THESIS

CONTROL OF NON-HOLONOMIC DRIFTLESS SYSTEM WITH UNKNOWN SENSORIMOTOR MODEL BY JACOBIAN ESTIMATION

December 2020

Shizuoka University
Graduate School of Science and Technology
Educational Division
Department of Information Science and Technology

Francisco Jesús ARJONILLA GARCÍA

Control of non-holonomic driftless system with unknown sensorimotor model by Jacobian estimation

Francisco Jesús **ARJONILLA GARCÍA**

Supervisor: Prof. Yuichi **KOBAYASHI**

Doctoral Dissertation
Department of Information Science and Technology
Graduate School of Science and Technology

Shizuoka University

December, 2020

Abstract

Robot navigation poses a difficult engineering problem. Autonomy in robot navigation is highly desirable because the range of problems solved by robotics without human intervention is increased. This thesis combines two different problems in robot navigation to improve the flexibility of autonomous robots. Firstly, non-holonomic mobile robots, *i.e.* mobile systems whose movement is restricted along some axis such as car-like vehicles, is a non-linear control problem that has no clear solution. However, the existing control policies to navigate non-holonomic robots assume that the kinematics of the robot as well as the sensor configuration, *i.e.* sensor type, orientation, *etc.*, are known beforehand. Secondly, some engineering problems require that the control system infer autonomously the relation between the control inputs and the sensor signals, referred as the sensorimotor mapping problem, and involves learning the kinematics of the robot, the sensor configuration, or the relation between control inputs and sensor change. In general, these learning approaches deduce the sensorimotor mapping in holonomic robots, that is to say, without the constraints characterised by non-holonomic systems. This thesis presents the formulation and results of an offline exploratory method that deals with the problems of non-holonomic control and unknown sensorimotor mapping in mobile robot systems. The proposed method initially focuses in completing the kinematics model of the system by retrieving the Jacobians of the sensor inputs with respect to the control outputs in the neighbourhood of the starting point of the robot. The Jacobian is a measure of rate of change of each sensor signal with respect to every control input. In order to retrieve the necessary Jacobian values to learn the kinematics model of the robot, each of the control inputs is controlled in a fixed pattern to obtain the Jacobian at predefined locations of the sensor space. The Jacobian element with the greatest orthogonality to the Jacobian at the starting location is selected and employed in deducing a virtual input that is equivalent to a sequence of real inputs. The virtual input has the same effect as navigating the robot along the direction forbidden by the non-holonomic constraint. The real and virtual inputs are then used to navigate and sample the sensor space in a systematic manner. Together with the sensor observations, the history of the control inputs to reach that sensor observation is also stored. The dataset of sensor observations and control inputs thus obtained is fed to a function approximation algorithm based on applying the least mean square algorithm on the parameters of the kernels of a Gaussian Radial Basis Function (RBF). The kernels are distributed uniformly in the sampled region of the sensor space such that all sensor observations are within the smallest convex region defined by the kernel locations. Once the virtual input is deduced and the sensor space is explored, the controllability of the mobile robot system is tested. For this purpose, the obtained mapping approximates sensor observations in the sensor space to chained-form control space, where it is controllable using well known control policies for non-holonomic systems. In this research, the time-axis transformation of chained form to linear control form was used, and state-space control methods to take the robot to the origin were applied. The method was validated by simulated and experimental results on a unicycle-like system, which consists of a three-state system with two control inputs and one holonomic constraint. The difference between a car-like system and a unicycle-like system is that the former has a constraint in the rotation radius that limits pure rotations of the system. The results in the simulated environment showed that the method is valid for the region of exploration of the sensor space and that the most important source of deviations of the trajectory of the robot during feedback control is caused by the accuracy of the approximation. Additionally, the effectiveness of the method was demonstrated in a real experiment by controlling a robot using the approximated mapping obtained previously. Simulation was performed in Matlab environment whilst the experiment was realised on a Pioneer 3-DX robot with processed images from an offboard camera. A comparison of the simulation with the Proximal Policy Optimization algorithm (PPO), which is a class of deep reinforcement learning algorithm, was also performed.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

There is a growing interest in autonomous navigation in recent years, especially with the introduction of mobile robots in domestic environments, autonomous driving and transportation of products in manufacturing facilities. For example, the market for automated vacuum-cleaning robots is expanding with new models released each year, some of them hosting advanced features such as Simultaneous Localisation And Mapping (SLAM), localisation by upwards-pointing cameras and radars. These robots cannot move sideways, so it is necessary to first rotate in the spot and align to the desired direction beforehand to achieve a similar movement. The car industry is rapidly transitioning from manually-driven, petrol-based vehicles to fully automatic, electric cars with the incorporation of countless cameras, proximity sensors, radar and GPS. Not only are car-like vehicles unable to move sideways as vacuum-cleaning robots, but also they cannot turn in the spot, requiring complex manoeuvres to realise similar movements. In these two examples of vacuum-cleaning robots and autonomous navigation, the kinematics of the mobile systems are well known beforehand during the design process, and so are the kinds of sensors, their placement and the specification of the output signal. Hence, the kinematics and the sensor configuration are hard coded into the controller of these kind of systems. Industrial mobile robots face similar challenges as well. But unlike domestic robots or vehicles, they have additional difficulties to overcome. Industrial robots must be highly flexible to accommodate to rapid changes in the manufacturing process or in the products to be manufactured. The result is that in many manufacturing environments, the sensors are not fixed to the mobile robot and must be reconfigured every time that there is a change in manufacturing. Even when repairing these robots, there could be differences in the replaced parts, such as wheel diameters or distance between wheels, that require manual readjustments to the control parameters. These changes are time consuming and expensive, requiring highly qualified personnel. Therefore, it would be convenient to have learning systems that quickly accommodates the controller to new sensor configurations.

This thesis proposes a method to improve the controllability of mobile robots when the sensor configurations or the kinematics are not known during the design process of the controller. It facilitates the emplacement of mobile robots by improving the flexibility in the installation of sensors. For example, controlling a mobile robot guided by LiDAR is strongly dependent on the specifics of the LiDAR model used, such as the number of scans and angular span of the scans. Also, original replacement parts for a damaged robot, such as wheels or reduction gears, might be impractical for economic reasons, and instead parts with different parameters to the original parts are installed on the robot. Replacing the LiDAR model used or the parameters of the robot kinematics might require a firmware update, or it might even be impossible due to development limitations by the company providing the controller. The method proposed here enables more flexibility in sensors and robot kinematics by automating the process of configuration of new sensors to mobile robots.

1.2 Related research

Research on **non-holonomic systems** has a long history [21][4]. A detailed explanation of non-holonomic systems and the relation with other systems is found in [7]. The major problem of non-holonomic systems is that control laws are difficult to design. Indeed, there is no general stabilising law for non-holonomic systems [5], partially because there are unavoidable obstructions for stabilisation of feedback control [42]. Yet, under specific assumptions about the kinematics of the robot and the environment, stabilising control laws are possible [23]. [3] overcame the obstructions in discontinuous exponentially stabilising control laws by a singular coordinates transformation, such as kinematic polar coordinate transformations [1]. In any case, by using Lagrangian formalism and differential geometry, 3-wheeled robots are stabilisable with static-state feedback controllers [8].

