots proved very successful in factories. At the assembly line, robots do predefined tasks in a known environment. Research and engineering expanded and improved the robot's abilities over the years. In recent decades a new ability has gained attention, operating in other environments than at the factory floor. Such as in people's homes, in supermarkets or in a hospital. These environments are changing fast and contain a variety of objects, making them more complex and more unpredictable. As a result the robot operates in a environment which is less defined compared to the factory floor. To operate in these lesser defined environments robots need to be more autonomous compared to the assembly line counterpart. There do not yet exist mature solutions to the arisen challenges which autonomous robots need to overcome, explaining why creating autonomous robots which can operate in unpredictable fast-changing environments is such a hot research topic nowadays. Although autonomous robots are a hot topic in research, it has gained less attention from the industry. Companies such as Boston Dynamics or SIASUN create autonomous robots that are promising with potential, but there is still a lot to improve before autonomous robots can be called a success.

One challenge of a autonomous robot is to define the environment with the objects in it. To motivate why the robot should need knowledge of the environment it operates in, a comparison with ourselves is made. For us humans, mostly we are rewarded if we improve our interactions, such as typing is faster with 10 fingers instead of 2 which makes you type faster, or improving playing an instrument gives the joy of being able to play more interesting music. Learning is thus rewarded. Which nature understood because nature fine-tuned humans to understand the world we live in. Already in the womb, a baby learns that it can move, that is has arms. A small child always seems to grasp everything, in a lot of cases to check if it is edible. Children go through a phase where they question almost everything to understand the world around them. Because if you understand the world around you, it's easier to accomplish tasks. For example, a barista who works at a coffee bar can struggle with just a few customers on the first day, but after a month he/she can provide coffee for large groups effortless because of an increased understanding of coffee and the coffee bar.

For autonomous robots, finding a action sequence to fulfil a task is hard, because actions taken now influence the future state of the environment, successful completion bears uncertainty and actions cannot always be rolled back. In this literature study, improving robots to be more autonomous is investigated in 2 areas. The **first** one is having system models of objects to describe their dynamics. A robot can interact with objects by grasping, pushing, pulling or manipulate objects in other ways (e.g. throwing, kicking). Different objects react differently to, for example a push by the robot. How these objects react can be captured with system models. The true system dynamics are dominated by nonlinear dynamics, contains uncertain parameters and can change over time, which is why autonomous robots need to constantly monitor. It is assumed that if the model estimation improves, manipulation improves. **Secondly**, to have more autonomous robots, attention should be paid to task planning because of the influence of current action on future tasks. In this report we investigate the most prominent task planning techniques such as backward induction [**krontiris_dealing_2015**]. The scope of this literature

is formalised through the following problem description.

1.1. Research Question

The main question is separated into multiple subquestions.

Main research question:

Can objects' system models be learned by a robot during task execution, and can these newly learned models improve task, motion and manipulation planning?

Research subquestion:

1. How are system models learned, and what are the limitations for different system learning techniques.
2. Why is task, motion and manipulation planning so much more difficult compared to task and motion planning.
3. Can planning in configuration space be extended to a piece-wise analytic configuration space with learned dynamics.

1.2. Problem Description

In this literature we focus on the problem of task, motion, and manipulation planning in a simplified simulated environment where a robot is commanded to push possibly unknown object to target locations. This is a well investigated problem in the literature [**sabbagh_novin_model_2021**, **wang_affordance-based_2020**, **scholz_navigation_2016**, **siciliano_path_2009**, **goldberg_asymptotically_2020**] and it touches on both system identification and task planning.

Assumption 1, 2, 3, 4 and definition 1 fully define the environment.

Assumption 1 ***Closed-World Assumption:*** *Objects are manipulated, directly or indirectly only by the robot. Objects cannot be manipulated by influences from outside the environment.*

Assumption 2 ***Quasi-Static Assumption:*** *Velocities are small enough that inertial forces are negligible [**stuber_lets_2020**], thus objects only move if manipulated by the robot.*

Assumption 3 ***Perfect Object Sensor Assumption:*** *the robot has full access to the poses and geometry of all objects in the environment at all times.*

Assumption 4 ***Task are Commutative*** *Tasks exist of multiple objects with specified target positions. The order in which objects are pushed toward their target position is commutative.*

Definition 1 ***Environment description***
The robot environment consists of a flat ground plane, gravity pulls all 3-dimensional objects perpendicular toward the ground plane. Let tuple $< g, Origin, Ob, E >$ *define the world, where:*

- g *is the gravity constant.*
- $Origin$ *is the worlds origin from where the position and orientation can be defined.*
- Ob *is a set of be a set of objects* $ob_1, ob_2, \ldots, ob_n$.
- E *is a set of motion equations which describe the behaviour of objects and interaction between objects. The motion equations coincide with the true dynamics, in a simulation environment these are known, in the real world the motion equations are unknown.*

Each object contains mass, a center of gravity, geometry and a state consisting of position, velocity, orientation and angular velocity $< pos_x, pos_y, pos_z, pos_\theta, pos_\gamma, pos_\rho, vel_x, vel_y, vel_z, vel_\theta, vel_\gamma, vel_\rho >$. *The state of an object* i *is expressed in short hand notation state* s_i *of an object* ob_i.

Now the robot's environment has been defined, let's look at the robot. The robot is a mobile robot, without any grippers. The only action which is can take is driving around, structure about the robot's driving dynamics is assumed to be known because the robot can be analysed during design and construction. Details about the robot remain unknown such as the exact radius of the robot's wheels, and mass because such parameters can change over time, system identification could estimate such parameters. Interaction with object can be performed by bumping into objects to perform a push. The robot contains a set of controllers and various methods to identify the objects system model. Because controllers use different types of models the term 'system model' generalises all models which capture the real dynamics and are able to simulate future states of objects as a function of the current states and robot inputs. The combination of a controller and a compatible system model can be used to navigate the robot in the environment.

A configuration c_i associated with object ob_i is defined as a subset of the object's state $c_i \subseteq s_i$. A task is defined as set of desired configuration of objects, containing one or more objects with associated configurations. Such a desired set of configurations, or target configurations do have some rules to adhere to. It is required that every object can at most have one target configuration, and objects cannot be overlapping with other objects target configuration, including geometry of the object. Task must be commutative, see assumption 4. The target configuration must be stable (so no object floating above the ground or orientations such that is tips over). The robot is commanded by means of driving and pushing to put the objects from their current state in their target configuration with an certain threshold. Above mentioned requirements do not make every task feasible, the robot could be trapped by unmovable objects, or the target is to move an object to heavy to push.

A robot having knowledge about the environment's objects and true dynamics is expected to be more efficient in reaching the target configurations compared to a robot without any knowledge. This literature investigates this claim providing a number of related works and their analysis.

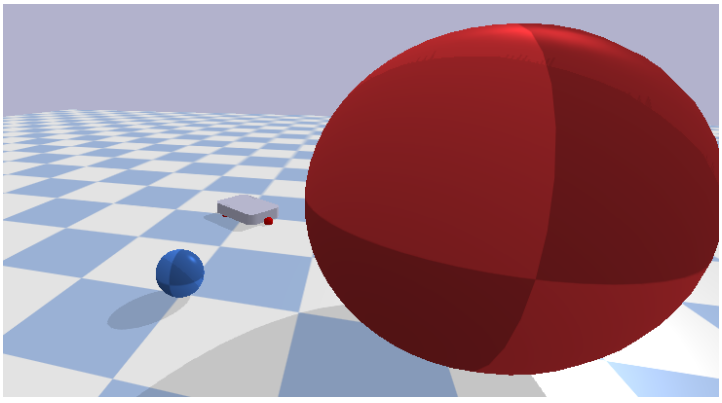


Figure 1.1: An example world with the boxer robot and 2 sphere obstacles

In a more practical setup, a simulation is used to create the robot environment. Figure 1.1 is an example of such an environment. Before running the simulation, the robot is provided with the geometry of every object, objects do not change geometry during simulation. When simulating the environment the robot is updated with objects states for every time step. What is unknown it the objects' weights and the set of motion equations E.

1.3. Report Structure

So far, an introduction, problem description and research questions has been presented. The goal was to narrow down the problem at hand, give a clear overview of the robot environment.

Chapter 2 answers the first research subquestion. Before an answer can be given, system models are categorised in section 2.1, followed by a categorisation of control methods that can make use of the

system models in section 2.2. The first research subquestion is then answered in section 2.3.

Chapter 3 gives an overview of Task And Motion Planning (TAMP) methods, split in a section for motion and manipulation planning in section 3.1, and a section for task planning in section 3.2. Research subquestion item 2 is then answered in the discussion section 3.3.

Chapter 4 drafts a new method to overcome the limitations shown in chapter 2 and chapter 3. This new method introduces the *hypothesis graph* in section 4.2 and introduces the *knowledge graph* in section 4.3. Section 4.4 creates a variety of environments with tasks for the robot to fulfil. These test display the expected hypothesis graph and expected knowledge graph indicating how the proposed method completes tasks. Finally the last research subquestion is answered in section 4.5.

Chapter 5 will provide a brief summary of the discussion sections followed by an answer to the main research question.

2

Interaction with the Environment and Model Identification Methods

*This chapter will provide an overview of **model identification methods** and **controllers** for robot interaction with the environment. Interaction includes for example robot driving, or push manipulation. A system model estimates the true dynamics of one, or multiple objects. A system model can be learned with system identification methods. Different model representations are categorised in section 2.1. In this report, the controllers' focus is on tracking a reference signal. Not all, but the majority of controllers require a system model to determine system input. The controllers are categorised in section 2.2. For models and controllers, a distinction is made for one body (e.g. the robot itself) and for multiple bodies (e.g. the robot pushing a ball). Performance and limitations are discussed in section 2.3.*

2.1. System Model Representation

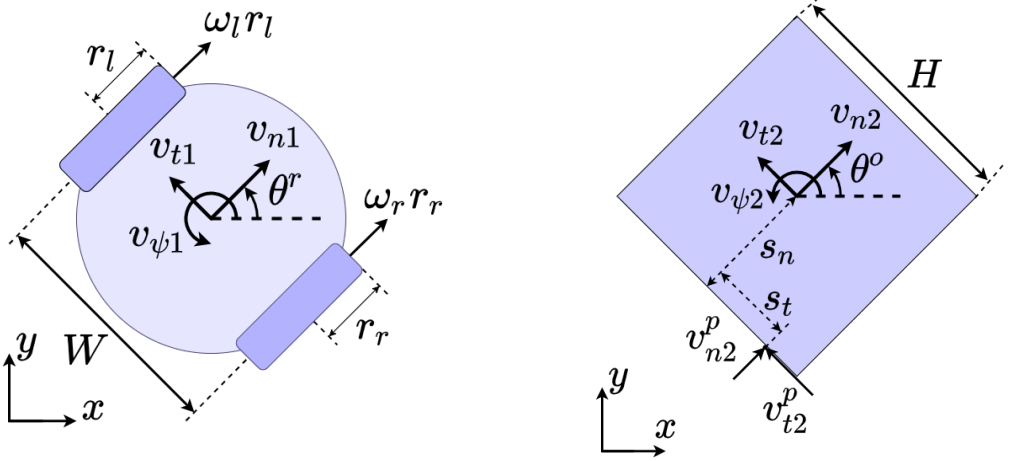
Let's first clarify the use of system models. The goal of system models is to capture the true dynamics of the system it describes. For instance, consider the following examples: when a robot arm reaches for some product to pick, a system model can estimate how the robot arm reacts before actually sending the input. Or consider a sphere which has received a push, the behaviour after the push is different from a cube which received an equal push, thus their models must be different also. A system model combined with the current state of the system and potential system input can estimate the state a system will be in as a result of the system input. An example of system input is wheel velocity ω_l and ω_r in figure 2.1a. State information and the effect of system input are crucial information for robot control design, and for controllers to operate effectively, robot controllers will be discussed in the next section.

Models investigated in this literature are split into 2 other categories, single-body models and multi-body models. Such a distinction is made because both models have quite different dynamical properties, let's first define both. A **single-body model** is a system model which estimates the true dynamics of an object, which is assumed to be connected for all times. An example is a robot arm existing of multiple parts which are connected by joints (assuming that a part of the robot arm does not break off). The robot arm is a single-body and any model estimating only the true dynamics of only the robot arm belongs to the set of single-body models. **Multi-body models** are system models which estimate the true dynamics which involve multiple objects. These multiple objects can be connected or disconnected. Multi-body models include any combination of single-/multi- body models with other single-/multi-body models. For example, a robot arm grasping a box with a gripper. For the period the box is grasped by the robot gripper, the single-body model estimating the true dynamics of the robot arm can be augmented such that it estimates the pose of the box. Such an augmented model belongs to the set of multi-body models.

The true dynamics of a single-body contain some nonlinear parts such as slip and friction, however in general nonlinear dynamics are not dominating such that a single-body system can be estimated with Linear Time-Invariant (LTI) models. Dynamics which allow simplification to a linear model dominate

compared to the nonlinear dynamics of the true dynamics, small accumulating errors as a result of for example slip can be accounted for. Stable control using LTI models is possible because the true dynamcis can be estimated by an LTI system model. Opposed to single-body system, multi-body systems are dominated by nonlinear dynamics, so much that a simplification to a linear system erases most true dynamics. Such a compromise for multi-body models is undesired since it would lead to a model mismatch between the model and the true dynamics, motivating the separation of single- and multi-body models.

2.1.1. Example Model



(a) Schematic 2D diagram of differential drive robot in world frame $< x, y, \theta >$, the robots frame's origin is located at (x^r, y^r, θ^r) and denoted as $< n1, t1, \psi 1 >$. The robot frames origin is where the robot's center of mass is located. The robot's left wheel has radius r_l, the right wheel has radius r_r, the robot wheels act upon the ground in the center of the wheel. The distance between where the left and right act upon the ground is notated as W. The angular speed of the left and right wheels are denoted by ω_l and ω_r respectively.

(b) Schematic 2D diagram of cube object in world frame $< x, y, \theta >$, the object frame's origin is located at (x^o, y^o, θ^o) and denoted as $< n2, t2, \psi 2 >$. The object frames origin is where the cube's center of mass is located. Both length and width of the cube are H. A contact point p is located at s which can be decomposed in s_n and s_t. An outside force pushes against at point p the cube resulting in velocity component v^p which can be decomposed in v^p_{n2} and v^p_{t2}.

Figure 2.1: Two example of single bodies

let's consider an example system model, a differential drive mobile robot displayed in figure 2.1a. During design and construction, the robot can be fully analysed. When given the task to model the differential drive robot, it is thus reasonable to assume that some prior knowledge about the robot is known (e.g. weight or relation between forward velocity v_{n1}, wheel radiuses r_l, r_r, angular speeds ω_l and ω_r). In that case, selecting and designing a *differential equation* or *parameterisable differential* equation is a logical option for reasons which will be discussed in sections 2.1.2 and 2.1.4. Literature reveals that a differential equation is a popular option to model a system [**bauza_data-efficient_2018**, **seegmiller_vehicle_2013**]. Robots can come across a variety of objects, such as the cube object displayed in figure 2.1b, both the differential drive robot and the cube object are modelled as differential equations displayed by equation (2.1) and equation (2.2) respectively.

$$\dot{\rho}^r(t) = \begin{bmatrix} \dot{x}^r(t) \\ \dot{y}^r(t) \\ \dot{\theta}^r(t) \end{bmatrix} = f(\rho^r(t), u^r(t)) = \begin{bmatrix} \cos(\theta^r(t)) & 0 \\ \sin(\theta^r(t)) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{r_l}{2} & \frac{r_r}{2} \\ -\frac{r_l}{W} & \frac{r_r}{W} \end{bmatrix} \begin{bmatrix} \omega_l(t) \\ \omega_r(t) \end{bmatrix} \tag{2.1}$$

With states $\rho^r(t) = [x^r(t) \quad y^r(t) \quad \theta^r(t)]^\top$, input $u^r(t) = [\omega_l(t) \quad \omega_r(t)]^\top$
and constants W, r_l, r_r, > 0. Equation (2.1) from [**seegmiller_vehicle_2013**].

Equation (2.1) displays an analytical single-body model, if the equation would be expressed in robot frame it would be more compact because the transformation matrix (matrix containing both $\cos(\theta^r(t))$ and $\sin(\theta^r(t))$) is then scrapped, and instead of 3 state variables only two are sufficient. Later on, the two

single bodies will be combined to one multi-body model, expressed in a world frame. For consistency, all system models accompanying the robot and cube example are in world frame.

Equation (2.2) estimates the true dynamics of the cube object, because v_{n2}^p acts upon the object, the object will translate and rotate. if the speed acts on the cube at the middle $s_t = 0$ then the speed will completely go to translation of the cube $v_{n2} = v_{n2}^p$. If the speeds acts on the cube at the corner $s_t = \pm\frac{H}{2}$, then the speed will completely go-to rotation of the cube, $\theta^o = v_{n2}^p s_t$. Linear interpolation between the middle and the corner points determines the ratio between translation and rotation, which is a function of speed v_{n2}^p described as:

$$\dot{\theta}^o = \frac{2s_t}{H}|s_t|v_{n2}^p$$

$$v_{n2} = (1 - |\frac{2s_t}{H}|)v_{n2}^p$$

yielding the following differential equation:

$$\dot{\rho}^o(t) = \begin{bmatrix} \dot{x}^o(t) \\ \dot{y}^o(t) \\ \dot{\theta}^o(t) \end{bmatrix} = f(\rho^o(t), u^o(t)) = \begin{bmatrix} (1 - |\frac{2s_t}{H}|)\cos(\theta^o(t))v_{n2}^p \\ (1 - |\frac{2s_t}{H}|)\sin(\theta^o(t))v_{n2}^p \\ \frac{2s_t}{H}|s_t|v_{n2}^p \end{bmatrix} \tag{2.2}$$

With states $\rho^o(t) = [x^o(t) \quad y^o(t) \quad \theta^o(t)]^\top$, input $u^o(t) = \begin{bmatrix} v_{n2}^p & s_t \end{bmatrix}^\top$
and constant $H > 0$ and constraint $|s_t| \leq \frac{H}{2}$. Equation (2.2) is inspired by the analytical model in [**bauza_data-efficient_2018**].

Equation (2.2) is not approximating the true dynamics accurately, because pushing a cube using this differential equation does not capture any speed in the direction of the t-axis at point p. There is no friction of the object with the ground, let alone a distinction between static and dynamic friction which both affect the dynamics considerably. Simplifying the true dynamics results is model mismatch, too much simplification would eventually lead to a nonsense model, incapable of modelling the system behaviour accurately and leading to stability issues when used by a controller. Equation (2.2) simplifies too much and is thus an ineffective modelling method. However, for the purpose of an example equation (2.2) is sufficient. To improve modelling the true dynamics of a robot pushing a box, more details of the true dynamics should be captured. [**bauza_data-efficient_2018**] created an analytical model for push manipulation involving Coulomb friction, force and friction constraints, resulting in a model modelled accurate enough to successfully track a reference signal. [**bauza_data-efficient_2018**] performed a close inspection of the object to push. Assuming prior knowledge about the object to encounter is unrealistic as opposed to robot dynamics. Now the two single-body models will be combined as a multi-body model, a schematic 2D diagram is displayed in figure 2.2.

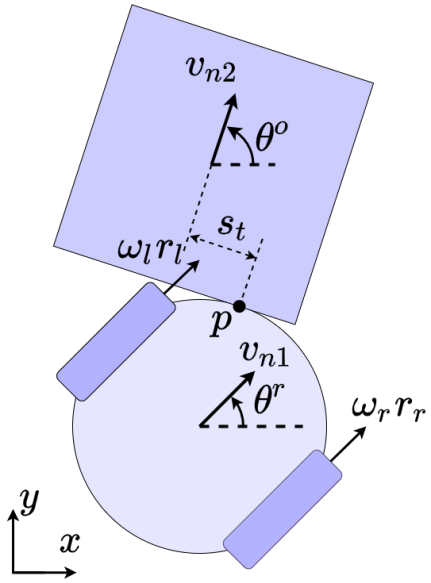


Figure 2.2: Combining single bodies figures 2.1a and 2.1b which are connected at contact point p to create a multi body.

Augmenting the differential equation (2.1) with equation (2.2) creates a multi-body model:

$$\dot{\rho}^{ro}(t) = \begin{bmatrix} \dot{x}^r(t) \\ \dot{y}^r(t) \\ \dot{\theta}^r(t) \\ \dot{x}^o(t) \\ \dot{y}^o(t) \\ \dot{\theta}^o(t) \end{bmatrix} = f(\rho^{ro}(t), u^r(t)) \tag{2.3}$$

$$= \begin{bmatrix} \cos(\theta^r(t)) & 0 & 0 \\ \sin(\theta^r(t)) & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & (1 - |\frac{2s_t}{H}|)\cos(\theta^o(t)) \\ 0 & 0 & (1 - |\frac{2s_t}{H}|)\sin(\theta^o(t)) \\ 0 & 0 & \frac{2s_t}{H}|s_t| \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ cos(\theta^r(t) - \theta^o(t)) & 0 \end{bmatrix} \begin{bmatrix} \frac{r_l}{2} & \frac{r_r}{2} \\ -\frac{r_l}{W} & \frac{r_r}{W} \end{bmatrix} \begin{bmatrix} \omega_l(t) \\ \omega_r(t) \end{bmatrix}$$

Combining both single-body models displayed in equations (2.1) and (2.2) to obtain a multi-body model.

with s_t now dependent on both the location and geometric properties of the robot and the object, defined as:

$$s_t = \sqrt{(x^o(t) - x^r(t) - \frac{H+W}{2}cos(\theta^o(t)))^2 + (y^o(t) - y^r(t) - \frac{H+W}{2}sin(\theta^o(t))^2}$$

The multi-body model in equation (2.3) displays how the first derivative of state variables can be calculated based on the input $u^r(t) = [\omega_l \quad \omega_r]$, the system constants H, r_l, r_r, W and the state variables. The multi-body model estimates the true dynamics of the robot and the box. The robot and cube object are touching at point p, when the objects become disjoint the multi-object model is not a valid

representation of the true dynamics any more.

This concludes the example of multiple single-body models into a multi-body model. In the example, we saw one way of analytically modelling a robot and an cube object. There is however a vast literature of different methods which could been applied to model figure 2.2 [**nascimento_nonholonomic_2018**], [**bauza_data-efficient_2018**], [**stuber_feature-based_2018**], [**stuber_lets_2020**]. An overview of modelling methods reviewed is conveniently condensed into figure 2.3. Now the distinction between single-body models and multi-body models is clear, and the advantages and disadvantages per class of models are discussed.

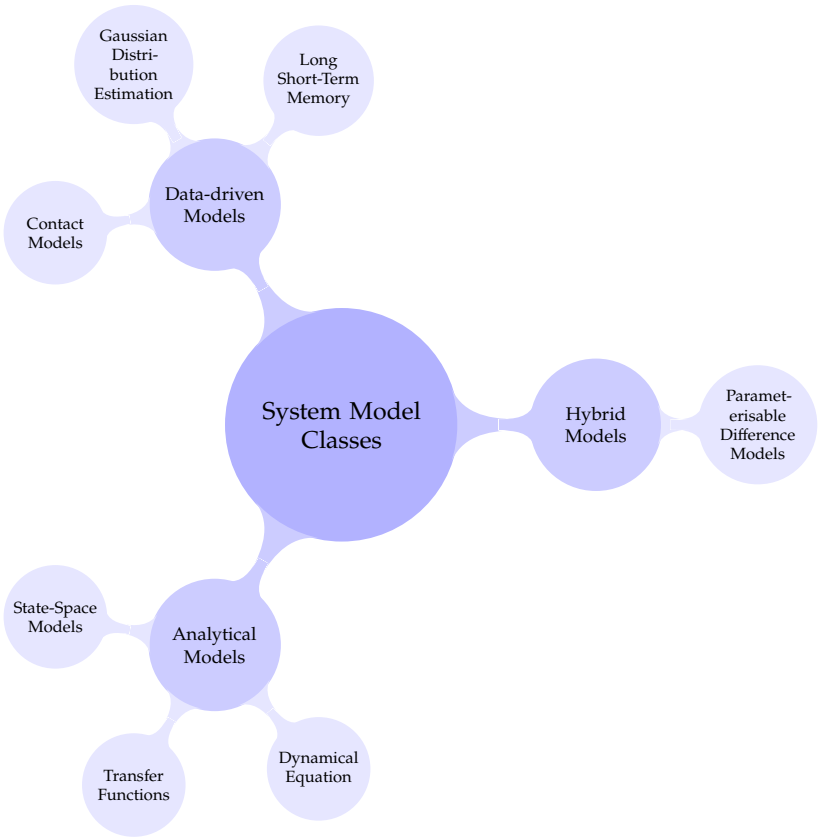


Figure 2.3: A mind map showing the classification of different system models representations investigated in this literature. The first level of children displays the model classes, the second level displays examples of the model classes, classification [**stuber_lets_2020**].

2.1.2. Analytical models

Historically, analytical models are the first models to emerge, most prominently used are *state-space* representations, *transfer functions* and *differential equations*. Building an analytical model requires thorough knowledge of the system it models, because every system parameter, such as mass, damping coefficient, the center of gravity, geometry, friction coefficient or inertia. Analytical approaches rely on accurate identification of physical parameters which makes analytical models unfit for manipulation while learning system models [**arruda_uncertainty_2017**], [**stuber_feature-based_2018**].

Nevertheless, the work in [**bauza_data-efficient_2018**] manages to create a stable controller for push manipulation using an analytical model. Because thorough model identification of the pushable object was performed, the trajectory error stayed within reasonable boundaries.

2.1.3. Data-driven models

Among recent studies, data-driven models shown an uptrend in popularity [**mericli_push-manipulation_2015**], [**bauza_data-efficient_2018**], [**stuber_feature-based_2018**], [**stuber_lets_2020**]. Fully data-driven meth-

ods don't model any structure of the system it describes, or use a generalised model which applies to all. A system is viewed as a black box, which is fed input and gives back output. This reduces the need for prior information about the system significantly. Input-Output (IO) data is analysed to estimate the structure of the black box. The IO data analysed which solemnly serves the creation of a model is called the *model train set*. The advantage of requiring a minimum of prior information comes at the cost of the amount of IO data required. If there is not sufficiently much data, or the data is not rich enough then the model will not be accurate. For example, [**bauza_data-efficient_2018**] compared a purely analytical approach with a data-driven approach in push manipulation. The data-driven approach can take up to 200 samples of IO data to sufficiently match the performance of an analytical controller. With more IO data, data-driven approaches lower output errors and increase performance, outperforming analytical approaches but also outperforming hybrid approaches, which are discussed in the next subsection. Data-driven approaches outperform because data-driven approaches capture even tiny dynamical details of the true dynamics. That is, assuming that the dynamical details reside in the IO data.

A field of research which has become very popular in recent years is artificial intelligence. In for example, estimating physical parameters from data. This is what [**denil_learning_2017**] has shown with the use of deep reinforcement learning. By learning physical parameters artificial intelligence can help tackle the main issue with analytical approaches, estimating physical parameters. Instead of explicitly estimating physical parameters, another approach is learning a dynamics model directly [**stuber_lets_2020**]. [**cong_self-adapting_2020**] showed a powerful self-adapting push controller based on a *long short-term memory* modelling approach. A powerful advantage is online adaptation, which lowers a time-consuming train set to a warm start of approximately 5 test pushes with an object to push.

Alternatively, a set of robot-object test pushes can be used to fit a underlying distribution. Where test pushes and their effect on the object's position and velocity are taken as a train set, then a distribution is fitted mapping the train set's input to it's output. Such an fitted distribution is called a *contact model*. Recent literature shows a *Gaussian distribution* is a popular distribution used by [**mericli_push-manipulation_2015**] and [**stuber_feature-based_2018**]. [**stuber_feature-based_2018**] includes, additionally to a robot-object model (fitted from test pushes) an object-environment model, which considers object surface features. Object-environment information is added by modelling the surfaces of an object. All objects have surfaces, from a simple cube with 6 equal surfaces to more complex shapes. For the object-environment contact model, a central aspect is the probabilistic modelling of surface features, which describes every surface as a 3D position, 3D orientation and 2D local surface descriptor that encodes local curvatures. The data stored for surface features come from 3D point clouds created with a depth camera. An advantage is that the object environment model can be constructed based on 3D point cloud, and no interaction with the object is required. After having learned the contact model (object-environment model), a robot-object motion model is learned for which test pushes are required [**stuber_feature-based_2018**].

Contact models used for push manipulation make use of robot-object contact or additionally use object-environment contact. With enough rich data contact models outperform analytical and hybrid approaches. To tackle the amount of data required a more hybrid approach is developed. Which separates agent-object contact from object-environment contact. To generate a new model, two things are required, first an object-environment contact model of the object to model, and second, a sufficiently learned agent-object contact model. The latter does not necessarily have to be created from the object to model, existing agent-object contact models, combined with transfer learning can be sufficient [**kopicki_learning_2017**].

The nonlinear effects which dominate multi-object system resides in IO data, because data-driven approaches models are not assuming any structure which could limits capturing nonlinear effects, the data-driven approaches are a worthy method for estimating true dynamics of in particular multi-body systems. It must be mentioned that data-driven methods outperform other model classes with enough data, for which the training time is in robotics not always available. If other modelling approaches are available which estimating true dynamics accurate enough, the data-driven approach should be avoided because of the long lasting training time.

2.1.4. Hybrid models

Hybrid models are an extension of analytical approaches with data-driven methods. Whilst the interactions between objects are still represented analytically, some quantities of interest are estimated based on observations (e.g. the coefficients of friction) [**stuber_lets_2020**]. Recent literature reveals the foremost hybrid methods are parameterisable differential equations. Parameterisable state-space models and parameterisable transfer models do exist, though the most widely used parameterisable model remains a parameterisable differential model, which takes the form:

$$\frac{dx}{dt} = f(x, u, p) \tag{2.4}$$

where x is the state vector, u is the input vector and p is the parameterisation which needs to be found such that $f(x, u, p)$ accurately estimates the true dynamics. With a random or educated initial guess of the parameterisation p, a system model is provided without full knowledge of all system parameters. An example parameterisation for analytical model example equation (2.1).

An initial guess as parameterisation may not be a very accurate model, but it does allow to skip a tedious system identification period. Online adaptation allows to converge to a local minimum during execution. Whether this local minimum also coincides with the global minimum is dependent on the optimisation technique and the initial guess. Parameterisable differential models are very powerful in situations where the general structure of dynamics is known, but certain parameters e.g. weight, the friction coefficient is unknown or change over time [**seegmiller_vehicle_2013**].

Single-bodies can and should be modelled as hybrid models, hybrid models allow a workable model which can be created from only the prior structural knowledge of the system. After some off or online system identification detailed parameters can be found, as effect the model will converge toward the true dynamics. While data-driven methods outperforms hybrid methods such methods take long to properly train. Multi-body models are dominated by nonlinear dynamics, to fully capture such nonlinear dynamics, data-driven methods can and should model multi-body model, even if this means collecting a large train set.

In a environment with unknown objects, the ability to rapidly interact with objects is provided by hybrid models. During interaction hybrid approaches can improve their model accuracy whilst also adapting to changing systems. To fully capture the push mechanics, data-driven methods should be used, because only data-driven methods are able to capture a large portion of the nonlinear dynamics.

2.2. Interaction Approaches and Model Identification Methods

This section will describe the most prominent control methods which are applicable for controlling single- or multi-bodies. In general controllers have 3 abilities, which are tracking a reference signal, stabilising a system or rejecting disturbances, a controller can focus on one or a combination of these 3 abilities. The controllers' goal in this literature lies in tracking a reference signal to lower the output error. In section 2.1 single-body models and multi-body models were introduced, now their control versions are introduced. **Single-body control** is an interaction approach which learns a single-body model and controls a single-body. As a reminder, a single-body is an object which is assumed to be connected for all times. **Multi-body control** controls two or more single- or multi- bodies. A multi-body controller uses a multi-body model. For example, a mobile robot pushing a ball. A controller actuates the robot directly and the object indirectly via the robot. Another example is a controller actuating a robot arm with a gripper holding a box. Single-body control involves driving for mobile robots and moving for a fixed robot. Multi-body control involves the robot pushing an object. The ability to push greatly broadens the robot's capabilities. Objects which are too heavy to lift could potentially still be pushed, objects out of reach to grasp could be in reach to push, and a gripper holding an object can additionally push another object but cannot grasp two objects at the same time.

Existing literature presents several methods for model-based robot control. The most prominent and established techniques can be categorised as predictive methods such as Model-Predictive Control (MPC) and reactive methods such as Proportional-Integrated Derivative (PID) control. Literature shows 2 types of model-free methods, completely model-free methods which act directly on IO data, and model-free methods which are provided with data, but not explicitly given any model. Such methods service by

analysing IO data and using system identification to update a model, whilst a controller uses such a model simultaneously.

Extensive research on robot controllers from last decades has been categorised in section 2.2. Table 2.1 provides a more detailed overview of some interaction approaches.

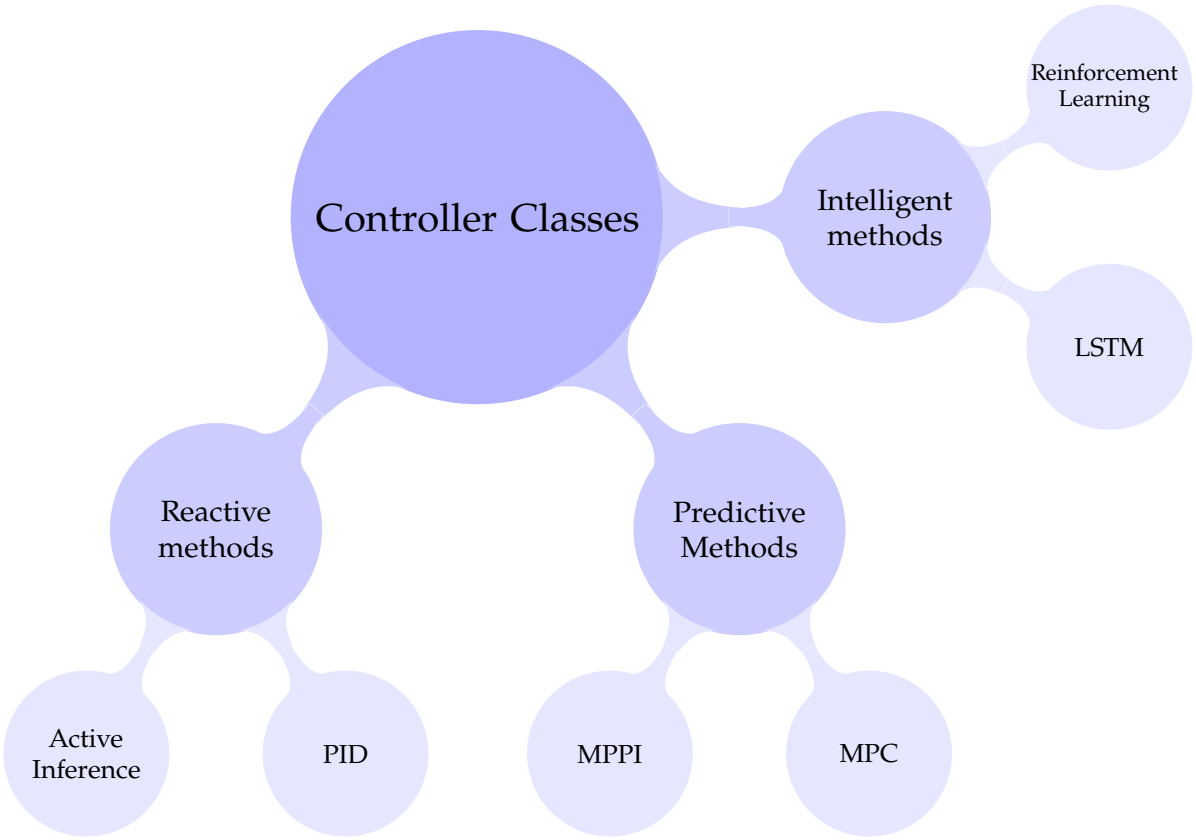


Figure 2.4: A mind map showing the classification of control methods investigated in this literature. The first level of children displays the control classes, and the second level displays examples of the control classes [**mehra_map_2022**]

2.2.1. Reactive Methods

Historically reactive methods were the first to arise. A widely used control method is the PID controller. It's widely used due to its simplicity, clear functionality and ease of implementation. Tuning a PID controller can be split up into 3 tuning approaches. Heuristic-tuning, rule-based tuning and model-based tuning [**tools_explore_2022**]. A heuristic tuning method is one where general rules are followed to obtain approximate or qualitative results. The trial-and-error method is an example of heuristic tuning. Heuristic tuning is a quick and easy method to obtain a reasonable result, finding good performance can be very time-consuming. Rule-based PID tuning methods assume a certain process response to obtain easy mathematical formulas that enable the tuning of a PID controller. Commonly used rule-based tuning methods are the Ziegler-Nichols, Chien, Hrones and Reswick tuning methods. Performance improves and some stability guarantees arise, however for rule-based tuning some prior process knowledge is required. Model-based tuning or optimization-based PID tuning allows you to obtain your P, I, and D parameters optimally. Control objectives such as disturbance rejection and reference tracking together with a model of your system and the engineering specifications of the closed-loop behaviour determine the final set of tuning parameters. Optimal performance comes at the cost of full prior system knowledge. For single-object control, the heuristic and rule-based tuning methods can apply because for these tuning methods full system knowledge is not required. For multi-body control, only the heuristic method could apply. In recent years automatically tuning PID gains with more complex methods has risen. Such as [**ahn_online_2009**] showed, who tuned a PID controller using fuzzy logic this shows reactive control methods can be used without the need for extensive prior knowledge of the system.

PID control has been around for many years. As opposed to this classical approach, the active Inference (AI) controller has been developed in the last decade and finds its origin in the field of neuroscience. Neuroscientist Karl Friston combined action selection with Prediction Error Methods (PEM), which created the free-energy principle [**friston_free-energy_2009**], [**friston_action_2010**]. In the field of robotics, [**pezzato_novel_2020**] proposed an AI controller and implemented it on a real system for the first time. The AI controller is implemented on a real system. The control approach is a model-free online joint space controller, which was implemented on a 7-DOF Panda robot arm. The controller adapted rapidly to a changing environment and accounted for additional Gaussian noise for the joint measurements. If the free-energy depends explicitly on the control actions, then AI can provide a excellent fault metric as [**baioumy_fault-tolerant_2021**] and [**pezzato_active_2021**] have shown. One of the current drawbacks with an AI controller, there does not yet exist a stability proof and the solution to which the controller converges might be a local minimum.

2.2.2. Predictive Methods

In recent literature involving predictive methods Model-Predictive Control (MPC) methods are dominating, before moving on to MPC and variations of MPC, MPC will briefly be explained. The basic concept of MPC is to use a dynamic model to forecast system behaviour and optimise the forecast to produce the best decision for the control move at the current time. Models are therefore central to every form of MPC. Because the optimal control move depends on the initial state of the dynamic system[**rawlings_model_2020**]. A dynamical model can be presented in various forms, let's consider a familiar differential equation.

$$\frac{dx}{dt} = f(x(t), u(t))$$
$$y = h(x(t), u(t))$$
$$x(t_0) = x_0$$

In which $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^m$ is the input, $y \in \mathbb{R}^p$ is the output, and $t \in \mathbb{R}$ is time. The initial condition specifies the value of the state x at $t = t_0$, and a solution to the differential equation for time greater than t_0, $t \in \mathbb{R}_{\geq 0}$ is sought. If little knowledge about the internal structure of a system is available, it may be convenient to take another approach where the state is suppressed, no internal structure about the system is known and the focus lies only on the manipulable inputs and measurable outputs. As shown in figure 2.5, consider the system $G(t)$ to be the connection between u and y. In this viewpoint, various system identification techniques are used, in which u is manipulated and y is measured [**rawlings_model_2020**]. From the input-output relation, a system model is estimated or improved. The system model can be seen inside the MPC controller block in figure 2.5.

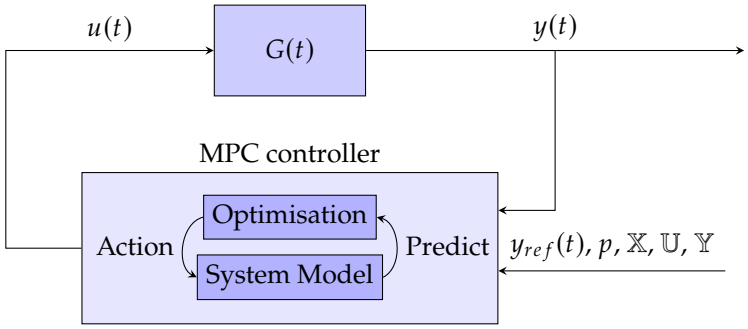


Figure 2.5: System $G(t)$ with input $u(t)$, output $y(t)$ and MPC controller with input $y(t)$, reference signal $y_{ref}(t)$, parameterisation p and constraint sets $\mathbb{X}$, $\mathbb{U}$, $\mathbb{Y}$

As indicated in the problem description in section 1.2, prior knowledge about the structure of the robot itself is known. Such structure can be specified in a parameterisable system model, in which the influence of inputs and current states describes the behaviour of states and outputs. Uncertain parameters have to be sought, such as the center of mass, mass, and diameter of wheels. These

parameters reside in the parameterisation p, see equation (2.4). Figure 2.5 shows the parameterisation p, for purely analytical system models the parameterisation is left empty. The following explanation about MPC is best understood with the differential drive robot in mind from figure 2.1a, with single-body model equation (2.1). Some states of the system might be inside an obstacle region, such a region is undesirable for the robot to be in or go toward. The robot states are allowed in free space, which is all space minus the obstacle region. The free space is specified as a state constrained set $\mathbb{X}$. Allowable input can be restricted by the input constraint set $\mathbb{U}$, a scenario in which input constraints are required is for example the maximum torque an engine produces at full throttle. Lastly, the set of allowed outputs is specified in the output constraint set $\mathbb{Y}$. State, input and output constraints must be respected during optimisation, the optimiser takes the state-, input- and output constraint sets $\mathbb{X}$, $\mathbb{U}$, $\mathbb{Y}$ and if feasible, finds an action sequence driving the system toward the reference signal while constraints are respected. The MPC system model predicts future states where the system is steered toward as a result of input actions.

The optimisation minimises an objective function $V_N(x_0, y_{ref}, \mathbf{u}_N(0))$, where $\mathbf{u}_N(k) = (u_k, u_{k+1}, \ldots, u_{k+N})$. The objective function takes the reference signal as an argument together with the initial state and the control input for the control horizon. The objective function then creates a weighted sum of some heuristic function. States and inputs resulting in outputs far from the reference signal are penalised more by the heuristic function than outputs closer to the reference signal. Because the objective function is a Lyapunov function, it has the property that, it has a global minimum for the optimal input $\mathbf{u}_N^*$. If the system output reaches the reference signal y_{ref}, x_{ref} then u_{ref} will be mapped to the output reference signal as such $y_{ref} = h(x_{ref}, u_{ref})$. As a result solving the minimisation problem displayed in equation (2.5) gives the optimal input which steers the system toward the output reference signal while at the same time respecting the constraints.

$$\begin{aligned} \underset{u_k, u_{k+1}, \ldots, u_{k+N}}{\text{minimize}} \quad & V_N(x_0, y_{ref}, u_k, u_{k+1}, \ldots, u_{k+N}) \\ \text{subject to} \quad & x(k+1) = f(x(k), u(k)), \\ & x \in \mathbb{X}, \\ & u \in \mathbb{U}, \\ & y \in \mathbb{Y}, \\ & x(0) = x_0 \end{aligned} \tag{2.5}$$

Figure 2.6 displays the predicted output converging toward the constant output reference. After solving the minimisation problem, equation (2.5), the optimal input sequence is obtained $\mathbf{u}_N^*$ (given that the constraints are respected for such input), from which only the first input is executed for time step k to $k+1$. Then all indices are shifted such that the previous time step $k+1$ becomes k, the output is measured and the reference signal, parameterisation, and constraints sets are updated and a new minimisation problem is created, which completes the cycle. Note that figures 2.5 and 2.6 is an example MPC controller, which hardly scratched the surface of MPC, there are many variations and additions such as deterministic and stochastic MPC, stage and terminal cost, distributed MPC, etc. which **[rawlings_model_2020]** visits extensively.

A major flaw for MPC was the computation time required to solve a minimisation problem every time step. Because processors' power has increased, the MPC framework can be applied in real time and is applicable for robotics. When applied to tracking a reference signal the MPC framework outperforms classic control approaches such as PID control **[nascimento_nonholonomic_2018]**. Where MPC excels at is tracking multiple objectives which can be weighted in the objective function. For example, MPC can primarily track a robot path, while secondarily satisfying some additional dynamic specifications. This makes MPC especially suitable in path tracking for nonholonomic robots. By restricting the state constraint set, obstacles can be avoided, such obstacles can even be avoided online because the state constraint set may be changed during execution. It should however be feasible to find an input which satisfies all constraints. Because of its ease of tuning, handling multiple objectives, and flexibility in adding constraints, MPC became the baseline standard to compare new control approaches with in control research. By now, linear MPC theory is quite mature, and important issues such as stability are well addressed in the last decade. Nevertheless, some systems are, in general, inherently nonlinear. Therefore, especially in highly dynamic systems such as mobile robotics, linear models are often inade-

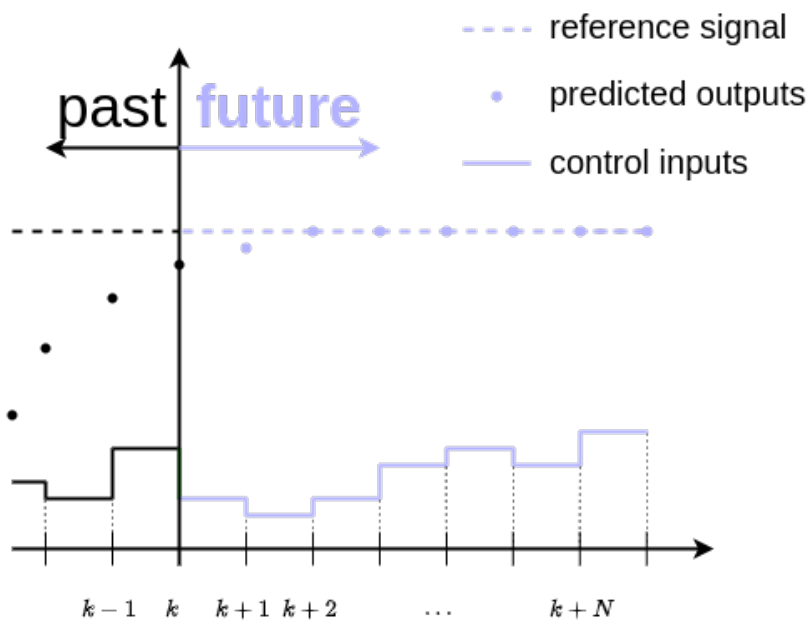


Figure 2.6: A discrete MPC scheme tracking a constant reference signal. k indicates the discrete time step, N the control horizon

quate to describe the process dynamics and nonlinear models have to be used. Thus nonlinear MPC theory is required for which closed-loop stability proofs are lacking [**nascimento_nonholonomic_2018**].