Control of non-holonomic systems is often achieved in **chained-form** [18], which is a linear coordinate system specially designed for easy controllability [9]. Navigation problems of systems in chained-form have been widely studied [37], and are practically realisable with bounded control signals [31].

Control systems that automatically learn the parameters needed for successful control have been available for a long time, although they are generally limited to holonomic systems [36]. These **learning controllers** model the relation between control inputs and sensor outputs in robots automatically [35], even continuously adapting the model to changes in the sensorimotor relation [38], or inducing regions of sensor observations without sensory data [20]. These systems of **unknown sensorimotor mapping** have been the target of trial-and-error methods as well [39].

Jacobian control methods for non-holonomic systems have been widely studied in the past [17][41][49][12], yet these methods assume that the Jacobian is known or can be deduced from the kinematics of the system. [19] also proposed estimating the Jacobian matrix with unknown system parameters by approximating the relation between actuators and sensors using a measurement given by Mutual Information (MI). Estimation of the Jacobian is sometimes referred to as the **bootstrapping** problem because obtaining the Jacobian is the first step of a learning system towards learning a model of itself and the environment that can be used for control purposes [6]. Similarly, **uninterpreted sensors and effectors** [40] and **calibration-free** robotics [15][14][28] refer to similar problems.

In the field of **Reinforcement Learning** (RL), there have been many researches related to mobile robotics [47][50]. In general, RL algorithms are well suited to the problem of unknown sensorimotor mapping because RL does not rely in models of the environment, but rather explores the environment, initially by applying control inputs by trial and error and disregarding the structure of the problem [24]. With the advent of deep neural networks and Actor-Critic RL methods [11][46][30][33], it has become easier to apply reinforcement learning on navigation problems of non-holonomic systems [55], although research on deep RL is generally focused to applications other than control of non-holonomic systems with unknown sensorimotor mapping [29].

Trajectory tracking control of non-holonomic systems has been often researched in the past [52][10], specially in relation to unknown camera parameters [51] and adaptive **visual servoing** [2][25][54]. Tracking control of uncertain non-holonomic systems is still of some interest and continues yielding results [53]. Reinforcement Learning has also been introduced to visual servoing for non-holonomic systems [32], as well as adaptive neural networks.[56]. Visual servoing has been studied from the viewpoint of Jacobian matrix estimators for holonomic systems as well [16].

In **summary**, there are few if not no researches directly tackling the problem of control of non-holonomic systems with unknown kinematics and unknown sensor configuration. Either of these problems has been widely documented in the state of the art, but the combination of both problems is a topic that remains practically untouched, specially in the scope of real robots. The closest methods that approach these two problems simultaneously are those based in reinforcement learning algorithms, but they are too inefficient for practical applications in real environments.

1.3 Objective

The objective of this research is to propose methods that increase the range of applicability of mobile robotics by combining the problem of non-holonomic control with learning capabilities of the sensorimotor mapping. Sensorimotor mapping refers to the changes in sensor values when the system is applied some control input. The proposed method must be physically feasible, *i.e.* it must be functional on a real robot, and not only in a simulation.

1.4 Outline

This thesis is organised as follows. Chapter 1 presents the problem of non-holonomic control with unknown sensorimotor mapping along with a review of the state of the art in this field. The objective of the research is also described here. Chapter 2 describes the problem setting formally and briefly describes the approach taken to tackle it. Chapter 3 introduces general notions about generalised coordinates and non-holonomic systems, which are recurring topics in this thesis. Specifically, the vector spaces considered throughout the thesis are presented: system space, sensor space, chained-form space and time-axis control form space. Chapter 4 describes well known general techniques used throughout the rest of the thesis that are necessary to construct the proposed method: Jacobians, function approximation by Gaussian RBF and state-space control of time-axis control form. Chapter 5 develops the formal tools to solve the problem with the proposed method and is the core of the research: the main contributions of this research are presented in Chapter 5 in detail. Chapters 6 is dedicated to showing the validity of the method in a simulated environment. It includes a comparison with other learning techniques in the field of robot control. Chapter 7 shows the implementation and results of the proposed method in a real experiment, demonstrating that the method is feasible in real world problems. Finally, Chapter 8 summarises the research and analyses the implications.

Chapter 2

Problem setting and method overview

In this chapter the problem is formalised with explicit indication of the assumptions and the unknowns. Then the general approach taken to solve the problem in this work is described.

2.1 Problem setting

Let $\boldsymbol{u} \in \mathbb{R}^m$ be the control (input) vector and $\boldsymbol{s} \in \mathbb{R}^n, n > m$ the sensor (output) vector of a dynamic driftless affine system with state and output equations

$$\dot{\boldsymbol{q}} = F(\boldsymbol{q})\boldsymbol{u} \tag{2.1}$$
$$\boldsymbol{s} = H(\boldsymbol{q}), \tag{2.2}$$

where $\boldsymbol{q} \in \mathbb{R}^n$ is the vector of generalised coordinates, $\dot{\boldsymbol{q}}$ is the vector of generalised velocities, and $H : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ is an isomorphic mapping of class C^1. The transformation H is arbitrary and has no units.

A driftless dynamic system is a system that has no independent term in the state equation: If $\boldsymbol{u} = 0$, then $\dot{\boldsymbol{q}} = 0$. Equation (2.1) is a driftless dynamic system. Intuitively, a driftless system does not move when the control input is null. For example, a car-like vehicle whose control inputs are acceleration and brake is not driftless because under no acceleration or brake, the vehicle keeps moving. But if instead of acceleration or braking, the control inputs are the speed of the wheels, then the vehicle is driftless because when the wheels are stopped, so is the vehicle (unless the driver is aggressively driving the car).

The problem tackled in this research is to find a control law $\boldsymbol{u} = \varphi(\boldsymbol{s})$ that realises a desired sensor value of $\boldsymbol{s}^{(d)}$ under the condition of unknown F, H (unknown kinematics and sensor configuration), with arbitrary $\boldsymbol{q}$, and with non-holonomic constraints compatible with Pfaffian form, *i.e.* $A(\boldsymbol{q})\dot{\boldsymbol{q}} = 0$ [7] (More details in Section 3.2.2). In other words, $\boldsymbol{q}$ is observable but only through an uncalibrated sensor measurement. The problem is similar under redundant observations $\boldsymbol{s}' \in \mathbb{R}^r, r > n$, in which case the equation to consider is $\boldsymbol{s}'' = H''(\boldsymbol{q})$, with $\boldsymbol{s}'' \in \mathbb{R}^n$ and H'' being an isomorphic mapping.

This is a list of **assumptions**:

1. The system has exactly one non-holonomic constraint $A(\boldsymbol{q})\dot{\boldsymbol{q}} = 0$.
2. The system is driftless $\dot{\boldsymbol{q}} = F(\boldsymbol{q})\boldsymbol{u} + C \quad | \quad C = 0$.
3. The system has exactly two control inputs $\boldsymbol{u} = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^\mathsf{T}$.
4. The inputs $\boldsymbol{u}_{(1)} = \begin{bmatrix} 1 & 0 \end{bmatrix}^\mathsf{T}$ and $\boldsymbol{u}_{(2)} = \begin{bmatrix} 0 & 1 \end{bmatrix}^\mathsf{T}$ are kinematically feasible.

5. Reverting an input returns the system to the previous state.
6. The sensor outputs are an isomorphic transformation of the internal state variables of the system.
7. The sensor outputs are differentiable with respect to the control inputs, *i.e.* $J = {}^{\partial \dot{\boldsymbol{s}}}/{}_{\partial \boldsymbol{u}}$ exists.