Now the reader has gained a basic understanding of the workings of MPC let's see relevant literature in the context of robot control in an environment with unknown objects. Starting with the literature accompanying single-body control.

(Integrated) Prediction Error Minimisation
As mentioned models are central in every form om MPC, so obtaining models is very important. Figure 2.5 displays a system model inside the block diagram used by the MPC controller. The MPC framework is flexible in handling different types of models, it accepts analytical, data-driven or hybrid forms. The MPC controller's performance heavily relies on the accuracy of the system model. Having an accurate system model is thus crucial in the MPC framework. One major system identification technique is the PEM approach, it is the core of black-box identification methods [**farina_convergence_2008**]. Generally in PEM methods, the objective is to determine, from a finite number of measurements of the input and output sequences, a one-step-ahead predictor without prior system knowledge, or the system and covariance matrices of stochastic disturbances. The creation of an estimated model yields a state-space model or a transfer function model from which predictions can be made. [**verhaegen_filtering_2007**]. An example is used to further clarify PEM methods.

2.2.3. Example Identification Method

Figure 2.7 displays a block diagram of the prediction-error model-estimation method. Here output $y(k)$ is created by summing system $G(q)u(k)$ with additional noise $H(q)e(k)$, the input $u(t)$ is stochastically independent of $e(k)$ a zero-mean white-noise sequence. The one-step-ahead predictor tunes the state matrices $A(p)$, $B(p)$, $K(p)$, $C(p)$, $D(p)$ such that the output error $\epsilon(k, p)$ is minimised.

Two important assumptions are: $G(q)$ is an LTI system, and the stationary one-step-ahead predictor displayed in equation (2.7) is of known order. In this example the goal is to obtain the state-space matrices from input-output data. Figure 2.7 displays a block diagram of the PEM methods. With input-output data generated as:

The one-step-ahead predictor of known order takes, for a state-space model the following form:

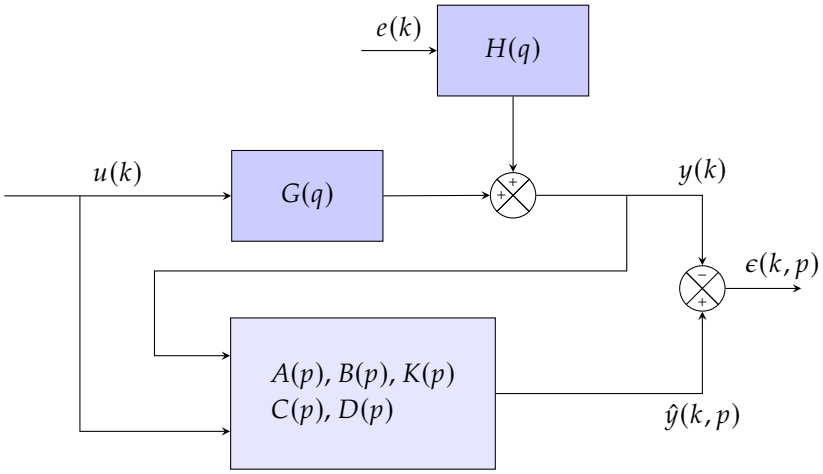


Figure 2.7: A block diagram displaying the structure of the prediction-error model-estimation method **[verhaegen_filtering_2007]**

$$y(k) = G(q)u(k) + H(q)e(k) \tag{2.6}$$

A signal-generating system, it's input-output data is to be used for identification. $G(q)$ represents the deterministic part and $H(q)$ the stochastic part of the system, both $G(q)$ and $H(q)$ are discrete transfer function models, from **[verhaegen_filtering_2007]**

$$\begin{aligned} \hat{x}(k+1) &= A\hat{x}(k) + Bu(k) + K(y(k) - C\hat{x}(k) - Du(k)), \\ \hat{y}(k) &= C\hat{x}(k) + Du(k) \end{aligned} \tag{2.7}$$

Given a finite number of samples of the input signal $u(k)$ and the output signal $y(k)$, and the order of the predictor. The goal is to estimate the system matrices A, B, C, D and K in this predictor such that the output $\hat{y}(k)$ approximates the output of equation (2.6)

A parameterisation of the one-step-ahead predictor as a stationary Kalman filter, equation (2.7) with parameterisation p and the one-step-ahead predictor of the previous time step gives the parameterised one-step-ahead predictor:

$$\begin{aligned} \hat{x}(k+1|k,p) &= (A(p) - K(p)C(p))\hat{x}(k|k-1,p) + (B(p) - K(p)D(p))u(k) + K(p)y(k) \\ \hat{y}(k|k-1,p) &= C(p)\hat{x}(k|k-1,p) + D(p)u(k) \end{aligned}$$

By minimising the output error $y(k) - \hat{y}(k|k-1,p)$ an optimal parameterisation is be found, such a parameterisation together with its parameterisable one-step-ahead predictor can be used by the MPC framework. Chapter 8, **[verhaegen_filtering_2007]** can elaborate further on PEM methods with subjects such as the ARMAX, ARX and Box-Jenkins parameterisation, the innovation model or closed-loop system behaviour. PEM system identification techniques rely on IO data, which must contain enough information, which in robotics can be an issue, since there is not always time to collect a rich enough IO data set. Though, PEM methods can with little prior knowledge formulate an accurate system model, if provided with enough IO data. **[farina_convergence_2008]** has compared single- and multistep PEM methods and convergence properties in a predictive control context, and provided proof that showing that single step and multistep PEM methods yield unbiased models. This allows the conclusion that PEM methods are proper methods to estimate a linear system model, provided that there is access to enough information-rich IO data of such a system. PEM assumes the true dynamics can be estimated accurately with an LTI system, PEM methods are ideal for single-body control. PEM for multi-body control is feasible, but not ideal since multi-body control is too nonlinear to be properly captured using an LTI-based system identification method.

An improvement on PEM which lacks capturing changing dynamics, was made by **[seegmiller_vehicle_2013]** where a system model is parameterised using Integrated Prediction Error Methods (IPEM). The idea is to integrate the left-hand side $\frac{dx}{dy}$ of the system dynamics, resulting in some favourable effects. IPEM can

calibrate offline (slip) and online (odometry), IPEM improves upon PEM by using only low-frequency measurements, and fewer ground truth measurements and slip can be accounted for. These plus points come at the cost of additional complexity. Because fewer measurements are required, IPEM detects and adjusts faster to changing system dynamics, which makes IPEM more suitable for robotic applications compared to PEM. Additionally PEM assumes an LTI system, where IPEM claims to converge for a linear time-variant system, broadening the set of systems IPEM can model. Especially for the following situation which is described in [**seegmiller_vehicle_2013**]. A robot plans to make an aggressive turn into a narrow corridor, however, due to unmodeled understeering, the robot actually makes a wider than expected turn driving into the wall of the corridor. The robot has deviated from the planned path. Based on position feedback, the robot compensates by planning a sharper turn. Once again, the robot fails to execute the planned path as its curvature exceeds the limits of the steering mechanism. The robot must now abandon the attempt and drive in reverse to avoid a collision. If the planner had an accurate model, it would have simply turned harder at the beginning of the turn when the turn was still feasible.

IPEM methods are ideal for estimating rapid-changing true dynamics with a stochastic nature, arising during tracking of a reference for a mobile robot with additional measurement noise. The mobile driving robot can be modelled as a linear time-variant model which accounts for slip and odometry changes.

PEM and IPEM are suitable for single-body models because single-body systems can be simplified to an LTI system. Multi-body systems cannot since they are dominated by nonlinear dynamics. Multi-body system thus require a different system identification method, recent literature reveals that a set of test pushes is the dominant data collection method for gathering a set of IO data. As the name "test pushes" suggests, data collection focuses on push manipulations and is collected by the robot performing several test pushes against the object from different angles. The push is taken as input, and the location of velocity of the object is taken as output to form a train set. From this train set, there exist multiple methods to create a system model. Let's review recent literature, for every new object, [**mericli_push-manipulation_2015**] creates, a sequence of random push actions and the effect of the push. A 3-dimensional Gaussian distribution (for 2D pose variables x, y and θ) is fitted on the sample pushes. The Gaussian distribution can be used as a one-step-ahead predictor used for control. A disadvantage of [**mericli_push-manipulation_2015**] is, after every push the object comes to a complete stop. A sequence of pushes is generated and executed in a push-stop-push-stop fashion, which could be one continuous push. [**bauza_data-efficient_2018**] overcomes this problem by converting the Gaussian process to linearized motion equations which can be directly fed into the MPC. Both [**mericli_push-manipulation_2015**] and [**bauza_data-efficient_2018**] use data-driven approaches, [**bauza_data-efficient_2018**] additionally has an analytical approach. Initially, an analytical approach has the lowest tracking error, after $\sim$ 100 number of tests pushes the data-driven approaches outperform analytical approaches because the push analytical models does model the more nonlinear parts of the true dynamics. Such as small variations in the sliding friction or the mass distribution. [**bauza_data-efficient_2018**] did show a data-driven stable controller can be created after only $\sim$ 10 test pushes.

In the MPC frameworks, several single- and multi-body methods are discussed, transitioning from single-body control to multi-body control can causes problems. Single-body control requires the constraints which belong to a single-body model, when single-body control switches to multi-body control the dynamical and kinematic constraints should update such that the multi-body constraints are respected. This discontinuity of model dynamics introduces control and planning problems. Control issues arisen during discrete dynamic models are now discussed, the arisen problems affecting planning will be discussed in the next chapter.

Reactive MPC

As [**toussaint_sequence–constraints_2022**] describes in their problem description: TAMP plans include switches in kinematic and dynamic constraints, and the original plan might provide temporal scheduling of such switches. However, for reactive execution the exact timing of constraint switches needs to be reactive, which either requires employing optimization methods invariant to such

switches [**toussaint_differentiable_2019**], [**posa_direct_2014**] or include explicit timing-estimation and -optimization as part of the MPC problem.

The solution proposed by [**toussaint_sequence–constraints_2022**] is to take the timing of switching constraints as a decision variable yielding a timing-optimal sequence-of-constraints MPC. Because the reactive MPC can work with somewhat accurate system models, the reactive MPC controller is ideal for objects for which a not very accurate model of the true dynamics is only available. The reactive behaviour makes the control algorithm well suited for pushing and avoiding collision of incoming objects. A disadvantage in reactive MPC is the CPU power it requires to run, another drawback is that a reactive MPC is unable to learn any system model, the reactive controller cannot give any guarantees when operating on unknown objects. However, if a (potentially inaccurate) single-and multi-body model for is provided the reactive MPC can successfully perform push manipulations. Single- and multi-body models obtained by different system identification methods can be combined with a reactive MPC controller.

Model Predictive Path Integral Control
Introduced by [**williams_model_2015**] Model Predictive Path Integral (MPPI) control arose. Which was followed by MPPI control combined with various system models, identification methods [**abraham_model-based_2020**], [**cong_self-adapting_2020**], [**arruda_uncertainty_2017**]. The core idea is from the current state of the system with the use of a system model and randomly sampled inputs to simulate in the future a number of "rollouts" for a specific time horizon, [**neuromorphic_tutorial_ltc21_2021**]. These rollouts indicate the future states of the system if the randomly sampled inputs would be applied to the system, the future states can be evaluated by a cost function which penalised undesired states and rewards desired future states. A weighted sum over all rollouts determines the input which will be applied to the system. If a goal state is not reached, the control loop starts with the next iteration. An example is provided, see figure 2.8.

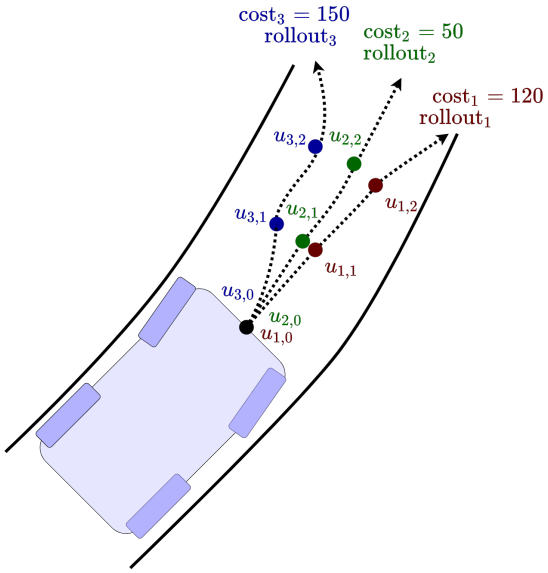


Figure 2.8: MPPI controlled race car using a control horizon of 3 time steps, with 3 rollouts all having their respected inputs as $u_{i,j}$ where i is the rollout index and j indicates the time step [**neuromorphic_tutorial_ltc21_2021**].

Here 3 rollouts are displayed, The objective function is designed to keep the car driving on the center of the road by penalising rollouts which are further away from the center of the road relatively more. resulting in a high cost for rollout$_1$ and rollout$_3$ compared to rollout$_2$. As a result, the input send to the system as a weighted sum of the rollouts is mostly determined by rollout$_2$. The weighted sum determining the input is displayed in equation (2.8), from [**neuromorphic_tutorial_ltc21_2021**].

$$u(k+1) = u(k) + \frac{\sum_i w_i \delta u_i}{\sum_i w_i} \tag{2.8}$$

Where δu_i is the difference between $u(k)$ and the input for rollout i, the weight of rollout$_i$ is detemined as: $w_i = e^{-\frac{1}{\lambda}\text{cost}_i}$, λ is a constant parameter. The reader is now somewhat familiar with the MPPI concept, Now some applications with MPPI and multi-object control are discussed.

From a test pushes train set, [**arruda_uncertainty_2017**] creates a forward model. The forward model is based on a Gaussian process and can sample multiple trajectories or rollouts in the future, these rollouts are then sent to the MPPI controller. While [**arruda_uncertainty_2017**] was able to create a forward model from scratch, it was unable to improve the forward model after the training phase. This strategy is ideal when the true push dynamics are fully unknown, but makes it less ideal when dynamics change over time.

[**abraham_model-based_2020**] proposed an Ensamble Model-Predictive Path Integral (EMPPI) controller which requires a partially unknown parameterisable system model of the robot and objects in the environment. Thus this method is as opposed to [**arruda_uncertainty_2017**] not data-driven. If such a model is provided, EMPPI constantly improves the parameterisation found during execution. Which is accomplished by recursively searching for a parameterisation for the partly known dynamics. It is however assumed that the true dynamics reside in the local minima of the parameterisation.

Both model identifying methods used in combination with a MPPI controller mentioned above need a training set obtained by test pushes. The training phase can only be over when the model results in a stable closed-loop controller. The capability to yield a stable controller is crucial in robotics where time and resources are limited. Currently the fastest stable controller for push manipulation was proposed by [**bauza_data-efficient_2018**] where the train set required a minimum of only ~ 10 train pushes. Then [**cong_self-adapting_2020**] even claims to require less than 5 train pushes. With the use of a recurrent Long Short-Term Memory (LSTM) based model to predict the motion of objects with unknown parameters. The LSTM model provides the Recurrent Model Predictive Path Integral (RMPPI) controller with motion predictions. By updating the LSTM model and executing RMPPI control at the same time the algorithm is self-adapting.

MPPI has some advantages in comparison to MPC. MPPI can be optimised by running multiple simulations in parallel. Parallel optimisation improves converging to a control policy [**williams_model_2017**]. MPPI is based on stochastic sampling, this makes the MPPI controller naturally take into account nonlinear dynamics, and incorporate nonsmooth/non- differentiable cost functions without approximations [**williams_model_2015**], which makes MPPI very applicable for multi-body control where the true dynamics mainly are nonlinear.

2.2.4. Intelligent Methods

Already seen in previous subsection paper [**cong_self-adapting_2020**] used a predictive method for control. To identify the system, an intelligent LSTM method was used. Intelligent system identification method are seen more often in recent literature, so has [**scholz_learning_2015**] shown multi-body control using a physics-based reinforcement learning approach. A key advantage intelligent methods provide is the learning speed intelligent methods offer, online adaption is for both [**cong_self-adapting_2020**] and [**scholz_learning_2015**] a major advantage over other methods, which makes them very suitable for learning dynamic properties of objects with unknown dynamics. However intelligent methods are out of the scope of this literature because they have trouble generalising. To elaborate, intelligent methods perform very well on the train set, but on unseen data intelligent methods lack performance. The entire point of learning is to be able to generalise to the unseen, even out-of-distribution, and it is currently very difficult to assess learning performance on novel tasks [**roy_machine_2021**].

Control and identification methods investigated in this literature are conveniently summarised in the following table:

System Iden. Type	Controller Type	Sources	Single- /Multi- body model	Requires
Single-Body Control				
	Fuzzy Control & PID	[**ahn_online_2009**]	Single	initial PID gains
PEM, IPEM	MPC	[**seegmiller_vehicle_2013**], [**farina_convergence_2008**]	Single	Nonlinear differentia equation
-	Active Inference	[**pezzato_novel_2020**]	Single	initial AI control parameter
Multi-Body Control				
-	Reactive MPC	[**toussaint_sequence–constraints_2022**]	Single, Multi	Dynamica model
Model Fitting	MPC	[**mericli_push-manipulation_2015**], [**bauza_data-efficient_2018**]	Multi	sample pushes
-	unknown	[**stuber_feature-based_2018**]	Multi	3D object point clou test pushe
-	EMPPI	[**abraham_model-based_2020**]	Multi	Partially Unknown Dynamics
LSTM	RMPPI	[**cong_self-adapting_2020**]	Multi	warm up s contact po
Physic-based Regrssion	-	[**scholz_learning_2015**]	Multi	Gripper Torques for all t

Table 2.1: Summary of interaction approaches and identification methods. The first column displays the model used by the controller, if no model identification method is used this is indicated with a "-". Prior knowledge is indicated in the "Requires" column, poses for single- and multi-bodies are assumed to be known for all time steps. The parameters used to fully store the model are indicated in the last column.

2.3. Discussion

In section 2.1 a categorisation of system models is made which are the analytical, data-driven models and hybrid models. Analytical models could be used in situations where a system is fully analysed. Even the robot itself cannot be fully analysed because of slowly changing true dynamics let alone any of the unknown objects. Analytic approaches are for this reason excluded from further investigation. Better suited is the data-driven approach where, with enough data nonlinear parts of the true dynamics are captured, given that enough data is collected and the nonlinear behaviour resides in the data collected. Whilst hybrid models quickly offer a stable model, data-driven models eventually outperform hybrid models. Assuming some structure of the true dynamics allows for obtaining a model fast, while losing from data-driven methods in accuracy during convergence. The hybrid approach is best suited for single-body models, while data-driven methods are best suited for multi-body models.

Multiple interaction approaches have been categorised, where predictive methods are the dominant

methods. The predictive methods are able to incorporate constraints en uncertainty comparably well. Predictive methods performance heavily depends on the models they use, the system identification method is thus an important factor for stability and overall performance.

Many approaches have been discussed to learn dynamical models and their limitations have been emphasised, Because limitations are method-specific the challenge lies in when to choose which interaction approach and which system identification approach. For example, in push manipulation without any prior knowledge the only methods applicable are ones which start with data-driven system identification. Some objects might jump discontinuously between single- and multi-body dynamics (e.g. a ball) such a situation ask for a timing-optimal MPC. Objects could be in a corner surrounded by walls, limiting the training phase to only perform test pushes from one side, the best candidate for such a situation would be LSTM based controller which requires a minimum amount of training pushes.

There is no best interaction approach, different task required different approaches. Mainly the identification approach should be chosen specialised for the task at hand. For robot driving the MPC control methods is best suited, using PEM for mostly constant system dynamics and IPEM for changing system dynamics. Multi-body systems are best controller using MPPI control because they naturally incorporate the nonlinear mechanics, the modelling method should take nonlinearities into account, which are data-driven methods such as contact models, or a more complex methods such as LSTM.

3

Task, Motion and Manipulation Planning

*This chapter will provide an overview of different **TAMP** methods. In the previous chapter, chapter 2 the nonlinear nature of single-body, but especially multi-body systems was shown. The constraints due to the nonholonomic nature of the robot is captured in system models, system models will assist as local planners during motion and manipulation planning. Motion planning discussed in section 3.1 is conveniently split in a part for single bodies in section 3.1.1 and a part for multi-bodies in section 3.1.2. Decidability of motion and manipulation planners is analysed in section 3.1.3. Solutions to a given task seldom exist of a single motion. For multiple motions, the effects of a motion executed propagates further to future task planning. Such arisen problems related to TAMP and possible solutions are discussed in section 3.2. Performance and limitations are discussed in section 3.3.*

3.1. Toward a Target Pose

Decades of research yield a numerous amount of path-finding algorithms for many different applications [**lavalle_planning_2006**, **karaman_sampling-based_2011**]. Such a broad field of research is lessened down to methods applicable to robotic applications. Before moving further, let's first clarify the configuration space and motion planning. The *configuration space* is a set of possible transformations that could be applied to the robot [**lavalle_planning_2006**]. A robot can be described as a point in configuration space, which is a robot's *configuration*. From the robot's dimensions and it's configuration all points of the robot can fully be described. The configuration space consists of the non-overlapping obstacle space and free space. The robot can encounter obstacles which should be avoided, obstacles are represented in the *obstacle space*. The robot is allowed in *free space*, represented as a set of configurations in configuration space, and not in the obstacle space. Finally, motion planning is, given 2 robot configurations in free space, finding a feasible path between such 2 robot configurations. Two examples clarifying motion planning are presented in figures 3.1 and 3.3. Motion planning is split into 2 categories, motion planning for single-bodies and motion planning for multi-bodies also called *manipulation planning*.

3.1.1. Motion Planning

Figure 3.1 displays a 2D representation of a motion planning task. In this example, the configuration space is 3-dimensional containing position and orientation (x, y and θ). In configuration space, a path between the starting and target configuration should be found while avoiding obstacles and respecting constraints. While learning object dynamics, motion planning should be performed while having access to partly known or unknown robot dynamics, translating into partly known or unknown knowledge of constraints. With limited knowledge of constraints, motion planners might tend to find paths infeasible for the robot to track. For example, a feasible path found without full knowledge of the constraints could actually be unfeasible. Model mismatch might also yield unfeasible paths, and might prevent finding feasible paths. Again stressing the importance of system models. As a rule of thumb, respect known constraints, expect unknown constraints. Recent literature on the topic of motion planning with constraints is a subject which has had a considerable amount of research [**kingston_sampling-based_2018**, **lavalle_planning_2006**]. The most prominent are sampling-based

methods and graph-based methods which will now be discussed.