Point 4 is deduced from (2.1):

$$\dot{\boldsymbol{q}} = F(\boldsymbol{q})\boldsymbol{u} \quad \Rightarrow \quad F(\boldsymbol{q})(-\boldsymbol{u}) = -\dot{\boldsymbol{q}}, \tag{2.3}$$

but it is made explicit in this list of assumptions due to its importance in the developments of the proposed method.

These are the **unknowns** of the problem:

1. The kinematics of the dynamic system $F(\boldsymbol{q}) =?$.
2. The output equation, *i.e.* the sensor configuration of the system $H(\boldsymbol{q}) =?$.
3. The control law $\boldsymbol{u} = \phi(\boldsymbol{s})$ that takes the system from any sensor value $\boldsymbol{s}^{(0)}$ to the desired sensor value $\boldsymbol{s}^{(d)}$.

In the following chapters a more detailed description of these concepts is given.

2.2 Overview of the proposed method

In this thesis, an offline learning algorithm that tackles the above mentioned problem setting is proposed. This method is based on finding an approximate function from sensor space (Section 3.1.4) to chained form (Section 3.1.5) by retrieving sensor observations at several points along the sensor space to construct an approximate model of the kinematics of the system and the sensor configuration. The data available to a controller of a dynamic system in the problem setting described in Section 2.1 is limited to the control input, which is produced by the controller itself, and the sensor observations. The relation between the control input and the sensor observations is called the *sensorimotor mapping*. By applying a control input to the dynamic system, a reaction in the form of variations of sensor observations can be measured. These reactions vary with the current value of the sensor observations. The Jacobian is a measure of these variations.

The method starts by exploring the vicinity of the sensor space at the starting position to obtain the changes in the sensor values with respect to the control inputs (Section 4.1), which is the Jacobian of the system. A Jacobian matrix J, or simply Jacobian, is a matrix of first order derivatives of a multivariate vector-valued function $\dot{\boldsymbol{s}}(\boldsymbol{u})$, with $\boldsymbol{u} \in \mathbf{R}^n$ and $\boldsymbol{s} \in \mathbf{R}^m$:

$$J = \begin{bmatrix} \boldsymbol{j}_1 & \cdots & \boldsymbol{j}_n \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \dot{\boldsymbol{s}}}{\partial u_1} & \cdots & \dfrac{\partial \dot{\boldsymbol{s}}}{\partial u_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathsf{T}} \dot{s}_1 \\ \vdots \\ \nabla^{\mathsf{T}} \dot{s}_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \dot{s}_1}{\partial u_1} & \cdots & \dfrac{\partial \dot{s}_1}{\partial u_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial \dot{s}_m}{\partial u_1} & \cdots & \dfrac{\partial \dot{s}_m}{\partial u_n} \end{bmatrix}, \tag{2.4}$$

where $\dot{\boldsymbol{s}}$ is the vector of rate of change of sensor values and $\boldsymbol{u}$ is the vector of control inputs. Based on this data, a virtual input (Section 5.1) is deduced that has same effect as travelling along the forbidden direction imposed by the non-holonomic constraints (Section 3.2). This virtual input is constructed by composing sequences of legal actions.

The second stage of the method involves exploring the sensor space in prefixed patterns that make use of the virtual input obtained previously (Section 5.2). During exploration, sensor observations are retrieved in regular intervals as well as the history of the control inputs required to arrive at each sensor observation. This data is stored in a dataset that is supplied to a Gaussian radial basis function approximator (Section 4.2), which is a supervised learning method. The prefixed patterns are designed such that the resulting approximated function approximates the system to an equivalent linear system in chained form (Section 3.1.5). By transforming each sensor observation to chained form, it is possible to control the system with linear controllers. Here the method was demonstrated by evaluating the accuracy of the chained-form transformation with a control policy based on time-axis control form (Section 3.1.6.1). The method was also compared with a state-of-the-art controller based on reinforcement learning (Section 6.3).

Chapter 3

Coordinates in non-holonomic systems

Before delving into the technical details of the proposed approach, it is necessary to understand some fundamental concepts in the field of dynamic systems. In this chapter, two concepts are introduced: Generalised coordinates, which is a generalisation of the state space of a system, and non-holonomic systems, which is a class of dynamic systems with restrictions on either the state values or their evolution in time.

3.1 Generalised coordinates

A generalised coordinate may be regarded as a generalisation of the state variables that describes the configuration of a dynamic system. Basically, generalised coordinates express the same information as the state vector of the system, but they can also be expressed in any other coordinate base with a coordinate transformation function. For the rest of this document, the so-called state variables of a system correspond to what will be referred as the canonical generalised coordinates.

3.1.1 Definition

The set of generalised coordinates $\{q_i\}$ is a set of parameters that uniquely describes the configuration state of a system. Cartesian coordinates is a concrete example of generalised coordinates. Polar, cylindrical and spherical coordinates are other concrete examples of generalised coordinates. The set of generalised coordinates is not unique. For example, Figure 3.1 shows two sets of generalised coordinates, Cartesian and spherical, of a hypothetical system that con-

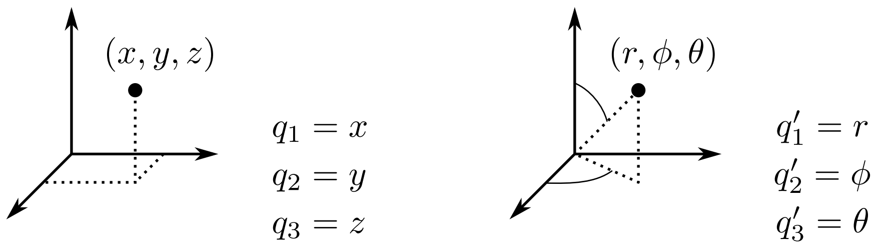


Figure 3.1: Two sets of generalised coordinates for describing a system of three degrees of freedom in the same state.

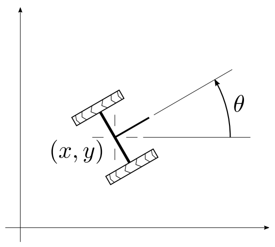


Figure 3.2: Canonical set of generalized coordinates (x, y, θ) for the unicycle.

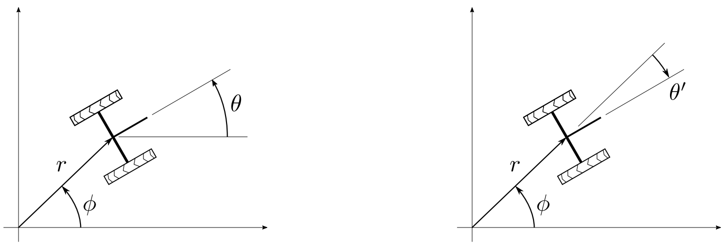


Figure 3.3: Alternative sets of generalized coordinates (r, ϕ, θ) and (r, ϕ, θ') for the unicycle.

sists of a point in three-dimensional space. The set of generalised coordinates are normally grouped in a vector $\boldsymbol{q} := \begin{bmatrix} q_1 & q_2 & \cdots \end{bmatrix}^\mathsf{T}$. The set of generalized coordinates $\boldsymbol{q}$ is a vector space.