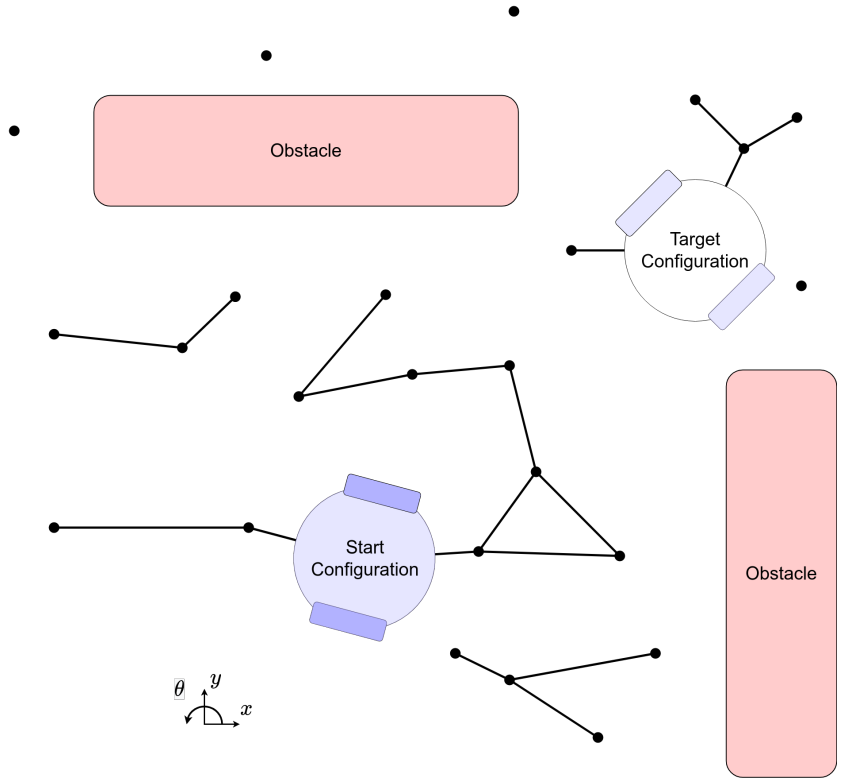


Figure 3.1: Sample-based motion planning task including 2 obstacles, a start and target configuration and a visual representation of sampled configurations with the connectivity graph

Sample-Based Planners The general idea behind sampling-based planning is to avoid computing the free space exactly and instead sample configurations in free space and connect them to construct a tree or graph that approximates the connectivity of the underlying free space [**kingston_sampling-based_2018**]. To connect 2 sampled points, a one- or multi-step-ahead predictor must be able to connect both, such a predictor is also commonly called a local planner in vast literature, a *connectivity graph* indicates sampled configurations which are verified by a local planner, an visual example for manipulation planning can be seen in figure 4.3. Using learned single-body models and one-step-ahead predictors discussed in chapter 2 sampled points in configuration space can be connected. For a sampling-based planner to plan with constraints, sampling and local planning must be augmented to satisfy constraints, as both of these elements directly affect whether the planning generates a valid path; as such, most methods focus on these two elements [**kingston_sampling-based_2018**].

Even with infinite sampling, the optimal path could be withheld because model mismatch does not allow the optimal path to be found. Logically a motion planner validating constraints using a nonsense local planner, will (most probably) not find the optimal path or will find a path which in reality is unfeasible. The overwhelming majority of solution techniques are sampling-based. This is motivated primarily by the extreme difficulty of planning under differential constraints [**lavalle_planning_2006**], such as constraints due to nonholonomic nature of the robot. Chapter 14 from LaValle [**lavalle_planning_2006**] is a interesting read on sampling-based planner under differential constraints. Contains a classification of planning problems, reachability guarantees, and various metrics to sample into a direction toward the target configuration.

Notably, most techniques for constrained sampling-based motion planning do not alter the core mechanics used by sampling-based planners. Generally, constrained sampling-based algorithms are adaptations of existing algorithms that incorporate a methodology for constraint satisfaction

[**kingston_sampling-based_2018**]. Local planners provided by system models provide a validator for constraint satisfaction, it is thus now sufficient to look at motion planners without constraints and complement them with local planners. A summary of sampling-based is displayed in table 3.1, where a formal analysis is done of most prominent sampling-based methods, such as Probabilistic RoadMap (PRM) and Rapidly-exploring Random Tree (RRT) and RRT*.

Algorithm	Probabilistic Completeness	Asymptotic Optimality	Monotone Convergence	Time Complexity Processing	Query	Space Complexity
PRM	✓	✗	✓	$O(n \log n)$	$O(n \log n)$	$O(n)$
RRT	✓	✓	✓	$O(n^2)$	$O(n)$	$O(n^2)$
RRT*	✓	✓	✓	$O(n \log(n))$	$O(n)$	$O(n^2)$

Table 3.1: Summary on sampling based methods with space and time complexity as function of the number of samples n in a fixed environment, from [**karaman_sampling-based_2011**]

A challenge in robotics is the convergence speed at which a path is found. Ideally, path finding is fast enough, such that the robot can continuously operate. Increasing convergence speed can be attained by implementing a faster algorithm or by applying techniques to improve the local planner.

One method to speed up convergence is to add sampled configurations to the connectivity graph starting from both the starting and target configuration, applied by [**chen_fast_2018**] who showed that RRT* with double-tree it outperformed existing RRT*. Techniques to speed up local motion planners are beautifully categorised in [**kingston_sampling-based_2018**] two major techniques are *relaxation* where constraints are allowed to be broken for a small threshold to find feasible paths faster. The other major technique is *projection* where an infeasible sampled configuration is projected onto the feasible region. [**kingston_sampling-based_2018**] explains these concepts in more elaborate and lists techniques such as tangent space, atlas or reparameterisation which all help to increase convergence.

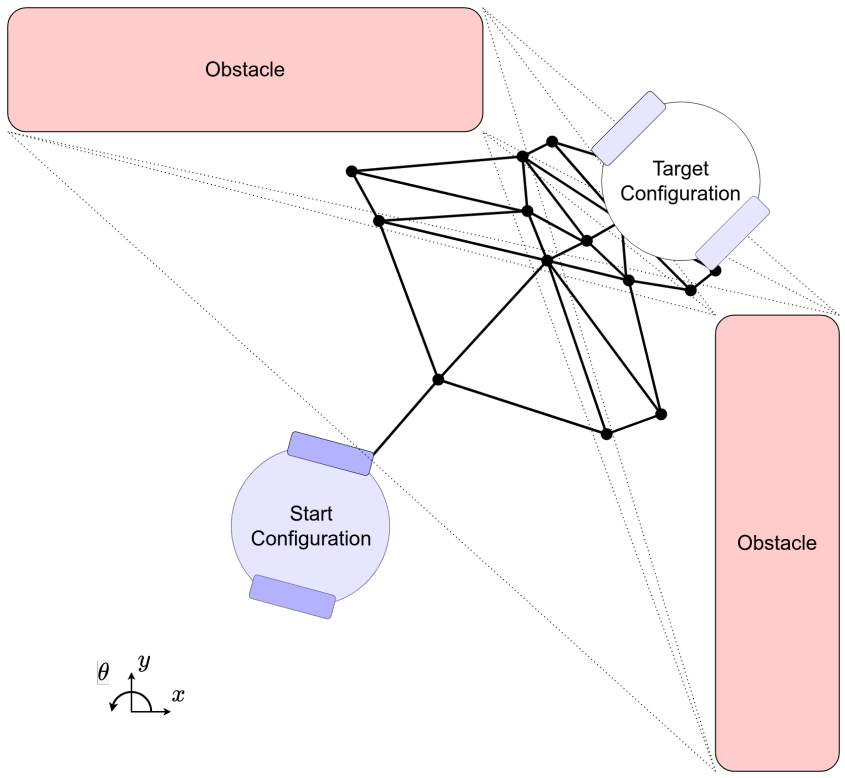


Figure 3.2: Graph-based motion planning task including 2 obstacles, a start and target configuration and a visual representation of graph-based configurations with the connectivity graph

Graph-Based Planners An alternative to sampling-based methods is a finite discretization (with for example a grid, or a cell decomposition) of the configuration space. Which gives rise to *graph-based* search algorithms such as: A* and Dijkstra [**karaman_sampling-based_2011**]. However because the number of grid points grows exponentially with the dimensionality of the configuration space, so does the worst-case running time. Which is the reason that randomisation is so powerful when exploring high-dimensional search spaces.

When the dimensionality of the configurations spaces increases, the time it takes to fully discretize grows exponentially. Only after a full discretezation, a graph-based search can start. Where as sample-based planners which can immediately start sampling configurations. In high-dimensional configuration spaces a graph-based methods take a comparatively longer time then sample based methods because of the required discretezation period, thus sample-based methods are the better choice for motion planning. When the system model converges toward the true dynamics, the local planner improves as a result, and the motion planner will yield more feasible paths. Graph-based planners will not play a role during planning but can provide a solution to another problem discussed in section 3.1.3.

3.1.2. Manipulation Planning

Manipulation planning differs from motion planning in a few aspects. Recall that manipulation planning is motion planning for multi-body systems, whereas motion planning refers to planning for a single-body system. With multi-body systems, one body (the robot) is controlled which indirectly controls the object when the robot pushes, pulls or picks the object, the robot is manipulating the object. Generally, with motion planning there is prior knowledge about the robots dynamics and constraints available, with manipulation planning there exist more constraints, which are lesser known, depending on the amount of system identification performed on the specific robot-object combination.

Let's consider the example manipulation planning task in figure 3.3, here both the robot and the cube object are in the starting configuration represented in configuration space again consisting of X, Y coordinate and orientation θ. Manipulation tasks are completed as the object is in the desired position,

the robots target configuration is irrelevant. Constraints which must be respected are the single-body constraints from the robot and the object, the multi-body constraints and that all configurations must be in free space.

For a configuration the object is in, infinite possibilities exist for the robot to be in, resulting in a exponential explosion of the number of possible configurations. Such a configuration space of the robot and objects is referred to a joint configuration space. Directly searching the joint configuration space is unwise because the number dimension is to high to find a path in reasonable time (far from real-time motion planning). Even so planning in a joint configuration space is shown to be successful, by implementing enormous simplification, section 3.2.2 discusses the joint configuration space more extensive.

In recent literature many push manipulations simplify pushing by using an holonomic pusher (e.g. a controlled stick) instead of an nonholonomic robot [**bauza_data-efficient_2018**, **kopicki_learning_2017**, **cong_self-adapting_2020**, **arruda_uncertainty_2017**]. If manipulation planning is performed with the use of a nonholonomic robot, other measured must be taken to counter the computational expensive search.

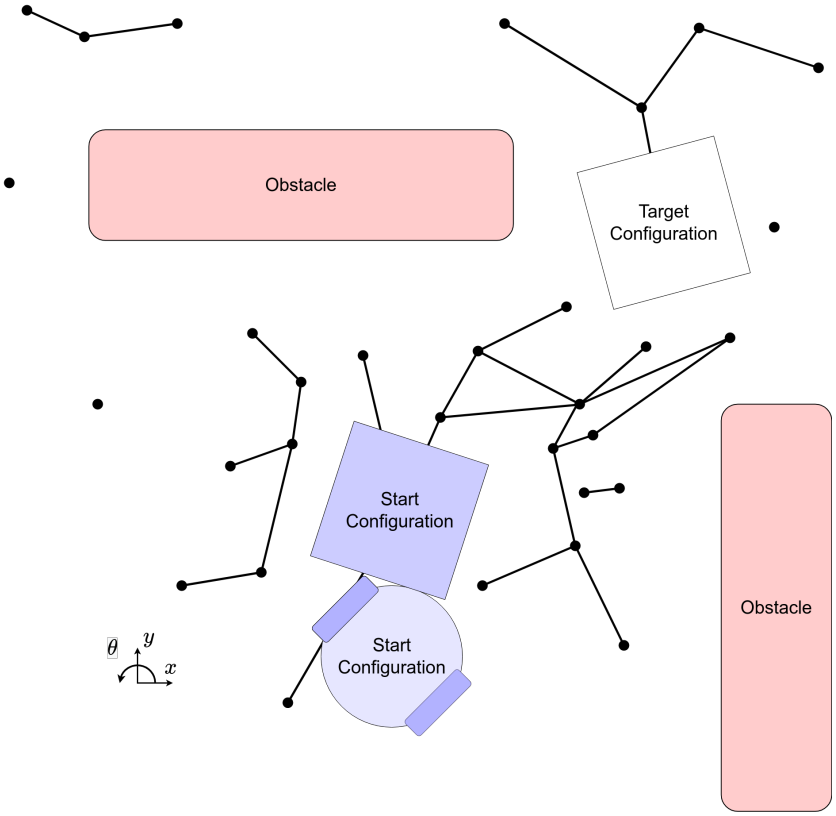


Figure 3.3: Manipulation planning task, to push the box toward the target configuration, including 2 obstacles, sampled configurations and the connectivity graph

The paper [**mericli_push-manipulation_2015**], already discussed in sections 2.1.3 and 2.2.2 for it's data-driven model and model identifying method, provides a RRT algorithm which uses the experimentally acquired model. An example simplification of the search space would be to only use large test pushes during data collection. Large test pushes only results in a local planner connecting 2 configuration with large pushes only. The planner then finds a path consisting of multiple large pushes and not a continuous push, as a result a path will be tracked by multiple pushes where the object comes at a complete stop after every push, which is far for optimal. While [**mericli_push-manipulation_2015**] managed to create a stable planner for a data-driven model identifying method, the push manipulation is far from optimal because the local planner is unable to produce a continuous push.

A simplification can be made forcing a multi-body into a single body, achieved by gripping an object with 2 grippers and then planning for a single-body model [**scholz_navigation_2016**]. During execution [**scholz_navigation_2016**] learns the multi-body model consisting of the robot and a gripped object. [**scholz_navigation_2016**] detects an immovable object or an object that is immovable in an subspace of its configuration space by comparing predicted with measured outcome. Detection can differ between an completely immovable object, or an partly immovable object (e.g. it can only rotate but not translate), which is made possible by checking prediction errors per subspace dimension.

Converting manipulation planning completely to motion planning simplifies the problem considerably. [**arruda_uncertainty_2017**] neglects the pusher, plans only for motion of the object, the robot pusher is then tasked with manipulating the object such that it tracks the found path with an MPPI controller. Such a strategy requires additional measures to counter the neglected constraints imposed by the multi-body model and the starting configuration of the pusher. In [**arruda_uncertainty_2017**] case this is done by planning a push in the direction with high model certainty, but most hard constraints can be neglected because the robot is an holonomic push arm and not an nonholonomic robot.

Using multiple methods to speed up computation, R. Sabbaragh Novim e. al. developed an dedicated search algorithm for path finding of single- and multi-body model [**sabbagh_novin_optimal_2016**]. Methods used are discretisation of the configuration space, a receding planning horizon which allows planning toward the target, but prevent planning all the way up until the target. Sabbaragh proceeds to use his algorithm in an hospital setting, in which a robot with gripper is tasked to bring walking aid to patients who request it [**novin_dynamic_2018**]. Improving upon his own work 2 years later, adding improved model identification methods allowing a larger variety of object to manipulate [**sabbagh_novin_model_2021**].

For manipulation planning recent literature has made one thing very clear, direct planning in a multi-body configuration space is computationally infeasible. Sample-based planners are a valid manipulation planning strategy as long as the multi-body configuration space is avoided. Thus a simplification of the joint configuration space is required. A few recently used simplifications are discretising by allowing only a small selection of pushes, gripping an object creating a single body, considering only the object to push during planning or planning toward the target without planning the path entirely. The bulk of manipulation planners are sample based. The RRT* algorithm is preferred because it is a sample-based planner, probabilistic complete and has a favourable time complexity.

3.1.3. Path Existence

The standard Piano Mover's Problem is PSPACE-hard [**reif_complexity_1979**], motion planning is PSPACE-hard [**canny_complexity_1988**], optimal planning is NP-hard [**canny_new_1987**], the only known methods for exact planning under differential constraints in the presence of obstacles is for a double integrator system $\ddot{x} = u$, with x the state and u the input [**lavalle_planning_2006**]. Aiming for an optimal solution in motion or manipulation planning with differential constraints is clearly out of the question, the point is that path planning is practically undecidable. With exeptions for linear systems and a dublin's car where motion planning is found to be decidable[**cheng_decidability_2007**], unfortunately such systems are not the scope of this literature. For any motion planning task any of the two outcomes is true, there exist a path from start to target and searching for it makes sense, or there does not exist a path from start to target and searching for it is meaningless. Especially for sampling-based methods, where without a stopping criteria, the sampling algorithm keeps on sampling. Of course a certain threshold can be set, but then the next difficult question arises, what threshold to set? It is thus worth searching for existence or non-existence of a path, proving path existence requires searching for a path with infinite sampling, which is impossible. But path non-existence can in some cases be detected, for example if a piano physically does not fit through a door (assuming the door is the only path and no other exists). Knowledge of non-existence of a path reduces the search space. One such example is caging, where a robot is trapped, which is proven to be detectable [**amato_caging_2020**].

For *holonomic* robots, checking for path path-existence can be estimated by discretizing the configuration space with the obstacles while accounting for the robot's dimensions [**akella_simple_2008**]. The

estimation becomes more reliable if the grid size at discretization decreases. Graph-based algorithms such as A* can find a path from start to target cell, or halt when no path exists for a finite graph. If no path exist for a holonomic version of the robot, then there does also not exist a path for the nonholonomic robot. If there exist a path for the holonomic robot, it might not exist for the nonholonomic version of the robot. The configuration samples from the grid cell can be used as a "warm start" during motion or manipulation planning. In the case a path exists for the holonomic version of a nonholonomic robot, a path will be most likely found during a check for path non-existence, but will not be found during motion or manipulation planning, even if the same samples from the path existence algorithm are used as a warm start.

The importance of estimating path existence has been pointed out. Non-existence of a path can be proven, and path-existence can be estimated. Simple checks such as suggested by [**akella_simple_2008**] can prevent unnecessary calculations.

3.2. Toward an Sequence of Target Poses

In previous section motion and manipulation planning, planning for a single action was discussed, feasibility of a path depends on local planners in the form of predictors provided by system models discussed in chapter 2. In this section planning for a longer horizon is discussed, *task planning*. As a reminder, a task was defined as a set of objects with associated target configurations. Task planning is defined as the following arisen problem. When given a task, which actions should successfully be performed and in which order to fulfil the given task. Finding such an action sequence in an environment with movable obstacles bears the name NAMO. An example is given in figure 3.4 and serves to clarify the planning problem. In this example the robot is tasked with placing the cube on the location where the sphere currently is.

A possible action sequence manually derived would be:

1. drive toward the ball
2. if no system model is available, perform system identification on the ball
3. push the ball away such that the box's target position is free
4. drive toward the cube
5. perform system identification on the cube
6. push the cube to the target location

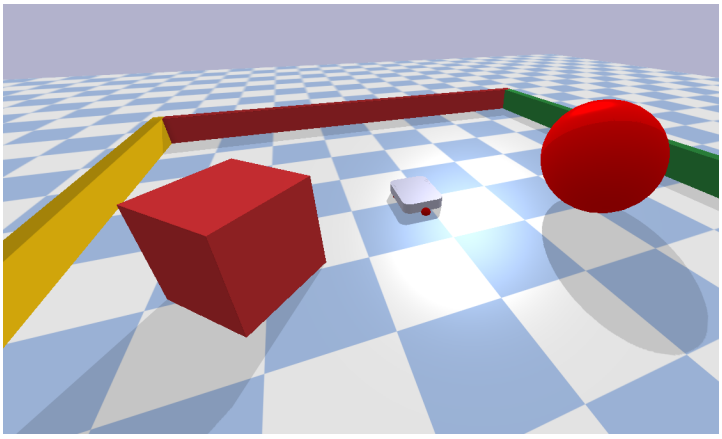


Figure 3.4: Example environment with robot, a sphere- and cube object and unmovable walls. The robot is tasked with pushing the cube to the location where the sphere currently is

Essentially the task planners job is to create an action sequence, containing subtasks in a logical order which fulfils a larger given task. The task planners reviewed in this literature study fall in to one of the 3 categories: a high-level planners, planning in a joint configuration space and hierarchical planners.

Task planners can validate a single motion by querying a motion planning algorithm, discussed in section 3.1.1. Motion planning algorithms can be queried resulting in a path or the message a path cannot be found, from the perspective of the task planner it appears thus that motion planning is decidable even though it may sometimes given an incorrect answer.

3.2.1. Arisen Problems with Task Planning with Movable Obstacles

Before diving into arisen problems, the joint configuration space and the piecewise-analytic configuration space are defined. A *joint configuration space* of the robot and the obstacles is created by augmenting the robot configuration space with every configuration space for all objects. For example, if the configuration space for both robot and objects consist of position X, Y and orientation θ, then the joint configuration space is $3n$-dimensional, where n is the number of objects including the robot. When the robot manipulates an object, certain multi-body constraints are applicable and are different from single-body constraints, a configuration space containing multiple modes where different constraints apply is called an *piecewise-analytic* configuration space [**goldberg_asymptotically_2020**]. The joint configuration space with multiple modes of dynamics is an example of a piecewise-analytic configuration space.

In this literature, the task planner must solve a NAMO problem with unknown but learnable true dynamics. Directly planning in joint configuration space of the robot and objects is tedious for NAMO because of different modes of dynamics. For example, the robot can be driving, then pushing and then driving. The push action influences the free space where the robot has to drive in if the push manipulation has ceased. With multiple objects, directly planning in a joint configuration space causes a exponential number of possibilities.

With a given task, the target poses of certain objects are given, but that does not specify the location of other objects which may be present in the environment, such as the location of the sphere in the example displayed in figure 3.4. If the final positions of some objects are unspecified, this leads to problem dimensionality that is exponential in the number of objects with unspecified target positions in the environment [**scholz_navigation_2016**], and is known to be NP-hard [**reif_motion_1985**].

The two reasons above are indicating the challenges an task planner is facing when solving the NAMO problems, both planning in a piecewise-analytic configuration space and trying to solve an NP-hard problem also occur with known dynamics. This literature assumes unknown objects, single-body systems dynamics are a priori not or party known, a priori the dynamics of multi-body systems are unknown. Thus planning has to be performed with learned dynamics which is important to keep in mind, because of estimations of the true dynamics motion and manipulation planning might return unfeasible paths, and feasible paths might not be found. Compared to known dynamics, learned dynamics adds a large layer of uncertainty to the NAMO problem.

3.2.2. Planning in Joint Configuration Space

As already indicated in section 3.2.1, 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. Even sampling cannot computationally find a path in reasonable time, only by leveraging simplifications a search be performed. The upcoming solutions to search in the joint configuration space all implement some simplification to prevent sampling the entire joint configuration space.

Note that while other task planning methods might use motion/manipulation planning to validate motions/manipulations, planning in joint configuration space does not. Planning in joint configuration space essentially is motion/manipulation planning for a longer horizon than a single action.

Novin [**sabbagh_novin_optimal_2016**] develops a task planner for the NAMO problem combined with placing objects on target positions which is capable of finding a local minimum in the joint configuration space. Tackling the combinational explosion using 2 different tactics, first, a disjunctive programming concept is applied to convert the continuous problem to discrete form, where a continuous path is equivalent to some points with equal time distance in between, second, a heuristic function

is implemented, which allows for planning toward the goal for a fixed number of time steps. Such a receding horizon allows only to search a path close to the current joint configurations. Planning toward configurations lowering a metric function indicating the direction toward the final target configuration. Convex optimisation then finds an optimal path regarding the considered horizon [**sabbagh_novin_optimal_2016**]. Novin then finds the NAMO problem while learning dynamical properties in a hospital setting. Where patients are assisted by the patients' assistant mobile robot which can move obstacles out of the way or hand patients their walker. In addition to the task planner, a Bayesian regression algorithm was used to estimate the object dynamical model and an MPC-based controller was used to follow a specified path [**novin_dynamic_2018**]. In a follow-up paper, trajectory errors have been lowered and the selection of objects to encounter has been enlarged [**sabbagh_novin_model_2021**]. Novin has shown that by sampling close to the current configuration, a path can be found in high dimensional configuration spaces in reasonable time. The patient assistant mobile robot is able to find and track a path in real-time.

[**goldberg_asymptotically_2020**] solves the NAMO problem by first extending existing algorithms developed by [**hauser_randomized_2011**] to an optimal but prohibitively computationally expensive algorithm. Then the configuration space is factored while preserving optimality, this reduces the complexity considerably, by considering only a finite collection of subsets of the configuration space, each of which is subject only to analytic constraints. By building a graph on each of these subsets and connecting the resulting collection of graphs, we can construct a random graph that spans the configuration space. As the collection grows sufficiently large, it will contain a near-optimal plan with probability one and an optimal plan in the limit of infinite samples.

The 2 papers discussed have shown that the computational nightmare of the joint configuration space can be tamed by techniques such as discretization, factorization or a heuristic function combined with a time horizon. 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. A relief bonus for solving the NAMO problem in the joint configuration space which requires much computation power is that it removes the need for individual motion or manipulation planning. It can be concluded that if clever techniques keep the dimensionality to an reasonable small subspace, such that the robot can start tracking a path in under a minute of searching, path searching in the joint configurations space is a successful methods for task planning.

3.2.3. High-level Planners

High-level planners, in this literature refers to an ontology defining the structure of knowledge in a certain domain and a planner which when queried with a task or question uses the ontology to derive an answer or action sequence. Advanced high-level planners automatically try to satisfy 'logical' constraints that for humans feel obvious such as holding a cup upright if it is filled with liquid. A summary of high-level languages, planners and frameworks specialised for robotic applications will now be discussed, for this discussion, the literature study of M. Mâachou has been used containing a recent summary of the high-level planners [**maachou_mohommed_knowledge-based_2021**].

Note not all high-level planners *must* use an *ontology*, many however do, to categorise objects and actions as concepts, roles and instances. An example concept is the set of mobile robots with grippers, Tiago (a mobile robot with grippers) is an instance of the concept mobile robots with grippers, apple_1 is an instance of the concept fruit and can be gripped by Tiago, then the role Tiago.canGrab(apple_1) indicates that Tiago can grab apple_1. Ontologies create hierarchical categorisations for example, a categorisation of products in a supermarket where chocolate croissant is a subconcept croissant, a subconcept of bread. Most popular languages to store an ontologies are the Planning Domain Definition Language (PDDL) and the Web Ontology Language (OWL). Ontologies cannot categorise and relate every action, concept or physical object existing in the real world, that is way too much data. That is why ontologies are specific to a certain domain such as a supermarket environment, an hospital environment or a family home. Large domain-specific ontologies are available open-source, for robotics applications the most popular ontologies are Suggested Upper Merged Ontology (SUMO), Core Ontology for Robotics and Automation (CORA) and Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE).

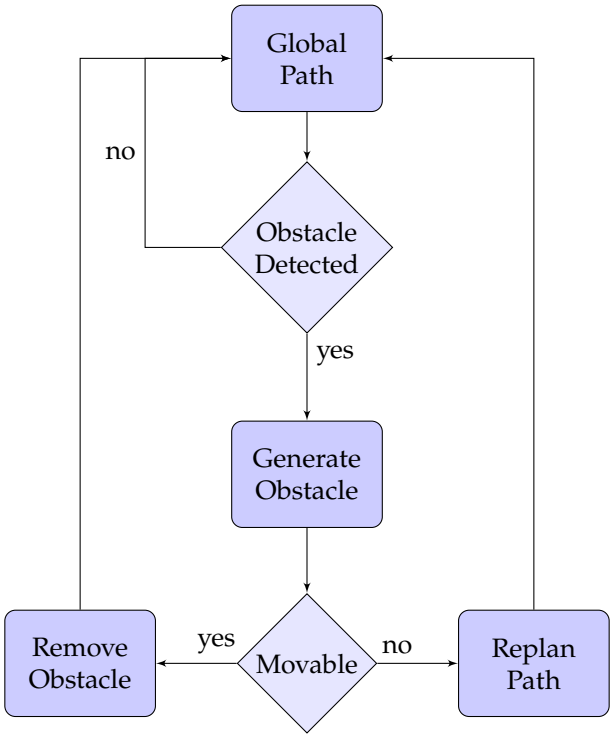


Figure 3.5: Flowchart representation of navigation algorithm [**wang_affordance-based_2020**]

Ontologies are deterministic, the closed-world assumption (note, not the same assumption as assumption 1) is used to make unknown statements decidable, the closed-world assumption states, anything not known to be true is assumed to be false. While learning dynamical properties many statements are unknown, with the closed-world assumption high-level planners mainly would assume manipulation to be impossible, for example, if the high-level planner is queried with "can the unknown object be moved", the answer would simply lead to false because it is unknown. Here *affordances* can help to make an estimation, affordance is defined as the ability to perform a certain action with an object in a given environment [**ardon_affordances_2020**]. Concluding that a certain manipulation action would lead to success with an unknown object requires some experience before any conclusion can be drawn. Actions which collect experience for a given object should be triggered in high-level planners to update the knowledge base which then bypasses the closed-world assumption.

Frameworks (which include a ontology, planner and more) such as KnowRob [**beetz_know_2018**] or the perception and manipulation knowledge [**diab_pmkknowledge_2019**] show the effectiveness of high-level planners in robotic applications.

High-level planners in practice are successful in generating an action sequences, however, in the scope of this literature study, where tasks are defined as a set of objects with target configurations high-level planners are not needed. There is not classification of objects required, no distinction between physical places such as kitchen and living room. An high-level planner in the scope of this literature is thus overkill. Simple heuristic and flowchart like decisions trees, see figure 3.5 achieve the same performance as expensive planners, thus the former is preferred.

3.2.4. Hierarchical Planning

Hierarchical planners separate the robot's configurations space into subspaces. In such graphs where the subspace is a section where a single dynamical mode holds true, such as driving, pulling or pushing. During seperations into subspaces, actions are generated and a graph (such as a Markov Decision Process (MDP) or search tree [**bronson_practical_2010**]) combines actions with subspaces, where the nodes correspond to robot and object poses, and transitions to actions. Solving a task planning problem

is then defined as finding a list of transitions which start in the current configuration (contained inside a subspace) and end in the target configuration (inside a subspace), an example is displayed in figure 3.6.

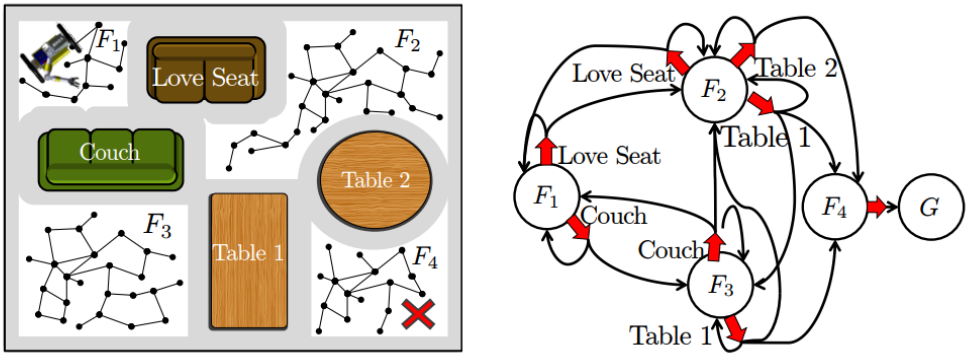


Figure 3.6: Schematic overview of the robots' free space separated is subgraps in PRM and the corresponding MDP, from **[scholz_navigation_2016]**

Figure 3.6 displays a schematic overview of a separated configuration space of the robot, free space is made into 4 subspaces, F_1, F_2, F_3 and F_4. Detecting subspaces is found by finding disconnected connectivity graphs after a limited number of samples, and objects inbetween disconnected graphs can be extracted based on position relative to the sampled configurations. An MDP can then be constructed and is viewed as a reward shaping mechanism to focus the low-level search on actions that are likely to clear paths to useful locations **[scholz_navigation_2016]**. Solving the MDP generates multiple ordered action sequences which would lead to the robot's target configuration.

Task planning with learned dynamics was first done by Scholz **[scholz_navigation_2016]**, the work has inspired figure 3.6 and it was the first attempt to solve the NAMO using learned dynamics **[scholz_learning_2015]**. By using a hierarchical MDP formulation of the NAMO problem designed to handle dynamics uncertainty, Scholz successfully demonstrated the ability of a robot to adapt to unexpected object behaviour such as a table which upon inspection turned out to be unmovable. A key difference between Scholz task description and this literture review's task description is that in **[scholz_navigation_2016]** the target position for only the robot is given, where as this literature includes possibly some target positions for objects. Including target positions for the obstacles makes such a problem quite a bit harder since additional planning is required, it is then unclear if such a problem can be solved using sampling algorithms combined with an MDPs.

Instead of defining actions and subgraphs as previously shown, *backward induction* determines the actions, and applies them to the target configuration to search for the starting configuration. An arisen problem is the unspecified target positions of the objects in the environment. A workaround is to simplify the NAMO problem to object rearrangement for which all start and target configurations are specified. Krontiris and Bekris created such conditions in object rearrangement problem in which object move over a straight line, and can only be moved once (monotone problem) **[krontiris_dealing_2015]**. A graph is created where nodes corresponding to the objects pose in the world. From target node to the current node is searched in a backwards fashion. With the monotone rearrangement search algorithm constructed by **[krontiris_dealing_2015]** finding a solution is sublinear in the number of objects to rearrange and the algorithm is run-time applicable.

Generally target positions other than the robot itself are not specified in NAMO problems. Unspecified target positions can be randomly chosen, as **[siciliano_path_2009]** has shown. By augmenting a search tree with manipulated objects to random locations, a special feature is that **[siciliano_path_2009]** keeps track of the robot and object configurations within the search tree. Randomly selected actions applied to randomly chosen actions guarantees probabilistic completeness for infinite sampling, but the computational power required increases drastically with the size of the number of objects and

size of the workspace (and thus directly configurations space). Heuristic functions could potentially prevent unnessecary sampling into unrelevant subspaces of the configuration space, such as [**sabbagh_novin_model_2021**] has shown.

Heuristic task planners find solutions which are *hierarchical*, they will return the best feasible plan that can be expressed within the task hierarchy they search [**goldberg_asymptotically_2020**]. System models contain some model mismatch with the true dynamics. Both the hierarchical solutions and effects of model mismatch are acceptable mainly because the NAMO problem is NP-hard. Hierarchical planners are relatively computationally efficient, compared to planning in joint configuration space and find a action sequence (even though that might be hierarchical). It can be concluded that hierarchical planners are thus a right fit to determine action sequences in the scope of this literature.

3.3. Discussion

sample- and graph based motion and manipulation planning algorithms have been discussed, the most applicable motion planning algorithms are the sample-based algorithms because they can search high dimensional spaces efficiently compared to graph-based planners. a connectivity graph build on sampled configurations indicates if samples are reachable from other sampled configurations while satisfying constraints imposed by the local planner. Feasibility has been shown to depend on local planners, who's accuracy directly depends on system models discussed in previous chapter. both path existence and convergence properties have been discussed, which allows to conclude that sample-based planners can efficiently search for motion and manipulation problems, additional checks such as path existence can speed up arriving to conclusions, such as "no path exists".

The joint configuration space is a piecewise-analytic configuration space containing multiple modes of dynamics. Because this joint configuration space dimensionality grows linearly (and the configurations space thus exponentially) with the number of objects, it is computationally infeasible to search for a path. Especially considering that with robotic application solutions found in real-time are highly appreciated. The exponentially fast growing joint configuration space is an answer to the second research subquestion.

Three main methods have been discussed to determine an action sequence to reach a given task, firstly planning in a joint configuration space where the dimensionality grows so enormously fast that measures have to be taken in order to make the problem computable. Using techniques such as discretisation, a finite planning horizon or factorisation decrease the search space such that real-time planning is possible, allowing to conclude that planning in joint configuration space is effective, and applicable for task planning with learned dynamics. Secondly, high-level planners, which with an ontology and planner determine an action sequence. The task is defined as a set of object and corresponding target configurations, this form of task and the lack of categorisation of objects yields high-level planners shy of providing effective solution. To conclude high-level planners are not fit for determining a action sequence in this literature. The last methods is hierarchical planning, by seperating the joint configuration space in local subspace where one mode of dynamics holds, motion planning algorithms can find a paths in these subspaces, a global plan is found by generating a search tree or MDP from planning to target. The solutions found are hierarchical, which limits converging to a global minimum, even so the methods is efficient compared to a search into joint configuration space. With learned, possibly inaccurate dynamical models, converging toward the optimal path is not the goal of the task planner, and hierarchical solutions with computational efficiency are preferred, allowing to conclude that hierarchical solutions are effective and applicable approach to find an action sequence for tasks to execute in an environment with movable obstacles.

4

Proposed Method

The goal of this chapter is to provide a description of the proposed algorithm and to list tests with expected outcomes which, if implemented provide evidence for the effectiveness of the algorithm. The previous chapter has answered the first 2 research subquestions by showing existing methods and their limitations which have been highlighted. This chapter answers the last research subquestion by providing a draft method to be investigated in the thesis, the method can complete tasks while learning system dynamics during task execution in an environment with movable obstacles. The proposed algorithm relies on many existing methods which have been discussed in chapters 2 and 3 and will be rediscussed briefly in section 4.1. Section 4.2 defines the hypothesis graph, as section 4.3 defines the knowledge graph. Benchmark tests are listed in section 4.4. The Discussion is presented in section 4.5.

In this literature study the focus lies on the knowledge graph (KGraph), the hypothesis graph (HGraph) and the robot simulation environment. If the literature study focus would be complemented with a high-level planner and an ontology, the robot framework would be theoretically capable of handling high-level tasks, such as grouping objects based on shape, or cleaning.

general outline

Figure 4.1 displays the general outline control structure for the robot. Upon receiving a task, the *hypothesis graph* is responsible for generating a hypothesis, an action sequence which might lead to successful completion of the given task. Replanning may occur after learning objects are immovable or when control fails to complete a subtask, if all possible hypothesis have been tried, then HGraph concludes that the task cannot be done. Section 4.2 is fully dedicated to explaining the hypothesis graph.

After execution of an action, the action is evaluated and newly learned knowledge is stored or updated in the *knowledge graph*. Aside from storing knowledge, the knowledge graph provides action suggestions to the hypothesis graph. section 4.3 is fully dedicated to explaining the knowledge graph.

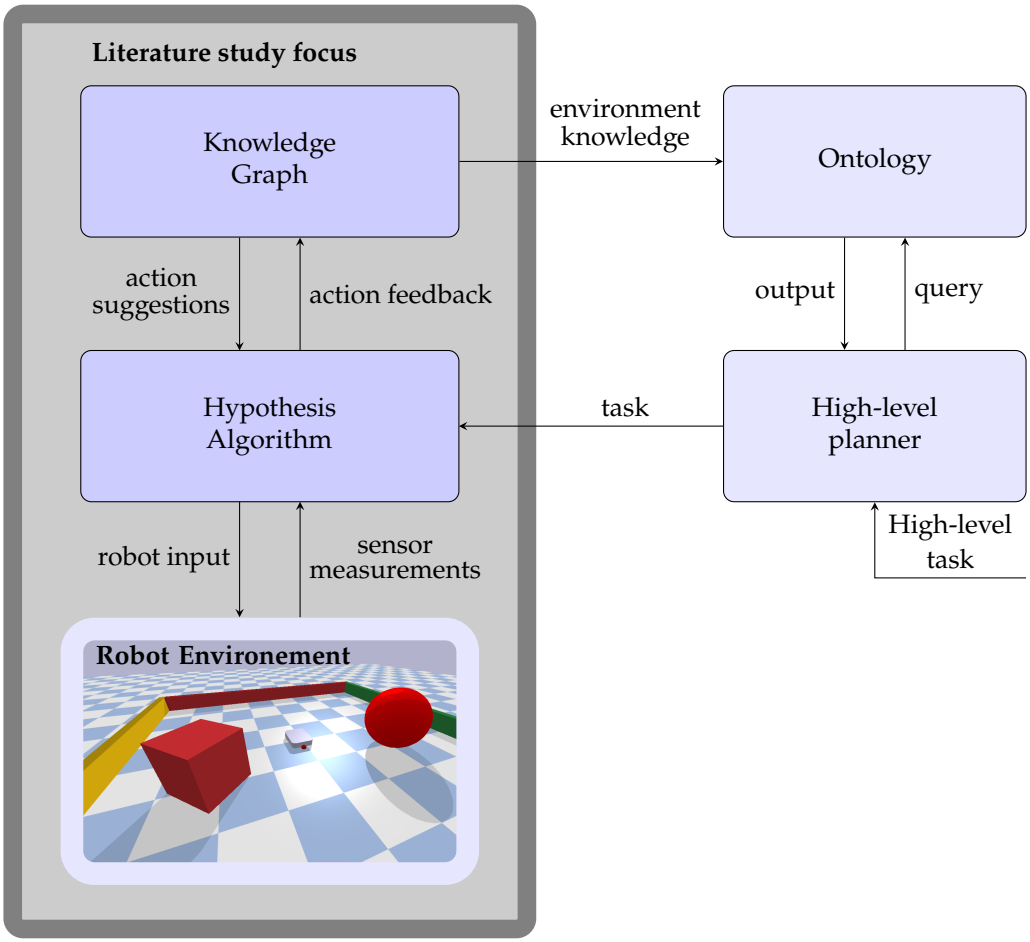


Figure 4.1: Complete control scheme in Flowchart representation.

4.1. Required Components

This section lists required components used by the proposed method, such as: path-finding algorithms, controllers or system identification methods. The required components are neatly grouped in this section, every component has a specific citation and it is indicated if any modifications must first be applied before it can be used by the proposed method. The required components are minimally explained, and their function and responsibilities are highlighted.

Path Non-Existence Before motion planning, the HGraph checks path non-existence, more information on path existence can be found in section 3.1.3. As opposed to motion or manipulation planning two simplifications are made: an obstacle (such as the robot or an environment object) is assumed to be holonomic, and only the unmovable objects are actively present in the configuration space, unknown and movable obstacles are ignored. In configuration space, the free space is discretised with a cell size based on the geometry of the robot or object, a graph-based planner then searches to find a path from starting cell toward the target cell, figure 4.2 displays a visual explanation of the path non-existence checker. Paper [**akella_simple_2008**] describes the method applicable to prove path non-existence, if a path is found toward the target position, the cells containing a path will be converted to sample points in configuration space, serving as a warm start for the motion or manipulation planning algorithms.

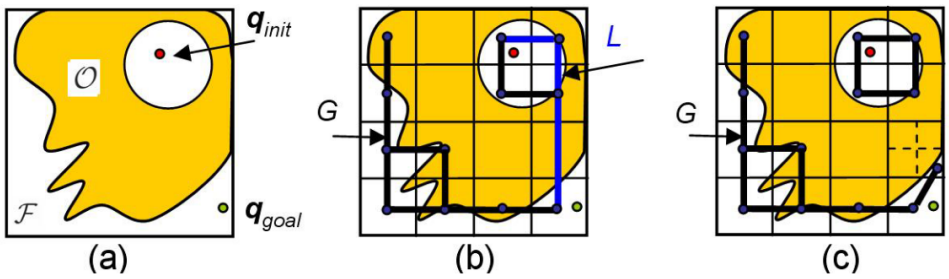


Figure 4.2: Path non-existence between q_{init} and q_{goal}. (b): A connectivity graph G is built. The path L, which connects the cells including q_{init} and q_{goal}, is computed from G. Any mixed cell along L is further subdivided. (c): In the new connectivity graph, the cell containing q_{init} and the cell containing q_{goal} are not connected. This concludes that there is no collision-free path between q_{init} and q_{init}, From [**akella_simple_2008**].

System identification and Control In section 2.2 various system identification en control methods have been discussed. Some methods are appropriate candidates for a nonholonomic robot and an environment with movable obstacles. The control and system identification methods are not discussed again here, since they have already been discussed in section 2.2. The system models generated by system identification methods act as forward models or local planners during motion and manipulation planning as discussed in sections 3.1 and 3.2.

Appropriate candidates for single-body control with system identification methods:

- MPC control and PEM [**verhaegen_filtering_2007**] system identification
- MPC control and IPEM [**seegmiller_vehicle_2013**] system identification

A parameterisable system model improves its model accuracy using the PEM (offline) or IPEM (online) method, the MPC controller however keeps constant parameters such as control horizon and the weight matrices. To have some variation in these constant parameters, the robot has access to multiple MPC controllers with various control parameters.

Appropriate candidates for multi-body system identification and control are:

- Reactive MPC which requires a single- and multi-body model [**toussaint_sequence–constraints_2022**]
- MPC controller and model fitting a 3D-Gaussian [**mericli_push-manipulation_2015**]
- MPPI controller and uncertainty calibrated forward model [**arruda_uncertainty_2017**]
- RMPPI controller and a LSTM [**cong_self-adapting_2020**]

Motion and Manipulation planning Finding paths between starting and target configurations is performed by a sampling-based method, a double tree RRT* search algorithm [**chen_fast_2018**], which searches the configuration space for an optimal path toward the target position. A connectivity graph, (initially only the start and target configuration) keeps track of the configurations reachability from the start and target configuration. Randomly sampled samples are compared to some nearest nodes inside the connectivity graph, before adding the newly sampled node, a check is performed by a local planner to validate if constraints are satisfied, if the random sampled configuration is not valid, it is discarded. The connectivity graph grows from the target configuration and the start configuration, when the connectivity graph from start to target is connected a path is found, pseudocode can be found in algorithm 1.

Manipulation planning is similar to motion planning. With motion planning, planning for the robot is performed, with manipulation planning, planning for the objet to push is performed (and the robot is neglected, but kept track of). So manipulation planning only happens as motion planning for the pushable object.

The robot configuration is manipulation planning is kept to validate the reachability to newly sampled samples. To elaborate, adding a new sample is accomplished by, sampling a new configuration for the object to manipulate, the sample is placed in configuration space, with a manually tuned metric function samples nearby the new sample are gathered. A local planner validates if these gathered samples are reachable from the new sample. One configuration is reachable from another configuration if for some input and some system model, one configuration becomes the other configuration in a small amount of time (e.g. < 5 time steps) an example of such a system model is displayed in equation (2.1). The difference in predictor between motion (single-body predictor) and manipulation planning (multi-body predictor) thus resides in the local planner and where local motion planner requires 2 configurations, manipulation planning additionally requires at least one robot configuration. To check if 2 configurations are reachable for manipulation planning 2 object configurations and 1 robot configuration is required, if reachable and connected, an extra robot configuration is generated as a result of the local planner, which is needed for checking reachability of future samples, an visual example is displayed in figure 4.3. A manipulation path from start to target will have both the objects and the robot's configurations, where the robot configurations only serve the local planner to validate if two object configurations can be connected. Additionally the robot's path should not collide with obstacles, which needs to be checked during planning.