In dynamic systems, the configuration state of the system is described by generalized coordinates that depend on time $\boldsymbol{q} = \boldsymbol{q}(t) = \begin{bmatrix} q_1(t) & q_2(t) & \cdots \end{bmatrix}^\mathsf{T}$. For any given dynamic system, the choice for generalized coordinates is not a trivial problem in general. Usually, the generalized parameters are selected so that they are convenient to operate with and make the solution of the equations of motion of the system easier. This choice is referred to as the canonical coordinates of the system. If the selected generalized coordinates are independent, then the number of generalized coordinates is equal to the number of degrees of freedom of the system.

In the unicycle system, the preferred choice of generalized coordinates are two cartesian coordinates x and y for the position of the system and the absolute angle θ of the unicycle heading with respect to the x-direction, as shown in Figure 3.2. As explained above, other generalized coordinates are possible such as those depicted in Figure 3.3. By convention, the generalized coordinates at Figure 3.2 are normally used in the literature, thus the canonical coordinates for the unicycle are (x, y, θ).

3.1.2 Problem description by generalized coordinates

The solution proposed in this thesis to solve the problem posed at Section 2.1 can be described by how generalized coordinates are used. To better understand the problem, here is a reformulation of the problem description in terms of generalized coordinates.

Given a set of generalized coordinates with unkown interpretation, that is, only the values are known but how these values are related to the system is unknown, design a set of generalized coordinates that facilitates controllability of the system and find a coordinate transformation between the given unknown generalized coordinates and the generalized coordinates for controlling the system.

During the remaining of the thesis, the following four sets of generalized coordinates, or state spaces, will be used:

1. System space: The canonical generalized coordinates. Used to implement simulators only.

2. Sensor space: The generalized coordinates defined by the values given by the robot sensors. The learning controller only has access to this space in order to know the system state.

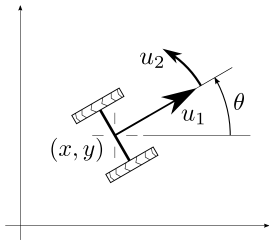


Figure 3.4: The unicycle, as studied here, has three generalized coordinates (x, y, θ), corresponding to location and heading, and two control inputs u_1 and u_2 that correspond to linear speed and rotational speed, respectively.

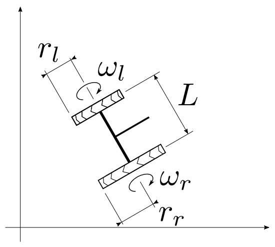


Figure 3.5: The unicycle with wheel rotational speed control inputs.

3. Chained-form space: This is a set of generalized coordinates where the state equation of the system is linear. The transformation from sensor space to chained form, also denoted as sensorimotor mapping and approximated mapping in this thesis, is the objective of this research. This space is used because the amount of research about controlling systems in this form is huge.

4. Time-axis control space: A variation of chained-form space. The generalized coordinates are the same as chained-form space, but the state equations of the system are modified as explained in Section 3.1.6 to facilitate the execution of a linear control policy for the purpose of evaluating the approximated mapping.

Therefore, the four state spaces reflect the same intrinsic state of the system, but each set of generalized coordinates serves a different purpose. The following sections describe some of these state spaces in detail and show how they are applied to the problem of the unicycle.

3.1.3 System space

The generalized coordinates in this space are the canonical coordinates that are used to describe the dynamic state equations of the system. In the unicycle, the generalized coordinates are x, y and θ, as depicted in Figure 3.4, and the state equation is

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u_2, \tag{3.1}$$

with u_1 being linear speed input and u_2 rotational speed input. Alternatively, a unicycle may be realized by a two-wheel vehicle as illustrated in Figure 3.5. Then the state equation can be expressed with reference to the rotational speeds ω_l and ω_r of the left and right wheels respeectively, with no change to the generalized coordinates, by transforming the linear and rotational speeds

$$\omega_l = \frac{1}{r_l}\left(u_1 - \frac{L}{2}u_2\right) \tag{3.2}$$

$$\omega_r = \frac{1}{r_r}\left(u_1 + \frac{L}{2}u_2\right), \tag{3.3}$$

where r_l and r_r are the radii of the left and right wheels, respectively, and L is the distance between wheels (Figure 3.5). Solving for u_1 and u_2,

$$u_1 = \frac{(r_l\omega_l + r_r\omega_r)}{L} \tag{3.4}$$

$$u_2 = \frac{(r_l\omega_l + r_r\omega_r)}{L}. \tag{3.5}$$

Therefore, the state equation of the unicycle system with wheel rotational speed inputs is

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ -1 \end{bmatrix} r_l\omega_l + \begin{bmatrix} \cos\theta \\ \sin\theta \\ 1 \end{bmatrix} r_r\omega_r. \tag{3.6}$$

This form of the state equation considers different wheel radii, but form (3.1) does not. (3.1) is generalizable to a two-wheel vehicle with different wheel radii: Taking $R := r_l/r_r$ as the ratio between the radii of the left and right wheel,

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R+1}{2}\cos\theta \\ \frac{R+1}{2}\sin\theta \\ \frac{R-1}{2} \end{bmatrix} u_1 + \begin{bmatrix} \frac{R-1}{2}\cos\theta \\ \frac{R-1}{2}\sin\theta \\ \frac{R+1}{2} \end{bmatrix} u_2. \tag{3.7}$$

By setting $R = 1$, (3.1) is obtained, as expected.

These equations are not available in the target controller of this thesis and are only used for analysis of the problem and for realizing simulators.

3.1.4 Sensor space

Sensor space is defined as an arbitrary transformation of generalized coordinates that reflects the sensor configuration of a system. The difference between sensor space and system space is just convenience of implementing simulators, because the state equation of a system with arbitrary choice of generalized coordinates can be complex. Even so, the choice of generalized coordinates for the sensor space and the choice for the system space might be the same in many systems, and it depends on the kinematics of the system, the configuration of the sensors and the preferences of the designer. In the problem setting that concerns this thesis, only the sensor space is directly accessible, so the choice of generalized coordinates for the system space is irrelevant.

3.1.5 Chained-form space

Let $\dot{\boldsymbol{q}} = \begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix}^\mathsf{T}$ be an arbitrary choice of generalized coordinates of a linear dynamic system that is controlled by input $\boldsymbol{u} = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^\mathsf{T}$. Then, there is a transformation $\boldsymbol{z} = F\boldsymbol{q}$, where F is a linear transformation matrix, such that the state equation of the system under generalized coordinates $\boldsymbol{z}$ is in chained form (proof in [9]):

$$\dot{\boldsymbol{z}} = \boldsymbol{g}_1(\boldsymbol{z})u_1 + \boldsymbol{g}_2u_2 \tag{3.8}$$

with

$$\boldsymbol{g}_1(\boldsymbol{z}) = \begin{bmatrix} 1 \\ 0 \\ z_2 \\ \vdots \\ z_{n-2} \end{bmatrix} \quad \text{and} \quad \boldsymbol{g}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{3.9}$$

In the case of the unicycle, a non-linear transformation $\boldsymbol{z} = F(\boldsymbol{q})$ is necessary. There are many such transformations, but all of them yield a singularity along at least one direction. Typically, the following transformation is applied to the canonical coordinates

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} x \\ \tan\theta \\ y \end{bmatrix} \tag{3.10}$$

and to the control inputs

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta)u_1 \\ \dfrac{1}{\cos^2(\theta)}u_2 \end{bmatrix}. \tag{3.11}$$

The singular direction is $\theta = \pm^{\pi}/_{2}$ because the inverse transform of v_1 diverges:

$$u_1 = \frac{1}{\cos(\pm^{\pi}/_{2})} v_1 = \infty. \tag{3.12}$$

Chained form for the unicycle is then as expected,

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ z_2 \end{bmatrix} v_1 + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} v_2. \tag{3.13}$$

In this thesis, the transformation function $F(\boldsymbol{q})$ and the state equation (3.1) are not known to the learning controller, so from the controller point of view, it is indifferent which control input $\boldsymbol{u}$ or $\boldsymbol{v}$ is applied. Actually, for rotations $\theta \simeq 0$, the transformation is the identity $\boldsymbol{v} \simeq \boldsymbol{u}$. This approximation has a degrading effect on the performance of the learning controller. In Chapters 6 and 7 it is shown that the performance is somewhat degraded, but the learning controller can deal effectively with it.