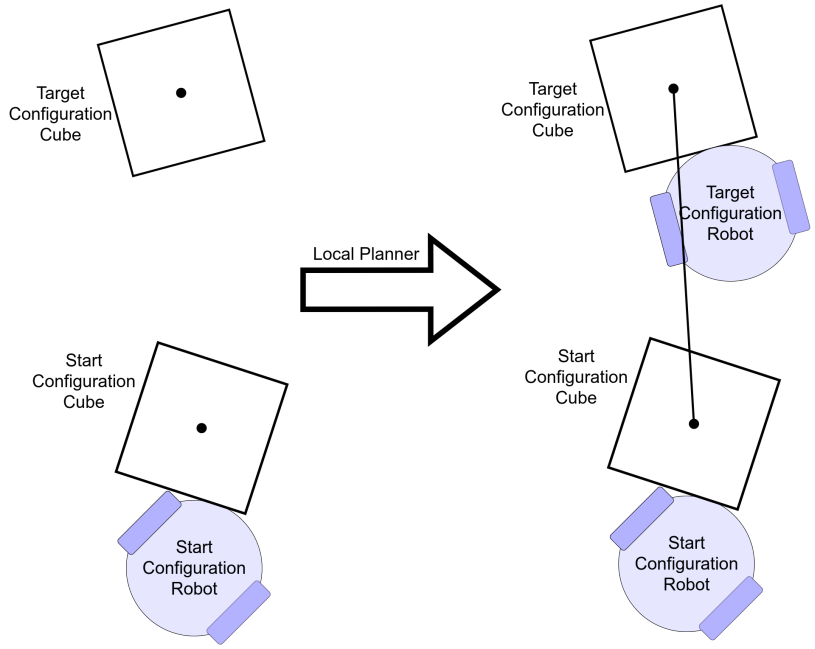


Figure 4.3: Local planner connecting 2 cube configurations and generating an new robot configuration

The RRT* algorithm takes a cost for path length into account, resulting in finding the shortest path with infinite samples if a path exists, an additional cost is added if the path overlaps with a movable or unknown obstacle, motivating the algorithm to find a path around unknown or movable obstacles, but to plan through the obstacles planning around is impossible or costs more. If a path overlaps with an obstacle, a subtask is created to first remove the obstacle, and then continue to track the path. Figure 4.4 gives a visual view of the proposed RRT* algorithm.

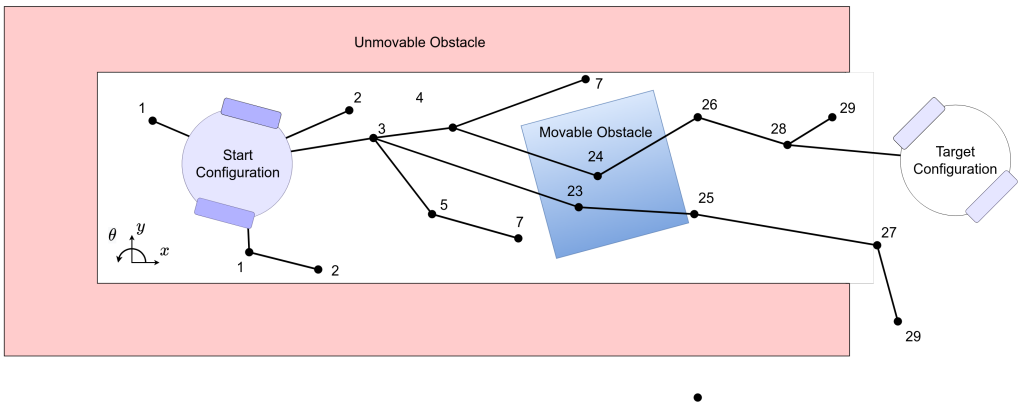


Figure 4.4: Schematic view of the proposed double RRT* tree taking movable and unknown obstacles into account with cost to reach a sampled configuration displayed.

Note that it remains hard to make predictions on the height of the cost because the object has unknown dynamics, which is why there is a fixed cost for unknown and movable obstacles, unmovable obstacles are captured by the obstacle space.

Algorithms algorithm 1 displays the pseudocode for the double RRT* algorithm which takes unknown and movable obstacles into account. The following definitions are used by the RRT* algorithm.

V: A set of vertices
E: A set of edges

The following functions are called by the algorithm 1.

x_{init}: Creates a start and target configuration.
$NotReachStop$: Returns true if the stopping criteria is not reached.
$Sample_{free}$: Creates a random sample in free space, free space includes the movable and unknown obstacles.
$Nearest(x, V)$: Finds the nearest vertices using euclidean distance
$NearestSet(x, V)$: Find a set of nearest vertices using euclidean distance
$Project(x, x')$: Project x' toward x such that it lies close enough to a vertices to be compared using a local planner
$CollisionCheck(x)$: returns true if x is in free-space, movable obstacle or unknown obstacle
$CostFromInit(x, x')$: Find the total cost from x to the initial vertices via x', cost is determined as a sum of path length and if the path overlaps with movable of unknown objects.
$ConstraintsCheck(x, x')$: return true if a local planner is able to connect x and x' using a forward model.

Algorithm 1 Proposed Double tree RRT* algorithm taking movable obstacles and constraints into account, edited RRT* pseudocode from [**chen_fast_2018**]

$V \leftarrow x_{init}$
while $NotReachStop$ **do**
  $Cost_{min} \leftarrow +\infty$
  $x_{rand} \leftarrow Sample_{free}$
  $x_{nearest} \leftarrow Nearest(x_{rand}, V)$
  $x_{temp} \leftarrow Project(x_{nearest}, x_{rand})$
  if $CollisionCheck(x_{temp}) == True$ **then**
    $x_{new} = x_{temp}$
  else
    $Continue$
  end if
  $X_{near} \leftarrow NearestSet(x_{new}, V)$
  for $x_{near} \in X_{near}$ **do**
    if $CostFromInit(x_{new}, x_{near}) < Cost_{min}$ **then**
      if $ConstraintsCheck(x_{new}, x_{near}) == True$ **then**
        $Cost_{min} \leftarrow CostFromInit(x_{new}, x_{near})$
        $x_{minCost} \leftarrow x_{near}$
      end if
    end if
  end for
  if $Cost_{min} != \infty$ **then**
    $V.add(x_{new})$
    $E.add(x_{minCost}, x_{new})$
  end if
end while

Notation standard For the upcoming sections 4.2 and 4.3, especially for the definitions of the hypothesis and knowledge graph a notation standard is used. A list of hypothesis and knowledge graph related symbols is shown in table 4.1.

Subscripts are used to indicate what the variable belongs to. For example, the state variable of the robot is written down as s_{robot}. Certain variables indicate the same object, but with different values, this is indicated by different superscripts. For example s^1_{robot} and s^2_{robot} are different states but both indicate the robot's state, formally $||s^1_{robot} - s^2_{robot}|| \neq 0$. k indicates the time step, for example, the state of the robot at time step k: $s_{robot}(k)$. For convenience, the time step index is sometimes dropped.

k	time step index
$s_{id}(k)$:	State
$c_{id}(k)$:	Configuration
$C_{id}(k)$:	configuration set
$ob_{id}(k)$:	Object
$O_{id}(k)$:	Object set
V_{id}^{ob}:	node storing set of objects
V_{id}^{conf}:	node storing set of configurations
$V_{id}^{\Delta conf}$:	node storing set of configurations and boolean lists
d:	dynamical model
α:	Success factor
$\tau_{(i,j)}$:	Transition
$G^{knowledge}$:	knowledge graph
$G^{hypothesis}$:	hypothesis graph
h:	plan, hypothesis
$path$:	list of configurations indicating a path, or False

Table 4.1: List of symbols used in the hypothesis- and knowledge graph, where id is a unique identifier of the object

4.2. Hypothesis Graph

When given a task the hypothesis graph starts generating action sequences, to give an overview of how such action sequences are generated, figure 4.5 was created. An HGraph can start generating hypothesises or action sequences when a task and the environment are provided. The HGraph alternates between executing and replanning to complete the task.

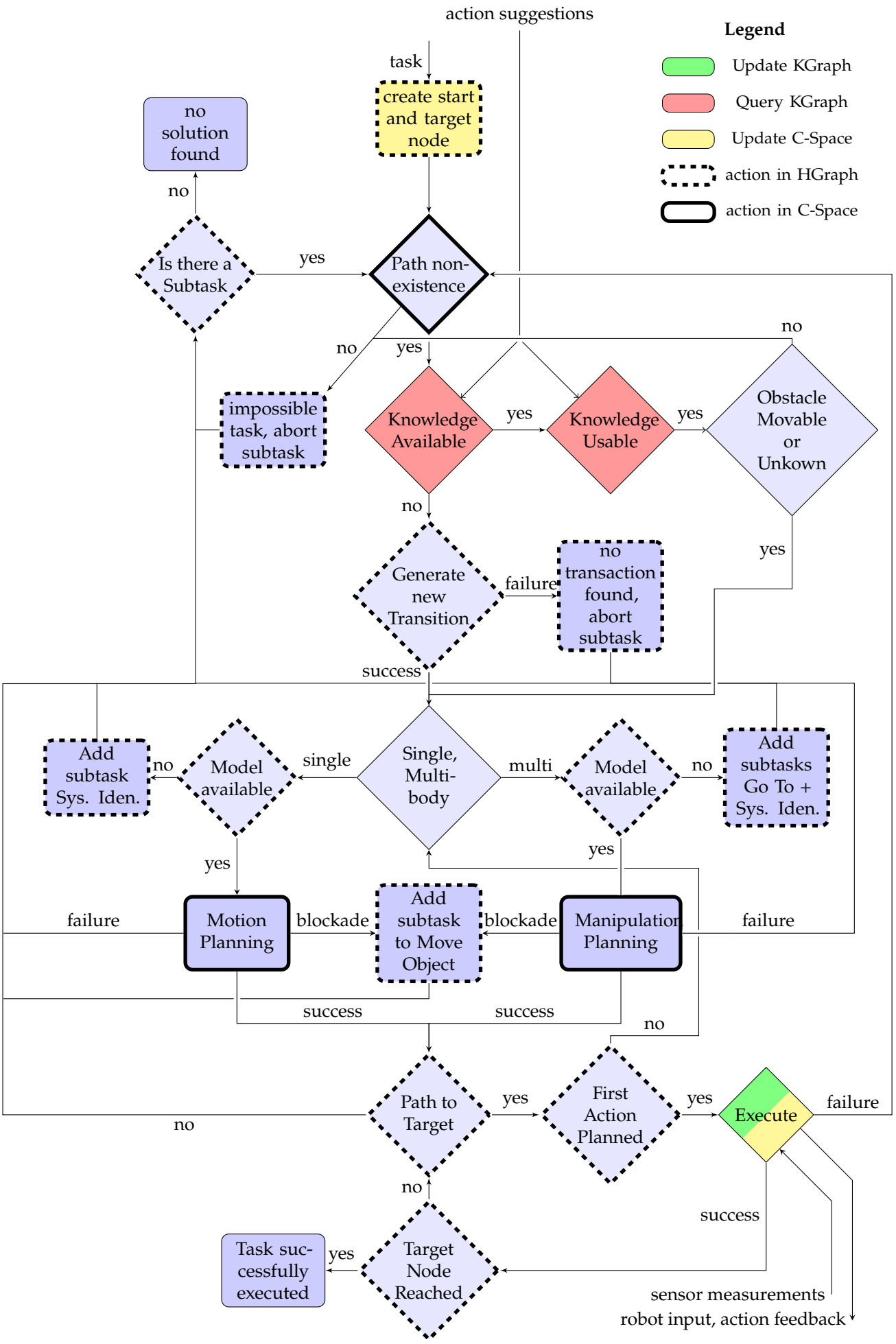


Figure 4.5: Flowchart displaying the hypothesis graph's workflow.

4.2.1. Definition

Every object in the environment with all its components is held in an unknown environment state $X(k)$. In a real application, sensors can partly observe this environment state with a certain measurement error. In a simulation environment and with the help of the full information assumption 3, the environments state can perfectly be measured.

A state describes the location, orientation and first-time derivative of an object's center of mass with respect to the environment's origin.
Formally, a **state**, $s_{id}(k)$ is a tuple of $(pos_x(k), pos_y(k), pos_\theta(k), vel_x(k), vel_y(k), vel_\theta(k))$
where $pos_x, pos_y, vel_x, vel_y, vel_\theta \in \mathbb{R}, \quad pos_\theta \in [0, 2\pi)$.

A configuration describes the configuration of an object. It is a subset of the state.
Formally, a **configuration**, $c_{id} \subseteq s_{id}$.

A configuration set is a collection of configurations of one or more objects.
Formally, a **configuration set**, $C_{id}(k) = \{c_1(k), c_2(k), \dots c_n(k)\}$, where $n \geq 1$.

An object holds the information about a object, such as state, configuration and shape.
Formally, a **object**, $ob_{id}(k) = (s_{id}(k), c_{id}(k), shape)$.
where $shape$ is linked to a 3D representation of the object, such as an urdf file and is used to construct the configuration space.
An object set is a collection of objects.
Formally, a **object set**, $O_{id}(k) = \{ob_1(k), ob_2(k), \dots, ob_n(k\}$, where $n \geq 1$.

A node is a set of objects with states or configurations. There are 3 types of nodes which can all be used interchangeably in the same graph. These are objectSetNodes, confSetNodes and changeOfConfSetNodes, their uses are to: keep detailed time-varying object information in a state (objectSetNode), keep only curcial time-invariant information in a configuration (confSetNode) or keep information about which subset of the state can be changed (changeOfConfSetNode).
Formally, a **node**:

objectSetNode, $V_{id}^{ob} = (id, \{ob_1(k), ob_2(k), \dots, ob_s(k)\})$ for some $s \geq 1$.

confSetNode, $V_{id}^{conf} = (id, \{c_1(k), c_2(k), \dots, c_s(k)\})$ for some $s \geq 1$.

changeOfConfSetNode, $V_{id}^{\Delta conf} = (id, (\{c_1, \mathbb{B}_1), (c_2, \mathbb{B}_2), \dots, (c_s, \mathbb{B}_s)\})$ where $\text{Dim}(c_i) = \text{Dim}(\mathbb{B}_i)$, with $\mathbb{B}_i$ a vector of booleans for some $s, i \geq 1$.

A transition or edge describes the details of how a node transitions to another node.
Formally, a **transition**, $\tau_{(from,to)} = (id_{from}, id_{to}, verb, controller, d, path)$ with $verb$ an English verb describing the transformation in configuration sets in human language, $controller$ the controller used for driving the robot, d dynamic model used by the controller, $path$ a list of configurations indicating the path connecting a start- to target node, if no path has been planned then $path = False$.

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

Transitions can also be system identification periods, system identification changes the environment because a train set drives or pushes objects. Every change in the environment has an accompanying transition.

Now the nodes and edges have been defined, the hypothesis graph can be defined. A hypothesis graph is a directed graph that includes the current configuration set and the target configuration set.
Formally, a **hypothesis graph**, $G^{hypothesis} = (V, E)$
comprising $V = \{V_i^{ob}, V_j^{conf}\}, E \in \{\tau_{(i,j)} | V_i, V_j \in \{V_i^{ob}, V_j^{conf}\}, i \neq j\}$.

Adding Transitions Two different nodes in the hypothesis graph can be connected with a transition, such a transition is parameterised by a controller and model. In section 4.1, a limited number of controllers and system identification methods have been listed. Controller selection is assisted by the knowledge graph, which recommends a controller and system model used in a comparable situation in the past. if the knowledge graph is unable to recommend a controller, selecting a controller is performed by uniformly sampling over the available controllers. If all the controller options have been tried, the generation of a new transition fails. The model or system identification method is depending on the controller and is thus selected based on the controller selection.

For example, the robot at starting position (start configuration space node) can drive with controller A using model a toward the target position (target configuration space node), and then (if randomly selected) the transition between the 2 nodes is parameterised with controller A and model a.

Backward Induction Adding transitions is done in a backward induction fashion. Meaning, transitions are added pointing toward the target or pointing toward a node having a path pointing toward the target node.

Transcending over a transition A hypothesis is a path in hypothesis graph from current to target node. Such a hypothesis might lead to completion of the task or subtask, because of unknown objects and uncertainty in the completion of transitions, there is not guarantee that the hypothesis leads to a completion of the task. If a hypothesis is found and is ready for execution, the first transition in the hypothesis is executed. Depending on the controller and the failure threshold, the execution leads to failure or success. Regardless of the outcome, the execution is reviewed and feedback is stored in the knowledge graph. Depending on the outcome of the execution the next transition in the HGraph is taken or the next transition is planned or path-non existence is checked, see figure 4.5. When executing, the environment is in a certain node, *the current node*. A transition in the HGraph represents a controller following a path, or system identification. System identification will create a model, even if during system identification faults occur, such as colliding with obstacles in the environment. It is expected that such a fault will decrease the system model accuracy significant.

Transition Execution Failure and Success While tracking of a path, predicted output is monitored. If the prediction output is crossing a predefined threshold for too long, execution will halt and the execution evaluates into failure.

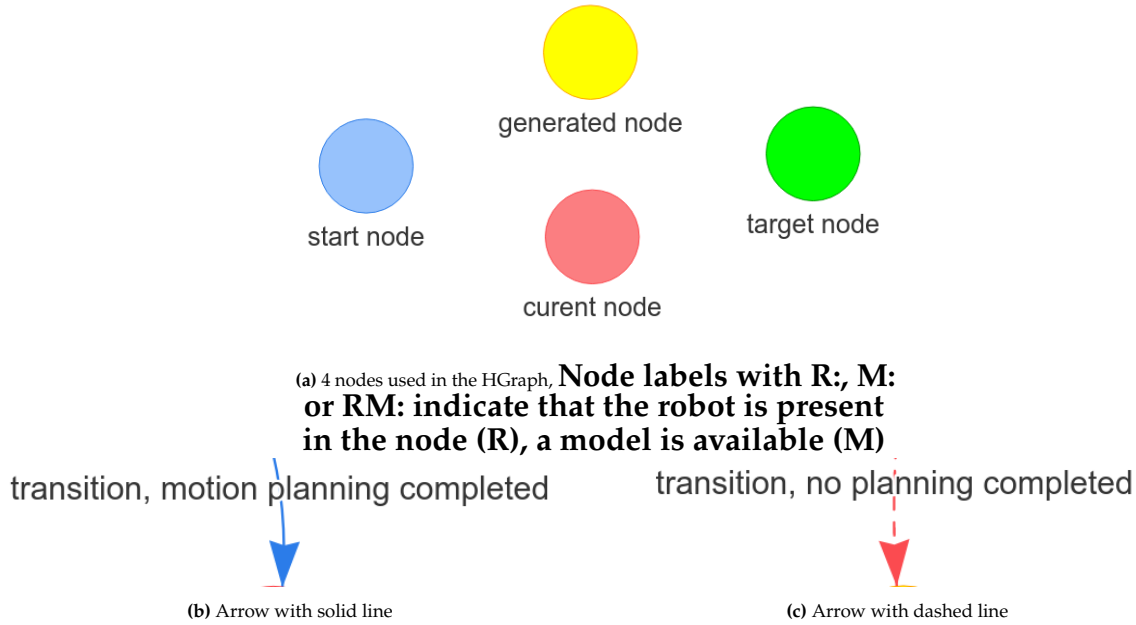
Hypothesis Graph Flowchart Let's walk through the flowchart displaying the HGraph workflow in figure 4.5, the HGraph is brought alive when a task is given (and the environment data is available). After the starting and target nodes have been created, the HGraph essentially alternates between 2 loops, the *creation loop* creates transitions and subtasks (in figure 4.5 from block Path non-existence to block Path to target and back through block Is there a Subtask). After checking path non-existence the knowledge graph is queried for action suggestions, a transition is generated or obtained from the KGraph, motion or manipulation planning occurs depending a model available and weather the transition is for a single- or multi-body control. Planning could yield new subtasks, these are added to the HGraph. If there is no path found from the current node toward the target node inside the HGraph, the creation loop keeps on generating new subtasks and transitions. Only a limited number of transitions and subtasks exists since there are a limited number of controllers and objects in the environment, eventually the Hgraph will thus halt. The *execution loop* executes transitions (in figure 4.5 from block Path to Target to block Execute and back via block Target Node Reached). If a an execution fails, replanning occurs by re-entering the creation loop. After every executed transition, action feedback is send to the KGraph.

If multiple hypothesis (path from current to target node in HGraph) fail during task execution, alternation between the creation and execution loop occurs. It is possible that a transition is generated,

fails during execution and later on is regenerated. To prevent regeneration of failed transitions, a blacklist is kept. Any transition applied to a node in the HGraph which failed is not allowed to reappear on any node containing the objects on which the transition failed. The blacklist is wiped clean after the HGraph halts. As example MPC control was unable to bring the Tiago robot from point A to B. Then during task execution this specific MPC controller is not allowed to control specifically the Tiago robot from any point to any other point.

4.2.2. Example HGraph

In figure 4.7 the robot is tasked with putting an object in a new position. Without any prior knowledge of the object or the robot dynamics, the robot must first create system models. The HGraph starts with adding nodes in a backward induction fashion connecting the target node to nodes until the start node is connected which is especially in figure 4.7c) and d) clearly visible.

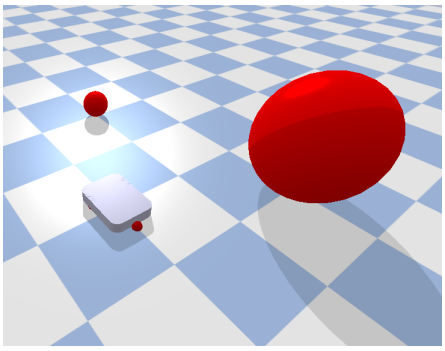


(a) 4 nodes used in the HGraph,

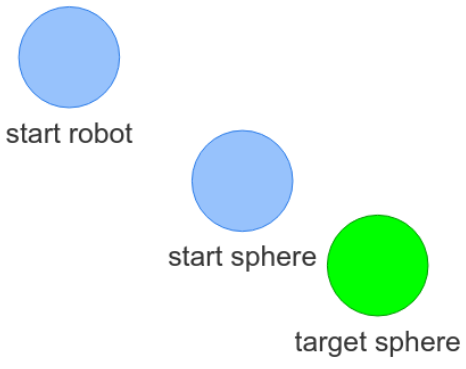
(b) Arrow with solid line

(c) Arrow with dashed line

Figure 4.6: HGraph Legend

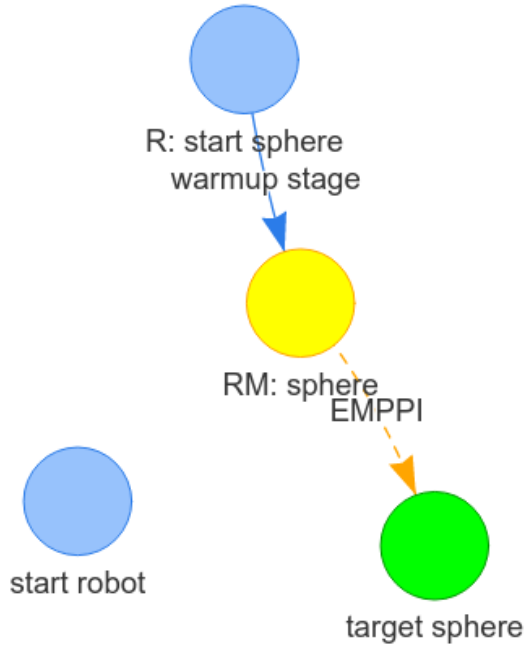


(a) Starting environment, including the robot, the large red sphere to push to the target position indicated with the small red sphere

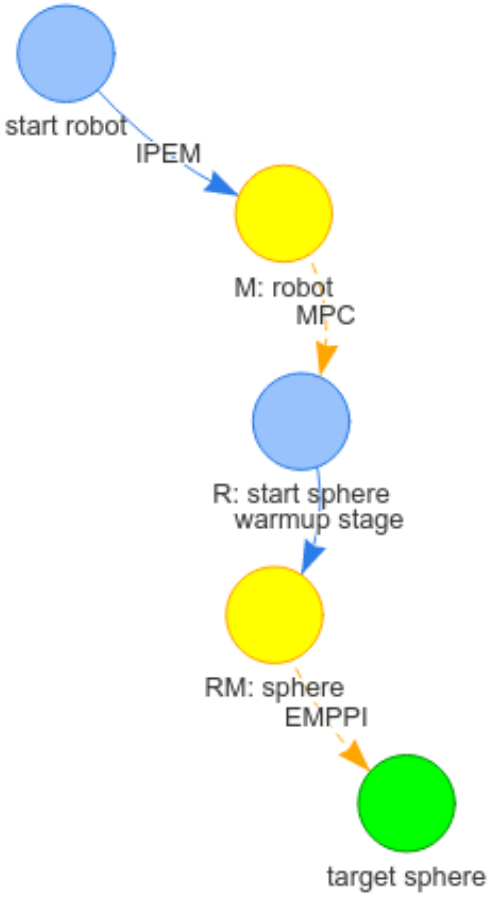


(b) Create initial start and target nodes

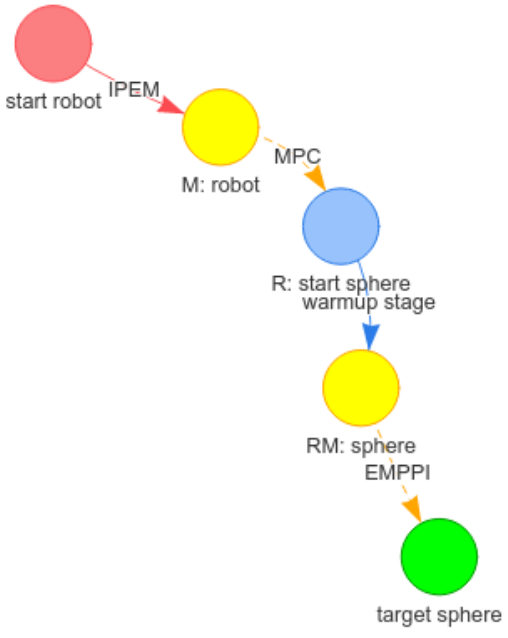
Figure 4.7: Figure continues on the next page



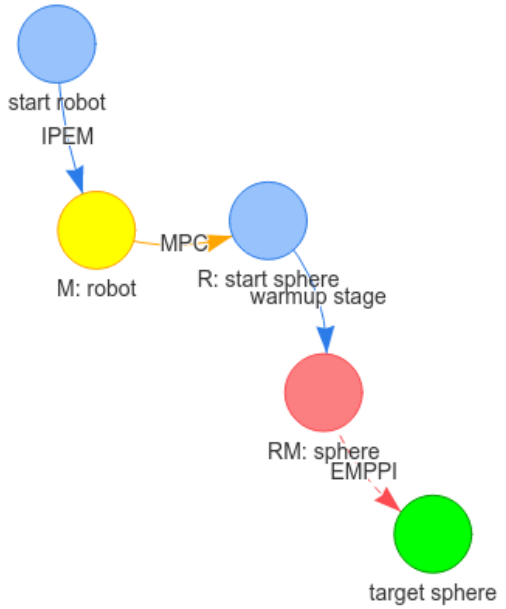
(c) Connect target sphere to starting node sphere with a randomly chosen control methods



(d) Connect start robot to start sphere because the robot is required for system identification



(e) Set start robot as current node



(f) Successfully execute first 3 transitions

Figure 4.7: The hypothesis graph successfully executing the first hypothesis generated. For the legend, see figure 4.6.

4.3. Knowledge graph

Transitions in the HGraph are executed and reviewed, the KGraph then act as memory to store the reviewed transitions using a $successfactor$. The success factor describes how transitions in the past have been executed, a high success factor means coming up high in the list of action recommendations.

4.3.1. Definition

A knowledge graph contains information about objects. $ObjectSetNodes$ describe the objects knowledge is known about, and transitions from $objectSetNodes$ point to $changeOfConfSetNodes$ indicating which part of the configuration was changed. The transition describes in detail how the objects in the object set can change their configuration set.

Formally, a **knowledge graph**, $G^{knowledge} = (V, E)$ comprising $V = \{V_i^{ob}, V_j^{\Delta conf}\}$, $E \in \{\tau_{(i,j)} | V_i \in V_i^{ob}, V_j \in V_j^{\Delta conf}\}$ where i and j are unique identifiers.

Reviewing an Executed Transaction Controllers and system models which produce desirable results (low prediction errors, many successfully executed transitions) should be used over controllers producing less desirable results. To keep track of how 'good' a transition was, feedback on a transition is introduced. Feedback consists of 2 parts, a transition succeeding or failing and the average prediction error. Feedback is stored, and converted to a success factor $\alpha \in [0.1, 1]$. Testing and fine-tuning α has not yet occurred, thus the following equation for the success factor has to be taken with a grain of salt.

$$\alpha = \begin{cases} (0.5 - \epsilon_{avg})^{1 - \frac{success}{success+fails}} & \text{if } \epsilon_{avg} < 0.4 \\ 0.1^{1 - \frac{success}{success+fails}} & \text{if } \epsilon_{avg} \geq 0.4 \end{cases} \tag{4.1}$$

Where ϵ_{avg} is the average prediction error over all stored prediction errors, $success$ is the number of times a transition was successfully executed and $fails$ is the number of times a transition failed.

Knowledge Graph Flowchart Let's walk through the KGraphs flowchart displayed in figure 4.8. Incoming action feedback (success/failure and average prediction error of an executed transition) is checked against existing knowledge in the KGraph and updated accordingly to equation (4.1). A query asking for an action suggestion is answered by an empty or ordered of action suggestions, the order is determined by the success factor. For the observant reader, there is no outgoing arrow named "environment knowledge" such as displayed in figure 4.1, which is left out on purpose since the ontology is not a part of the literate study.

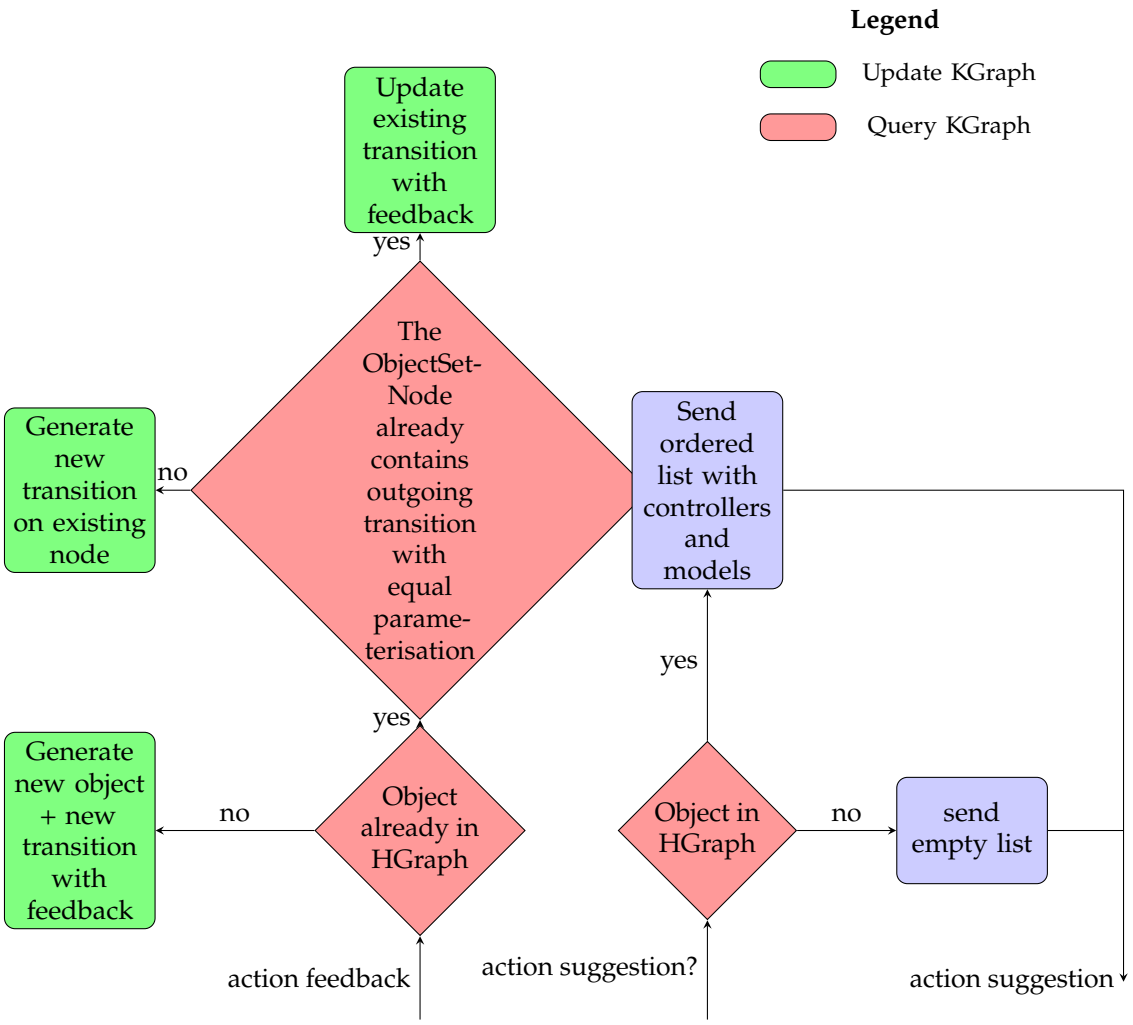


Figure 4.8: Flowchart displaying the knowledge graph's workflow.

4.3.2. Example KGraph

As a result of the example hypothesis graph displayed in figure 4.7, the following knowledge graph should emerge displayed in figure 4.9. The transition containing the EMPPI controller can be seen connecting the red sphere and robot to the changeOfConfSetNode which indicates that the position and velocity of the red sphere can be changed. And vice versa for the MPC controller.

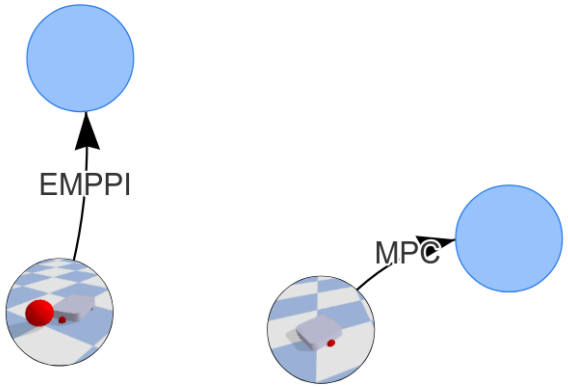
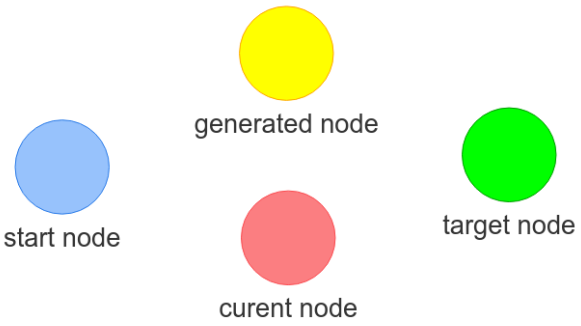


Figure 4.9: The knowledge graph after pushing a red sphere, this knowledge graph corresponds with the hypothesis graph in figure 4.7, ChangeOfConfSetNodes are displayed in blue, objectSetNodes display an image with the objects they represent.

4.4. Benchmark Tests

This section gives insight how the proposed method improves upon methods proposed by existing literature. An improvement is presented with an elaborate example, called a benchmark tests. The benchmark tests, BTests for short include an image of the starting environment, a description of the task, the final knowledge graph and a hypothesis graph which should be generated once the algorithm is implemented. The hypothesis graph may skip certain steps which are not the focus the BTest, and would only clutter the HGraph.

Once again the legend for a HGraph is displayed.



(a) 4 nodes used in the HGraph, **Node labels with R:, M: or RM: indicate that the robot is present in the node (R), a model is available (M)**

(b) Arrow with solid line

(c) Arrow with dashed line

Figure 4.10: HGraph Legend

4.4.1. Blockade by the Duck

Figure 4.11 displays a robot, 3 unmovable walls, a movable yellow duck and a movable red cube. **The task** given to the robot is to place the red cube inside the walls. The duck is "guarding" the target location of the red cube. The robot has to first detect, and then to take care of the duck by pushing it to a unspecified location such that the cube can pass through. It is assumed that there is enough space around the objects for system identification, alternatively system identification could be performed on the individual objects, and then perform the blockade by duck task. This BTest already involves many different aspects. Such as system identification on the robot itself, the red cube, the green wall and the duckling, motion planning, manipulation planning and replanning. The goal of this BTest is to give insight that the proposed methods can do all these different aspects.

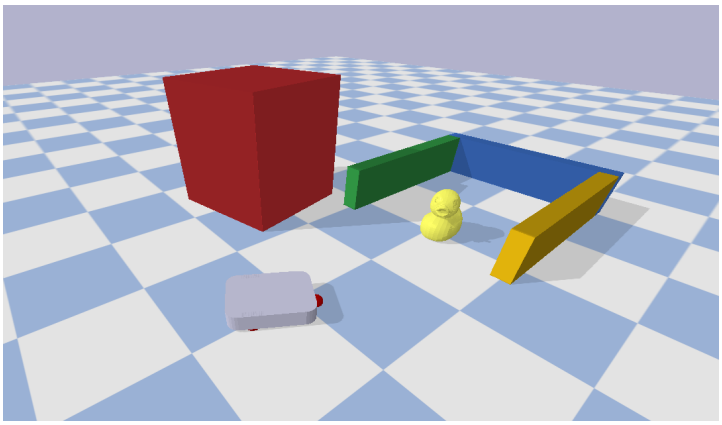
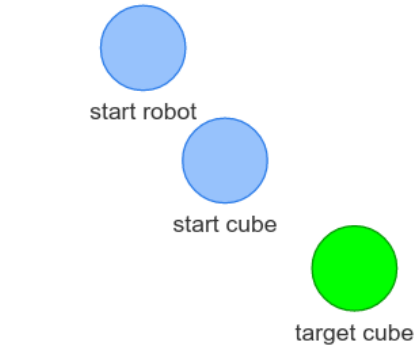
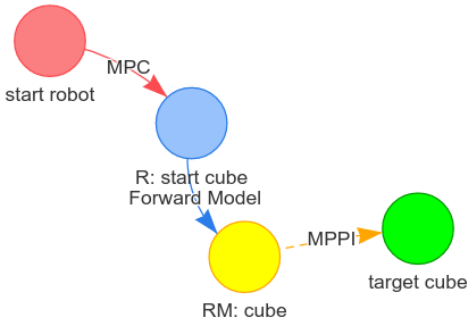


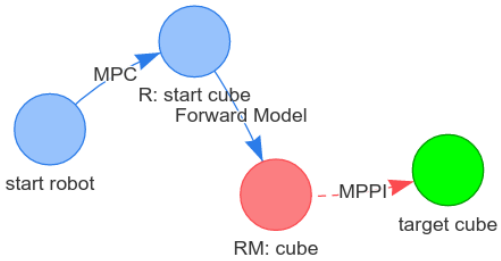
Figure 4.11: The robot is tasked with placing the cube inside the walls, only this is guarded by a duckling.



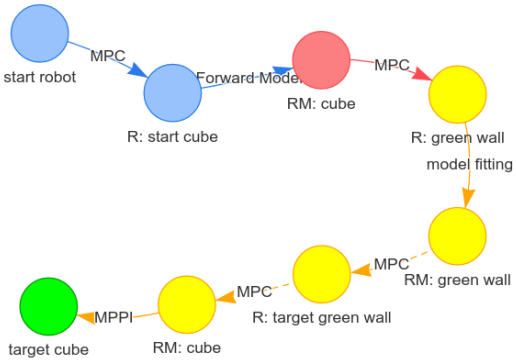
(a) Creating starting and target nodes



(b) Generate transitions and set start robot as current node



(c) Execute first 2 transitions

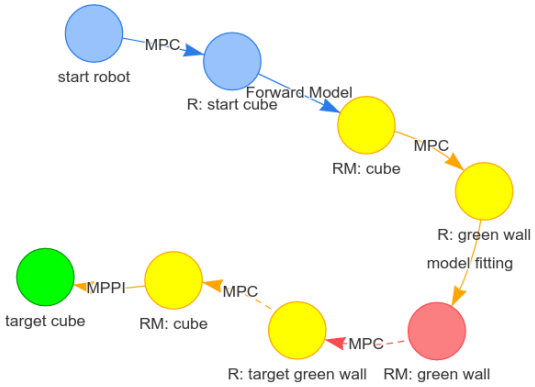


(d) In manipulation a subtask is created to move the green wall, these transitions are expected to complete successfully, completing the task

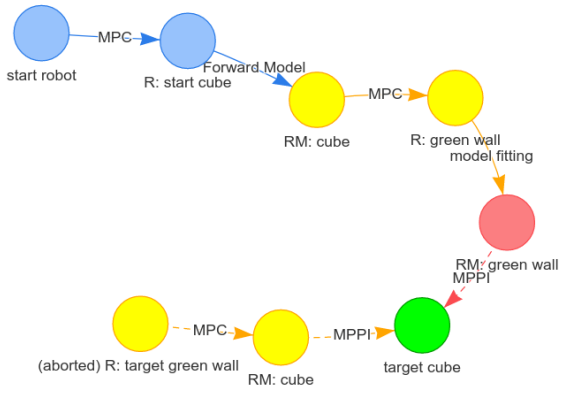
Figure 4.12: HGraph generating hypothesis to push the unmovable green wall and then push the cube to the target position. The figure continues on the next page. For the legend, see figure 4.6

As can be seen in figure 4.12, the HGraph has planned to push the cube toward the target position over the shortest path. On this path the (up to now) unknown green wall stands which needs to be moved before the cube can be pushed toward it's target position. The green wall needs to be pushed such that is does not intersect with the planned path for the cube.

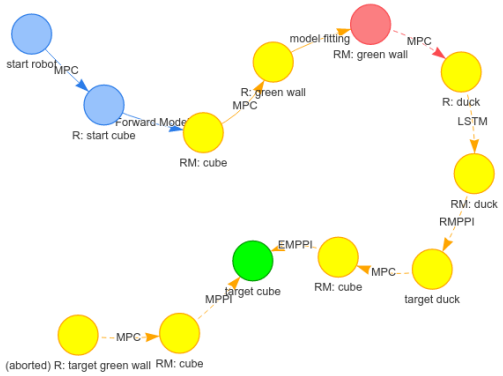
The HGraph displayed in figures 4.12 and 4.13 skips multiple steps in order to focus on the important steps. One of these skipped steps for example is system identification to find a model for the robot itself or the removal of insignificant nodes.



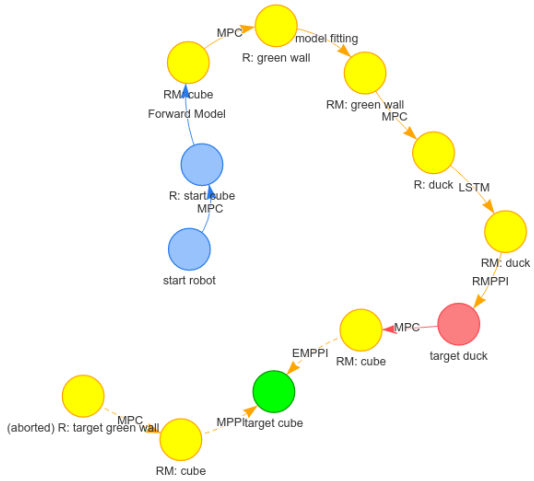
(a) Robot drives to green wall and performs system identification



(b) Green wall appears to be unmovable, pushing the green wall is impossible, abort subtask to push green wall to target position



(c) Generate a new starting state since much in the environment has now changed, replan a path from starting to target node



(d) Execute new plan to push the duck and then push the cube to the target position

Figure 4.13: HGraph detecting that the green wall is unmovable, generate new hypothesis pushing the movable duck and then push the cube to the target position, For the legend, see figure 4.10

The proposed approach and expected HGraph tackles a problem which [**siciliano_path_2009**, **wang_affordance-based_2020**, **scholz_navigation_2016**] are not able to solve because they lack a crucial component. [**siciliano_path_2009**] can navigate through movable obstacles and place objects on target positions. It is however provided with a set of movable and unmovable obstacles, the lack of learning dynamics prevents adaptation to new or changed environments. [**wang_affordance-based_2020**] is able to navigate among movable obstacles and where required, identify if an object is pushable and if so, push an object to free it's path. A push to free the path is quite different from pushing to a target position. Lacking is thus the ability to manipulate objects to target positions, which is required in the blockade by duck task. The same argument holds for [**scholz_navigation_2016**] who can navigate and move an object to free the path, but is unable to manipulate an object toward a target position.

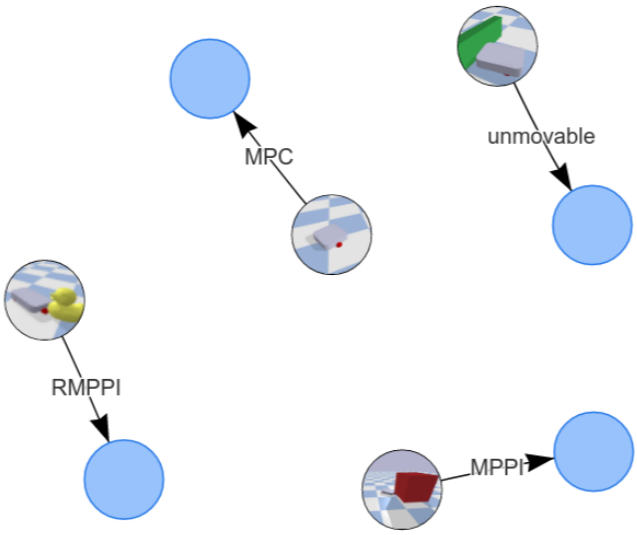


Figure 4.14: Knowledge graph after completion of the blockade task.

4.4.2. Surrounded by Cubes

Figure 4.15 displays a robot and 6 cubes, only the green cube is movable, the red, blue and yellow cubes are unmovable. **The task** given to the robot is to go toward a target location outside of the surrounding boxes. It is assumed that to drive toward target location the robot has to move a cube, moving the red cube gives the shortest path toward the target position, moving the blue cube gives a longer shortest path and moving the green cube gives an even longer shortest path. The goal of this BTest is to show that learned knowledge is stored and makes a difference when encountering objects for the second time.