3.1.6 Time-axis control form

Time-axis control form was first proposed in [44][45] and reformulated in [26][27]. Despite time-axis control form not receiving much attention in the literature, it has been applied for adaptive control strategies[34].

3.1.6.1 Definition

Time-axis control form is a modification to chained form that simplifies the control problem of a linear system with two inputs to a linear system with just one input. Time-axis control form divides the control problem into two control problems of separate linear systems. The first linear system is given by the state equation

$$\frac{dz_1}{dt} = \mu_1, \tag{3.14}$$

where the control input $\mu_1 = u_1$ controls the first state variable of chained form. This state variable replaces the time variable in the second linear system, whose state equation is

$$\frac{d}{dz_1}\begin{bmatrix} z_n \\ \vdots \\ z_3 \\ z_2 \end{bmatrix} = \begin{bmatrix} z_{n-1} \\ \vdots \\ z_2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \mu_2, \tag{3.15}$$

where $\mu_2 = {}^{u_2}/_{u_1}$.

3.1.6.2 Derivation

By applying the chain rule on the time derivative of the generalized coordinates $\boldsymbol{z}$ with respect to the first coordinate z_1 in the state equation of a dynamic system,

$$\frac{d\boldsymbol{z}}{dt} = \frac{\partial \boldsymbol{z}}{\partial z_1}\frac{\partial z_1}{\partial t}. \tag{3.16}$$

Analyzing (3.16) component by component and considering the chained form state equation (3.8), the following equations hold.

For z_1,

$$\frac{\partial z_1}{\partial z_1}\frac{\partial z_1}{\partial t} = \frac{\partial z_1}{\partial t} = u_1. \tag{3.17}$$

Hence, the first control input $\mu_1 := u_1$ in time-axis control form matches the first control input in chained form. For z_2,

$$\frac{\partial z_2}{\partial z_1}\frac{\partial z_1}{\partial t} = \frac{\partial z_2}{\partial z_1}u_1 = u_2, \tag{3.18}$$

resulting in the second control input $\mu_2 := {}^{u_2}/_{u_1}$ in time-axis control form. For the remaining components of $\boldsymbol{z}$,

$$\frac{\partial z_i}{\partial z_1}\frac{\partial z_1}{\partial t} = z_{i-1}u_1, \tag{3.19}$$

so ${}^{\partial z_i}/_{\partial z} = z_{i-1}$ for all $i > 2$. Summing up, the time-axis control equations for two inputs and n state variables is

$$\frac{d}{dz_1}\begin{bmatrix} z_n \\ \vdots \\ z_3 \\ z_2 \end{bmatrix} = \begin{bmatrix} z_{n-1} \\ \vdots \\ z_2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \mu_2 \tag{3.20}$$

$$\frac{dz_1}{dt} = \mu_1 \tag{3.21}$$

which are precisely (3.15) and (3.14), respectively. Hence, in time-axis control form, $\begin{bmatrix}\tau & \boldsymbol{\zeta}\end{bmatrix}^\intercal = \begin{bmatrix} z_1 & z_2 & \cdots & z_n \end{bmatrix}^\intercal$ and $\boldsymbol{\mu} = \begin{bmatrix} u_1 & {}^{u_2}/_{u_1}\end{bmatrix}^\intercal$.

3.1.6.3 Time-axis control form on the unicycle

Positive time values Let us apply the above equations to the state equation of the unicycle in canonical form. Recall that the unicycle is a non-linear system whose state equation is given by

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u_2, \tag{3.22}$$

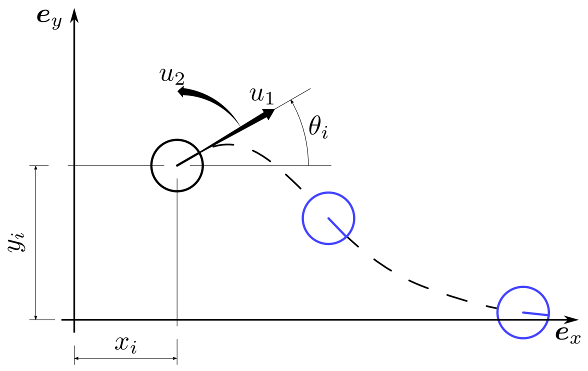


Figure 3.6: A unicycle-like mobile controlled by a linear controller in time-axis space, with $\boldsymbol{e}_x$ as time axis.

where u_1 is the linear speed and $u_2 = \dot{\theta}$ is the rotational speed, as depicted in Figure 3.6. The variables ξ_i are assigned as follows:

$$\boldsymbol{\xi} = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}. \tag{3.23}$$

Now, the state variables are transformed such that

$$\boldsymbol{s} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = \boldsymbol{f}(\boldsymbol{\xi}) = \begin{bmatrix} \xi_1 \\ \tan(\xi_3) \\ \xi_2 \end{bmatrix}. \tag{3.24}$$

The control inputs are transformed too:

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = h^{-1}(\boldsymbol{\xi})\boldsymbol{u} = \begin{bmatrix} \cos(\xi_3) & 0 \\ 0 & \dfrac{1}{\cos^2(\xi_3)} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \tag{3.25}$$

Substituting (3.24) and (3.25) into (3.22),

$$\dot{\boldsymbol{\xi}} = \begin{bmatrix} \dot{s_1} \\ \dot{s_3} \\ \dot{s_2}\cos^2(\xi_3) \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \dfrac{1}{\cos(\theta)} & 0 \\ 0 & \cos^2(\theta) \end{bmatrix} \boldsymbol{v}. \tag{3.26}$$

Hence,

$$\frac{d}{dt} \begin{bmatrix} s_1 \\ s_3 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \tan(\theta) & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{v}, \tag{3.27}$$

so we arrive at the chained form for the unicycle under transformation (3.24):

$$\dot{\boldsymbol{s}} = \begin{bmatrix} 1 \\ 0 \\ s_2 \end{bmatrix} v_1 + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} v_2 \tag{3.28}$$

Applying the previously derived equation (3.20), the time-axis control form equations for the unicycle are:

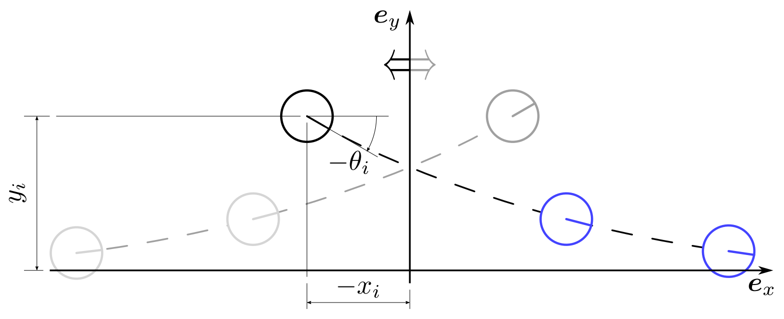


Figure 3.7: Reverse displacements are controlled by a simmetric equivalent system where $x' = -x$ and $\theta' = -\theta$.

$$\begin{aligned}\frac{d}{d\tau}\begin{bmatrix}\zeta_3\\ \zeta_2\end{bmatrix} &= \begin{bmatrix}\zeta_2\\ 0\end{bmatrix}+\begin{bmatrix}0\\ 1\end{bmatrix}\mu_2\\ \frac{d\tau}{dt} &= \mu_1\end{aligned}\tag{3.29}$$

where μ_1 controls the *time* variable of (3.29) and μ_2 is the input to apply a control law. The control law is deduced in Section (4.3). Note that positive values of μ_1 will make a control policy converge whilst negative values will diverge.

In the problem setting tackled in this thesis, the transformations from system space to chained form are not known. Specifically, the transformation $\boldsymbol{v} = f(\boldsymbol{u})$ is unknown, so in order to simplify the problem, $\boldsymbol{v} = \boldsymbol{u}$, as will be explained in Section 5.2.

Negative time values Feedback control of 3.29 is only valid for $\mu_1 > 0$. However, time-axis state control often involves forth and back movements to compensate for displacements along the time axis. A control system that is valid for values $\mu_1 < 0$ is required for these control policies. Here, an equivalent system to 3.29 and compatible with $\mu_1 < 0$ is deduced. The basic strategy is to mirror the whole state space across the time axis, in the case of the unicycle that would be the x-axis, such that backwards movement in the original system becomes forward movement in the equivalent system (Figure 3.7).

The process of the previous section is repeated but with a negated x-axis $x' = -x$, that is, $\xi_1' = -\xi_1$. In the problem of the unicycle, the θ state variable is not only affected by the mirror operation, but it is also necessary to consider a 180° turn to account for the backwards motion. Thus, $\theta' = (\pi - \theta) - \pi$, so $\theta' = -\theta$, that is, $\xi_3' = -\xi_3$. Consequently, the chained form state variables are

$$\boldsymbol{s}' = \begin{bmatrix}s_1'\\ s_2'\\ s_3'\end{bmatrix} = \boldsymbol{f}(\boldsymbol{\xi}') = \begin{bmatrix}-\xi_1\\ -\tan(\xi_3)\\ \xi_2\end{bmatrix} = \begin{bmatrix}-s_1\\ -s_2\\ s_3\end{bmatrix}.\tag{3.30}$$

The inputs are also affected. Linear velocity is reversed due to the backwards movement, whilst angular velocity is also reversed due to the symmetry operation across the x axis, so

$$\boldsymbol{u}' = \begin{bmatrix}u_1'\\ u_2'\end{bmatrix} = \begin{bmatrix}-u_1\\ -u_2\end{bmatrix}.\tag{3.31}$$

The inputs in chained form are, thus,

$$\begin{bmatrix}v_1'\\ v_2'\end{bmatrix} = \begin{bmatrix}\cos(\xi_3') & 0\\ 0 & \dfrac{1}{\cos^2(\xi_3')}\end{bmatrix}\begin{bmatrix}u_1'\\ u_2'\end{bmatrix} = \begin{bmatrix}-u_1\cos(\theta)\\ \dfrac{-u_2}{\cos^2(\theta)}\end{bmatrix} = \begin{bmatrix}-v_1\\ -v_2\end{bmatrix}.\tag{3.32}$$

The state equation in chained form of the equivalent system is

$$\dot{\boldsymbol{s}}' = \begin{bmatrix} -\dot{s}_1 \\ -\dot{s}_2 \\ \dot{s}_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ s_2 \end{bmatrix} v_1 + \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} v_2 = \begin{bmatrix} 1 \\ 0 \\ s_2' \end{bmatrix} v_1' + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} v_2'. \tag{3.33}$$

Thus, by transforming the sensor observations with $\begin{bmatrix} \tau' & \boldsymbol{\zeta}' \end{bmatrix}^\mathsf{T} = \begin{bmatrix} -s_1 & -s_2 & s_3 \end{bmatrix}^\mathsf{T}$, we get to a controllable system in time-axis control form,

$$\begin{aligned} \frac{d}{d\tau'} \begin{bmatrix} \zeta_3' \\ \zeta_2' \end{bmatrix} &= \begin{bmatrix} \zeta_2' \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mu_2' \\ \frac{d\tau'}{dt} &= \mu_1' \end{aligned} \tag{3.34}$$

where the actual control inputs are obtained by substituting into the developments from the previous section:

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \dfrac{-\mu_1'}{\cos(\theta)} \\ -\mu_1'\mu_2' \cos^2(\theta) \end{bmatrix}. \tag{3.35}$$

As mentioned above, this transformation is unknown to the controller, so $\boldsymbol{v} = \boldsymbol{u}$ is considered, as will be explained in Section 5.2.

3.2 Non-holonomic systems

Non-holonomic systems are a class of dynamic systems which have constraints about which values their general coordinates can take or about the evolution of these values in time. When a system is non-holonomic, it is not possible to apply Langrangian mechanics, which are based on energy considerations as opposed to forces, to solve the dynamics of the system. Control of these systems becomes more difficult, and are often non-linear, compared to holonomic systems. In this section, non-holonomic systems are characterized.

3.2.1 Holonomic constraints

When the selection of generalized coordinates is redundant, that is, there are more coordinates than the strictly minimum required to describe the configuration state of the system, then some of the coordinates are dependent on the others and on time. This dependency is termed a *constraint* on the system. There are many ways to categorize constraints. Here, we focus on the categorization that distinguishes *holonomic constraints* and *non-holonomic constraints*. The importance of this categorization is that holonomic constraints indicate redundancy of generalized coordinates, whilst non-holonomic constraints indicate limitations inherent to the structure of the problem. In other words, a holonomic constraint indicates that the choice of generalized coordinates is not minimal and that a more concise set of generalized coordinates is possible.

A holonomic constraint is a constraint that obeys the general equation

$$f(q_1, \cdots, q_n, t) = 0. \tag{3.36}$$

for some choice of generalized coordinates $\boldsymbol{q} = \begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix}^\mathsf{T}$. A holonomic system is a dynamic system whose constraints are all holonomic. That is, the constraints of a holonomic system all depend merely on the values of the generalized coordinates and on time. The definition of holonomic system is independent of the choice of generalized coordinates because as long as the constraint can be expressed as (3.36) in some choice of generalized coordinates, the constraint is holonomic.

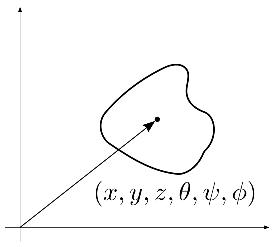


Figure 3.8: The rigid body in space has a minimum of 6 states. Canonically, these are the three cartesian coordinates of the centroid and three angle coordinates.