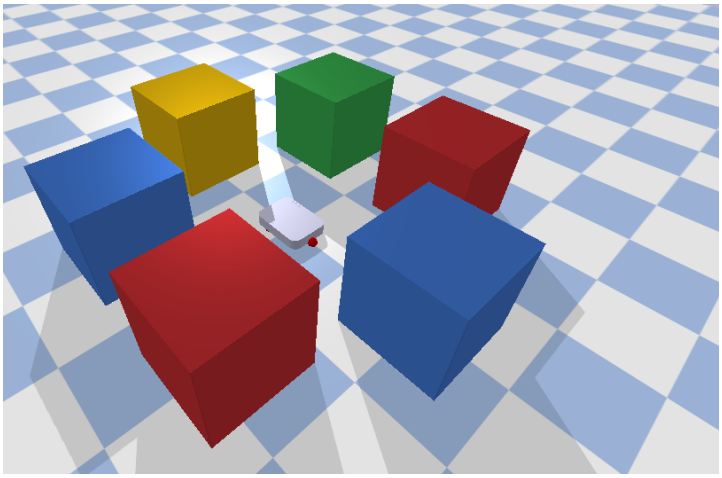
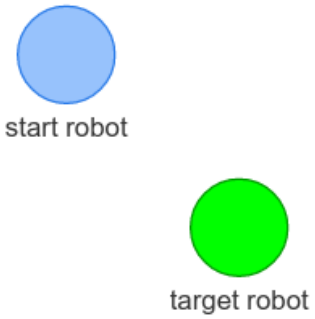
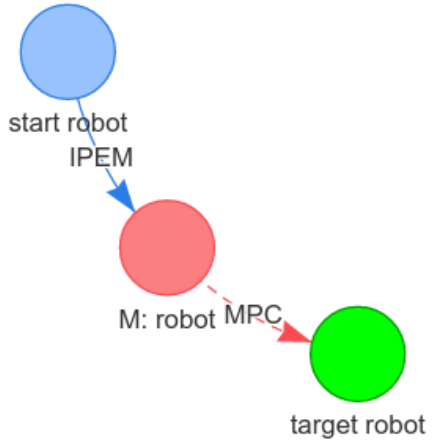


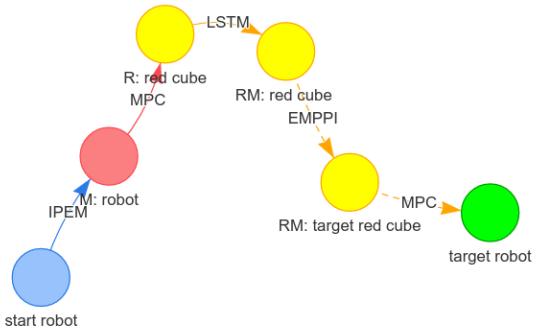
Figure 4.15: The robot is tasked with going to a target position outside the surrounding boxes, only one box is movable



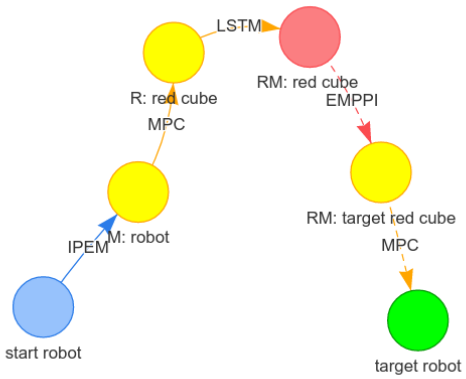
(a) Creating starting and target node



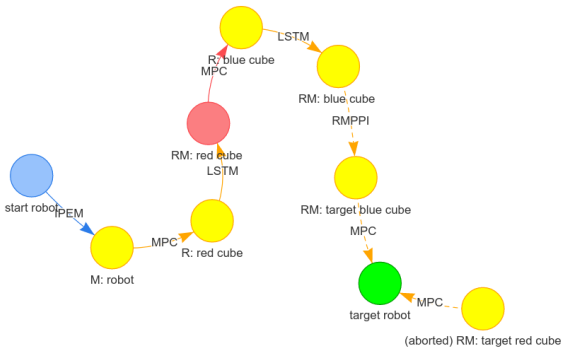
(b) Generate hypothesis driving toward target position and perform system identification on robot



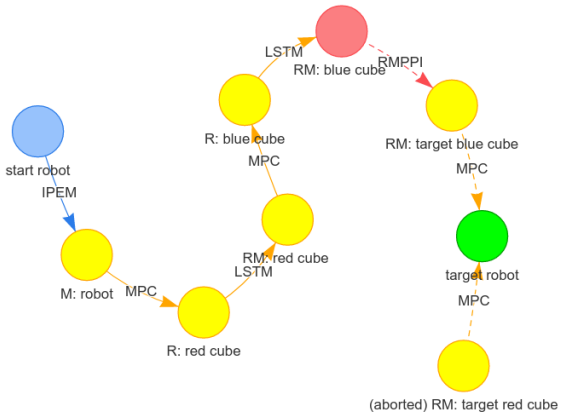
(c) Find red cube blocking the path which needs to be moved



(d) Drive toward red cube and perform system identification

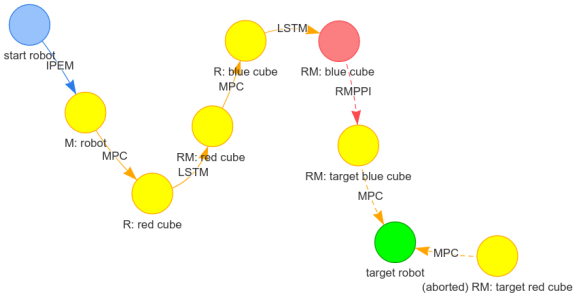


(e) The red cube in unmovable, abort subtask and replan to move the blue cube

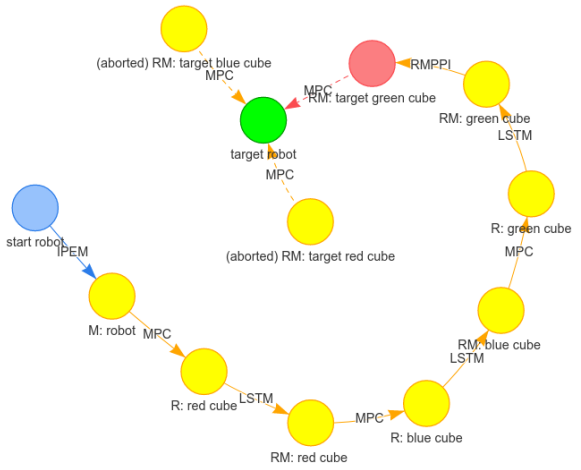


(f) Drive toward blue cube and perform system identification

Figure 4.16: Figure continues on the next page



(g) Blue cube is unmovable, abort subtask and replan to move the green cube



(h) Green cube is movable, after system identification push it. Now the path is free to drive toward the robot's target position

Figure 4.16: After multiple failed hypothesis, a succeeding hypothesis is found which completes the task. For the legend, see figure 4.10.

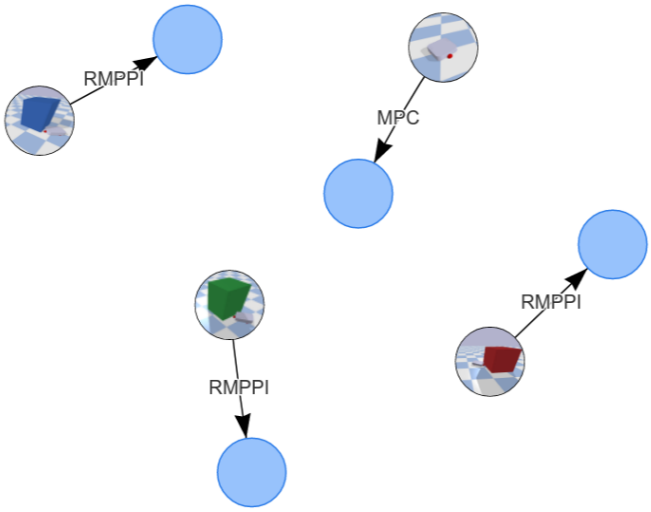


Figure 4.17: Knowledge graph after completion of the surrounded task.

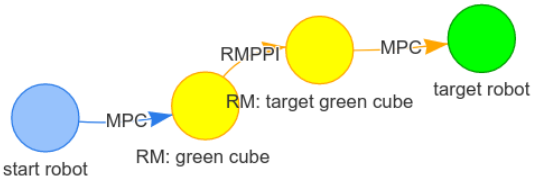


Figure 4.18: The hypothesis graph after redoing the same surrounded task for the second time. For the legend, see figure 4.10.

The second time the same surrounded task given to the robot some crucial knowledge is captured by the knowledge graph. Prior knowledge about the movability of the red, blue and green cubes is provided and system models for the cubes an the robot. as can be seen in figure 4.18 immediately goes for pushing the movable cube to then drive toward the target position.

4.4.3. Swapping Objects

Figure 4.20 displays a robot and movable obstacles. **The task** given to the robot is to swap the two objects, the duck must go to the position where the cube currently is and vise versa. The goal of this BTest is to give insight into the proposed methods ability to handle multiple objects, and to show that hierarchical methods yields a suboptimal solution.

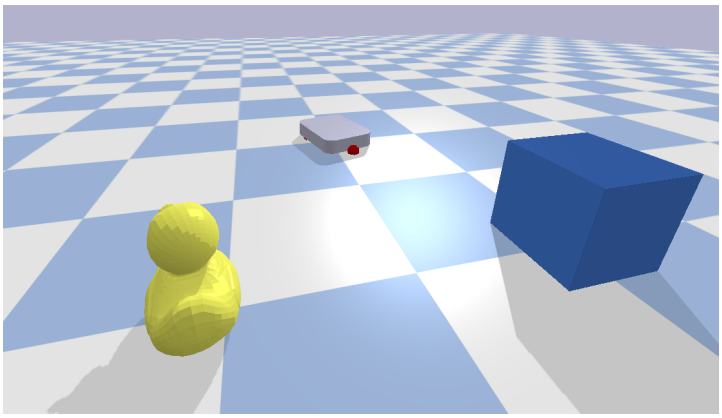
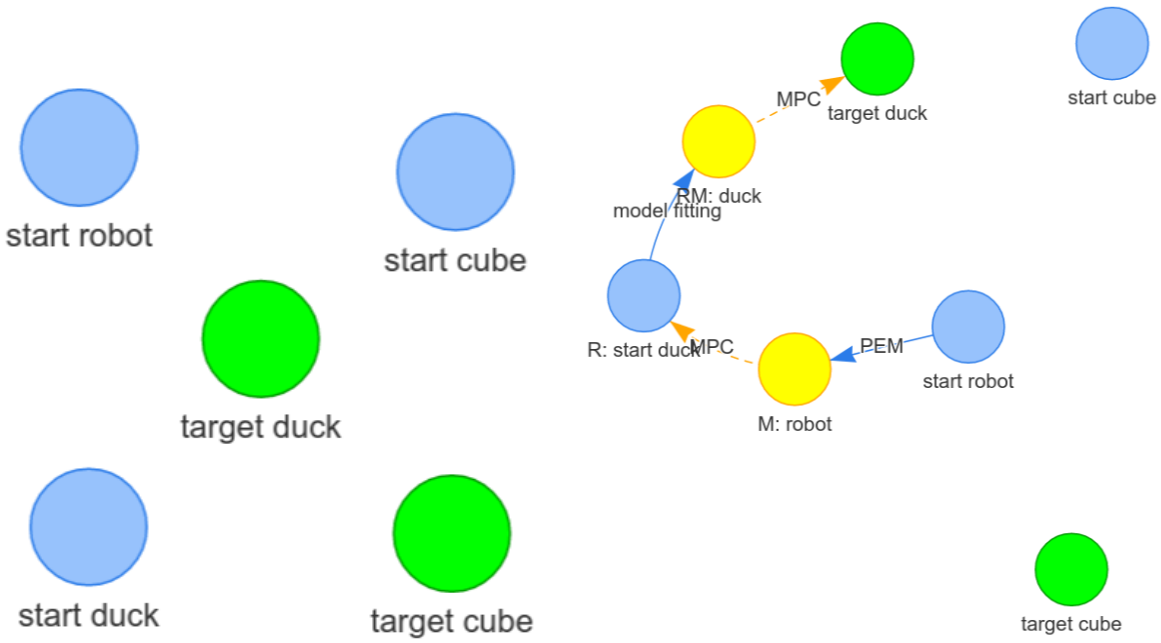


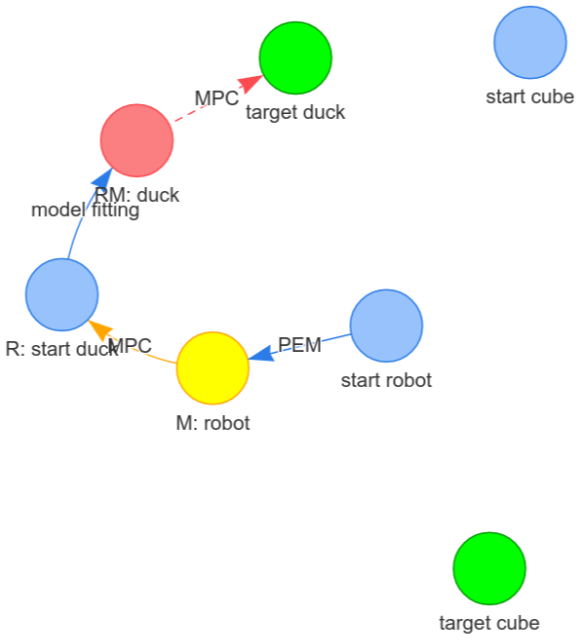
Figure 4.19: The robot is tasked with swapping the two obstacles



(a) Creating starting and target nodes

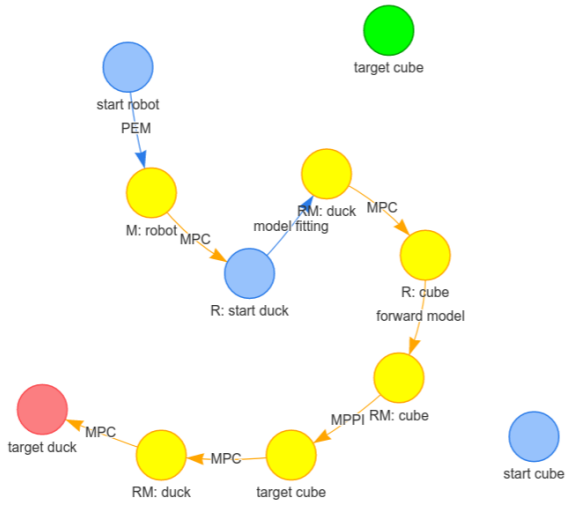
(b) Generate hypothesis to perform system identification on duck and place duck on target position

Figure 4.20: Figure continues on the next page

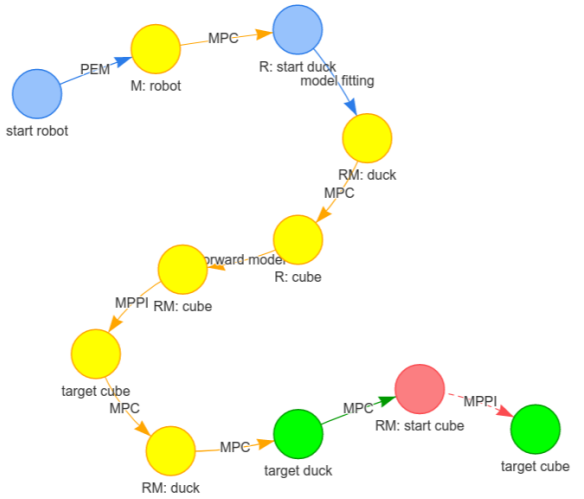


(c) Execute first 3 transitions

(d) The cube is a obstacle which needs to be moved during planning the duck to target position



(e) Execute transitions up until placing the duck on the target position



(f) Create hypothesis to place cube on target position and drive toward cube

Figure 4.20: Swapping two objects, for the legend, see figure 4.10.

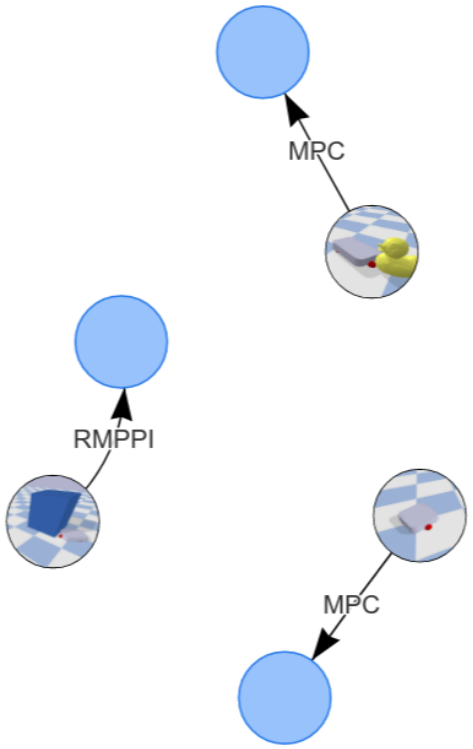


Figure 4.21: Knowledge graph after completion of the swap task.

The expected HGraph shows promising results, the ability to handle multiple objects of which the start and target positions are overlapping. Whilst object rearrangement algorithms exists [**krontiris_dealing_2015**], only [**sabbagh_novin_model_2021**] implements target positions of environment objects, it however only considers a single object and a free path. Allowing to conclude that learning object dynamics *and* NAMO *and* specifying objects target positions is a fairly new area of robotics.

A limitation due to the hierarchical search in the HGraph becomes visible. Imagine the objects to be swapped are very far apart. The proposed solution drives (possibly whilst being closer to the cube) to the starting position of the duck then to the cube, moves the cube a bit, back to the duck, then push the duck to target position and finally push cube to target position. Worst case, the robot is driving the distance between both starting positions 5 times, optimal is less than 3 times (assuming the robot is less than the initial cube-duck distance from either the duck or the cube). [**goldberg_asymptotically_2020**] already emphasised the suboptimality of hierarchical solutions. In addition, model mismatch and failure to find the existing path during motion and manipulation planning decrease the optimality of the plan found by the proposed method. Suboptimality is acceptable since that is not the goal of the literature study.

4.5. Discussion

The general structure of the hypothesis- and knowledge graph have been displayed in figures 4.5 and 4.8, the required components are listed, and benchmark tests display the expected outcome for specific situations and tasks. With a given task the robot will search for an action sequence to complete the task, upon failure of planning, generating new transitions or failure of execution, the robot will retry to find a different path toward completion of the given task. Executed transitions receive feedback which is processed in the knowledge graph. After trying all possibilities without success, the robot concludes it cannot find a solution to the task.

Table 4.2 compares the expected results with recent literature. As can be seen, only [**sabbagh_novin_model_2021**] is able to place objects on target positions *and* learn object dynamics, however [**sabbagh_novin_model_2021**] searches directly in a joint configuration space which is very different approach compared to the pro-

posed solution. The proposed solution theoretically can navigate among movable objects *and* put an object on target location *and* learn object dynamics, which is the main contribution, to combine both 3 with a hierarchical solution.

Citation	NAMO	Specify object target positions	manipulation	Learns object dynamics
[sabbagh_novin_model_2021]	✓	✓	driving, grasp-push, grasp-pull	✓
[wang_affordance-based_2020]	✓	✗	driving, pushing	✓
[scholz_navigation_2016]	✓	✗	driving, pushing	✓
[siciliano_path_2009]	✓	✗	driving,gripping	✗
[goldberg_asymptotically_2020]	✓	✓	driving,gripping	✗
proposed solution	✓	✓	driving, pushing	✓

Table 4.2: Comparing the proposed method to recent literature

5

Conclusions

This literature study sets up an task planning and execution framework without actually implementing it. Many techniques are used in the framework which can best be distinguished in subjects: control methods, system identification and lastly, motion and manipulation planning. These subjects have all been visited, current literature of the research field is categorised, summarised and discussed. Individually, these subjects all have techniques which have been proven to be effective. The proposed framework connects the subjects and claims they can complement each other.

After the robot environment has been defined, assumptions are listed an the task was described, the survey starts with answering the first research subquestion.

How are system models learned, and what are the limitations for different system learning techniques. To fully answer this question, models capable of describing the system models and control methods have been categorised. For **single-body control, predictive methods and hybrid system models are favoured.** Predictive methods such as MPC or MPPI can painlessly incorporate constraints, which is essential when controlling nonholonomic robots. Analytical models provide an sensible model at the initial time step, but require too prior knowledge which is unrealistic to assume, additionally analytic models are unable to adapt to system changes. Out of the reviewed system models, data-driven models converge closest to the true dynamics incorporating even small nonlinear effects, but converging takes much IO data which takes too much time. Hybrid system models take the best of both worlds, by assuming some structure hybrid systems are able to provide a workable model which converges toward the true dynamics with offline or online adaption. For **multi-body models predictive methods combined with data-driven approaches are favoured.** Multi-body models are contain primarily nonlinear dynamics which are harder to capture if any assumptions over the true dynamics are taken, this rules out hybrid models and the only viable option are data-driven models. Preferred system identification for single-body control are dominated by PEM, with variants specialised for online adaptation or minimal IO data required. Multi-body models are best learned with a set of test pushes which are converted to a data-driven model, truth be told, there exist no best system identification methods because it depends on single-/multi-body model, the choice of controller and mainly the situation (environment, robot geometry, object parameters, emphasis on minimal time required or minimal prediction errors). But generally for single-body control use predictive methods with hybrid models, for multi-body control use predictive methods with data-driven models.

In chapter 3 the second research subquestion is answered.

Why is task, motion and manipulation planning so much more difficult compared to task and motion planning. The configuration space becomes piece-wise analytic when adding manipulation planning to motion planning. Such a piecewise-analytic configuration space operates in different modes of dynamics. For example driving or pushing, the dynamical model for both verbs is different. This causes discontinuities in the configuration space at the boundaries where two different modes of dynamics meet. Planning in such a piece-wise analytic configuration space causes an combinational explosion of possibilities,

because pushing an object now influences how the configuration space looks like in the future for the mode of driving. The configuration spaces dimensionality grows linearly with the number of objects in it, which even for sampling based planners is not computationally feasible. As addition to motion planning, manipulation planning finds extra constraints to satisfy due to a generally more complex dynamical model. Most changes between motion and manipulation planning reside in the local planner, which validates if 2 sample configurations are connectable with a system model. Task planning has been shown to be NP-hard by comparing it to the standards piano's mover problem, which is known to be NP-hard. Luckily the goal is not to solve a NP-hard problem, but to learn dynamical models during execution of actions.

Chapter 4 answers the last subquestion partly.

Can planning in configuration space be extended to a piece-wise analytic configuration space with learned dynamics. Next to recent literature answering the subquestion with a yes [**sabbagh_novin_model_2021**], the answer is in the form of a proposed method. Anwering the research subquestion mediocrily. There are no proofs given, tests performed or even an implementation of the proposed method. However the components used by the proposed method individually have established guarantees. The proposed method contains the hypothesis graph, which appears to be an robust algorithm after visual inspection, as does the knowledge graph. The proposed methods claims to solve NAMO problems, with the ability to push movable obstacles to target positions while learning object dynamics. To reinforce the claims that the proposed methods makes, benchmark tests have been listed. Expected outcomes to the benchmark test have been hardcoded, which is for the final thesis to actually implement and compare against expected outcomes. The subquestion can be answered with yes, it is possible to extend a configuration space to a piece-wise analytic configuration space with learned dynamics. The answer can be given only because of the citation given [**sabbagh_novin_model_2021**], the proposed method on its own would not be sufficient.

Lastly, the main research question.

Can objects' system models be learned by a robot during task execution, and can these newly learned models improve task, motion and manipulation planning? Maybe, the proposed method gives a promising algorithm, and gives promises with benchmark test what is might accomplish. Every benchmark test clearly indicates how the proposed methods accomplishes results which existing literature would not be able to accomplish.

Glossary

List of Acronyms