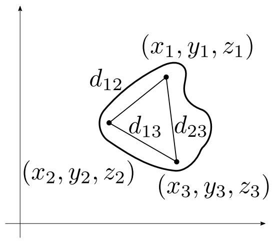


Figure 3.9: The state of the rigid body can be represented with redundant generalized coordinates by the cartesian coordinates of three points.

For example, consider a rigid body B in free space (Figure 3.8). The number of degrees of freedom is 6: three for position (x, y, z) and three for rotation (θ, ψ, ϕ). Therefore, the minimal set of generalized coordinates has six parameters, for example

$$\boldsymbol{q}_B = (x, y, z, \theta, \psi.\phi), \tag{3.37}$$

and no constraints. A rigid body in free space may also be described with the location of three fixed points $\boldsymbol{p}_1 = (x_1, y_1, z_1)$, $\boldsymbol{p}_2 = (x_2, y_2, z_2)$ and $\boldsymbol{p}_3 = (x_3, y_3, z_3)$, as shown in Figure 3.9. Then, the system is defined by nine generalized coordinates

$$\boldsymbol{q}'_B = (x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3) \tag{3.38}$$

and the constraints imposed by the rigid body, which are constant distance between points in the rigid body:

$$\|\boldsymbol{p}_1 - \boldsymbol{p}_2\| - d_{12} = 0 \tag{3.39}$$
$$\|\boldsymbol{p}_1 - \boldsymbol{p}_3\| - d_{13} = 0 \tag{3.40}$$
$$\|\boldsymbol{p}_2 - \boldsymbol{p}_3\| - d_{23} = 0. \tag{3.41}$$

The choice $\boldsymbol{q}'_B$ of generalized coordinates consists of nine parameters and three holonomic constraints, which, when substracted to each other, results in the six degrees of freedom of the rigid body.

3.2.2 Non-holonomic constraints

A non-holonomic constraint is a constraint that is not holonomic (!). A non-holonomic system is a dynamic system that has non-holonomic constraints. So, if a constraint cannot be expressed as (3.36) in any choice of generalized coordinates, then the constraint is non-holonomic. A dynamic system with at least one non-holonomic constraint is a non-holonomic system. An example of a non-holonomic constraint is that whose equation is defined by an inequality:

$$f(q_1, \cdots, q_n, t) \geq 0. \tag{3.42}$$

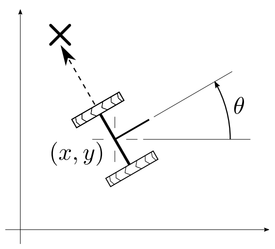


Figure 3.10: The non-holonomic constraint in the unicycle obeys the Pfaffian-form equation $\tan(\theta)\dot{x} - \dot{y} = 0$.

Non-holonomic constraints of this kind can be solved by partitioning the problem into two. Firstly, when $f(q_1, \cdots, q_n, t) > 0$ (note there is no equality), the system is equivalent to one without the constraint. Secondly, when $f(q_1, \cdots, q_n, t) = 0$, the constraint is holonomic and thus reducible. Therefore, solving for these kind of non-holonomic systems is equivalent to solving two other systems, one without the constraint and another with a holonomic constraint.

Another example of non-holonomic constraint is any defined by

$$f(q_1, \cdots, q_n, \dot{q}_1, \cdots, \dot{q}_n, t) = 0. \tag{3.43}$$

Sometimes, despite a constraint is expressed with first derivative parameters as in (3.43), it is still a holonomic constraint. It is only when (3.43) cannot be transformed to a constraint in the form of (3.36) that it is a non-holonomic constraint. Often, (3.43) only needs to be integrated with respect to time to show that the constraint can be expressed with (3.36), thus if (3.43) can be integrated, then it is a holonomic constraint.

The kind of non-holonomic constraint dealt with in this thesis is of the first derivative kind (3.43). More specifically, with non-holonomic constraints in Pfaffian form:

$$A(\boldsymbol{q})\dot{\boldsymbol{q}} = 0. \tag{3.44}$$

If the constraint is a true non-holonomic constraint, these kind of constraints are characterized by different integral values depending on the integration trajectory. So the reason why it is a non-integrable constraint is because a unique integral does not exist and it cannot be converted to (3.36).

3.2.3 The unicycle as a non-holonomic system

The unicycle is a non-holonomic system with three degrees of freedom and one non-holonomic constraint (Figure 3.10). The canonical generalized coordinates are

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}. \tag{3.45}$$

The lateral movement in the unicycle is restricted, which is expressed in the form of a non-holonomic constraint of the first derivative kind in Pfaffian form (3.44),

$$\arctan \frac{\dot{y}}{\dot{x}} = \theta \quad \Rightarrow \quad \tan(\theta)\dot{x} - \dot{y} = 0. \tag{3.46}$$

This constraint is not integrable because, considering two different integral trajectories X_1 and X_2,

$$\int_{X_1} \left(\frac{dy(x)}{dx} - \tan\theta \right) dx \neq \int_{X_2} \left(\frac{dy(x)}{dx} - \tan\theta \right) dx, \tag{3.47}$$

so the unicycle is indeed a non-holonomic system.

Chapter 4

Methods and application to the unicycle

This chapter describes some general concepts and methods that are well known in the field of robotics and are used as mathematical tools in later chapters. Firstly, the derivation of the Jacobian of the sensor space with respect to the control inputs is derived from the state and output equations. Secondly, the method of function approximation used for the learning algorithm in Chapter 5 is described. Then, the control policy, which is based on linear state-space control, used for evaluating the method is presented.

4.1 Jacobian

As explained in Section 3.1.4, sensor space corresponds to the state space of the transformation of generalized coordinates given by the output equation (2.2) $\boldsymbol{s} = H(\boldsymbol{q})$. The state equation in sensor space is obtained by deriving (2.2) with respect to time:

$$\dot{\boldsymbol{s}} = \frac{d\boldsymbol{s}}{dt} = \frac{\partial H(\boldsymbol{q})}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} \overset{(2.1)}{=} \frac{\partial H(\boldsymbol{q})}{\partial \boldsymbol{q}} F(\boldsymbol{q})\boldsymbol{u}. \tag{4.1}$$

Since the kinematics F and the sensor configuration H are unknown, they can be grouped to a single unknown factor

$$J(\boldsymbol{q}) := \frac{\partial H(\boldsymbol{q})}{\partial \boldsymbol{q}} F(\boldsymbol{q}), \tag{4.2}$$

which is defined as the Jacobian of the generalized coordinates given by the sensor outputs with respect to the control input $\boldsymbol{u}$. Because $\boldsymbol{s}$ and $\boldsymbol{q}$ are assumed isomorphic (Section 2.1), $\boldsymbol{q}$ can be replaced with $\boldsymbol{s}$ in (4.2). Thus, the state equation of a dynamic system in sensor space is

$$\dot{\boldsymbol{s}} = J(\boldsymbol{s})\boldsymbol{u}, \tag{4.3}$$

with the Jacobian J unknown. Designing a controller for (4.3) requires having a model of $J(\boldsymbol{s})$. This thesis proposes an approach to obtain $J(\boldsymbol{s})$ automatically under the problem settings described in Section 2.1.

4.2 Function approximation by Least Squares of Gaussian Radial Basis Functions

The proposed method uses Gaussian RBF [22] for approximating the sensorimotor mapping from sensor space to chained-form space. In this section, f refers to the approximated function and ϕ_k refers to the Gaussian kernel at point k.

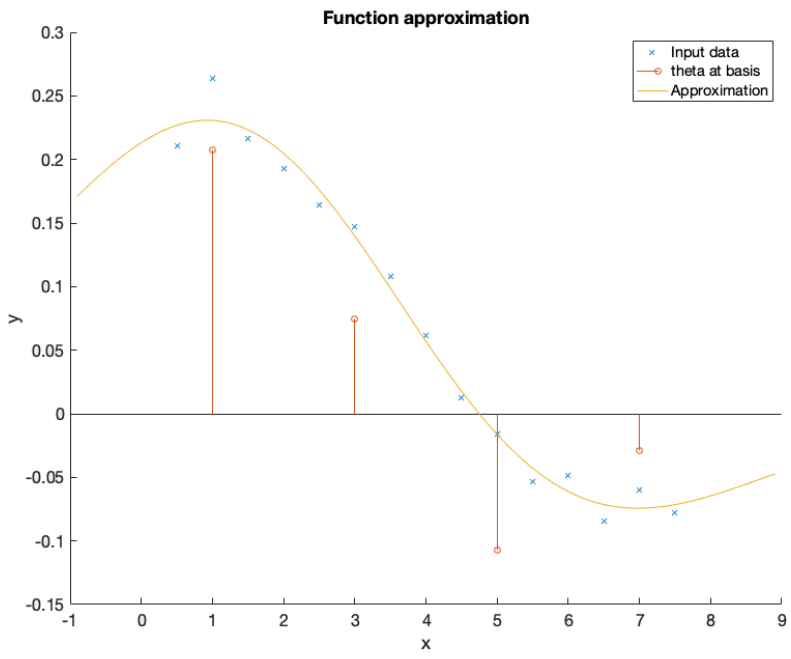


Figure 4.1: Example of a function approximation with unregularized Gaussian radial basis functions. Input data, coefficient values θ of the kernels, and the plot of the approximated function are represented.

Description	Constant	Index letter
Number of components of a state vector	D	$i \in \{1, \dots, D\}$
Number of entries in the dataset	N	$j \in \{1, \dots, N\}$
Number of basis	P	$k \in \{1, \dots, P\}$

Table 4.1: Meaning of variables derivation of in Gaussian RBF

4.2.1 Description

Function approximation is the process of inferring a limited set of parameters, which collectively define a function f, from a set of pairs of values $(\boldsymbol{x}, y)$, called the training dataset, such that $f(\boldsymbol{x})$ is as close as possible to y for every pair $(\boldsymbol{x}, y)$ in the dataset (Figure 4.1). There are many ways to perform function approximation. Here the least squares regression[1] by linear combination of Gaussian radial basis functions (Gaussian RBF) is described and implemented, with multidimensional $\boldsymbol{x}$ and unidimensional y.

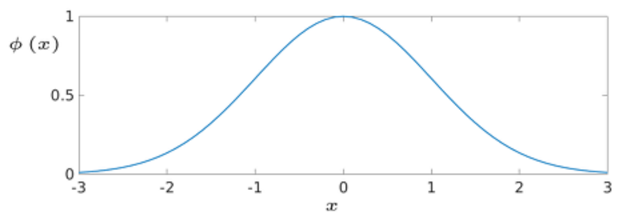
Let $\{(\boldsymbol{x}_i, y_i) | i \in \{1, \dots, N\}\}$ be a dataset of pairs of values with N components (points), where $\boldsymbol{x}_i \in \mathbf{R}^D$ is the input vector and $y_i \in \mathbf{R}$ is the expected output. D is the dimension of the input vector. The objective is to find a function $f : \mathbf{R}^D \longrightarrow \mathbf{R}$ that approximates the unknown underlying relation between $\boldsymbol{x}_i$ and y_i over all the dataset.

4.2.2 Mathematical derivation

The convention for the variables used in this section is indicated in Table 4.1. A Gaussian kernel ϕ_k with center in $\boldsymbol{b}_k \in \mathbb{R}^D$ is defined as (Figure 4.2)

$$\phi_k(\boldsymbol{x}) := \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{\|\boldsymbol{x} - \boldsymbol{b}_k\|^2}{2 \cdot \sigma^2}). \tag{4.4}$$

[1]Risi Kondor (2004) Regression by Linear Combination of Basis Functions.



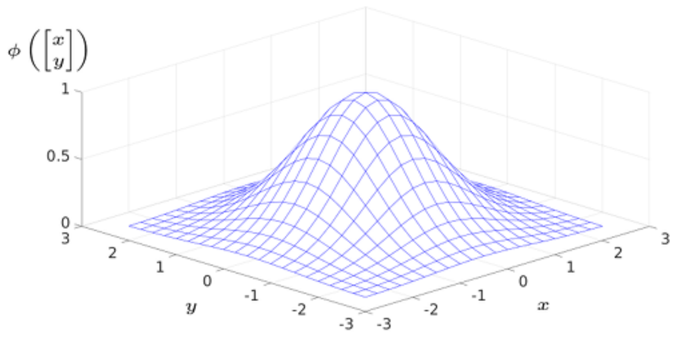


Figure 4.2: Plot of Gaussian kernel $\phi(\boldsymbol{x})$ centered in $\boldsymbol{x} = 0$ and standard deviation $\sigma = 1$ for $\boldsymbol{x} \in \mathbf{R}$ (above) and $\boldsymbol{x} \in \mathbf{R}^2$ (below).

f is the function approximation using a linear combination of Gaussian kernels with centers at $\begin{bmatrix}\boldsymbol{b}_1 & \cdots & \boldsymbol{b}_P\end{bmatrix}$:

$$f(\boldsymbol{x}) = \sum_{k=1}^{P} \theta_k \phi_k(\boldsymbol{x}). \tag{4.5}$$

In this regression problem, the parameters $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 & \theta_2 & \cdots & \theta_P \end{bmatrix}^\intercal$ are calculated using least squares on $f(\boldsymbol{x})$. Generally, there would be an additional term $\theta_0\phi_0(\boldsymbol{x})$ where $\phi_0(\boldsymbol{x}) = 1$ is the *bias* term. Here, the approximation f has no bias, *i.e.* $\theta_0 = 0$. To calculate $\boldsymbol{\theta}$, it is assumed that the problem is not underspecified, that is, there are at least as many data points as basis functions $N \geq P$. $\boldsymbol{\theta}$ is solved by defining a loss function and then minimizing it. For each dataset point $(\boldsymbol{x}_i, y_i)$, the loss function is

$$L_j(y_j, f(\boldsymbol{x}_j)) \coloneqq \frac{1}{2}(y_j - f(\boldsymbol{x}_j))^2, \tag{4.6}$$

and the total loss function is

$$L(\boldsymbol{\theta}) = \sum_{j=1}^{N} L_j(y_j, f(\boldsymbol{x}_j)) = \frac{1}{2}\sum_{j=1}^{N} (y_j - \sum_{k=1}^{P} \theta_k\phi_k(\boldsymbol{x}_j))^2. \tag{4.7}$$

The minimum of this multivariate function is found where the gradient is 0: $\nabla_{\boldsymbol{\theta}} L = 0$. Let us calculate the gradient component-wise $\dfrac{\partial L}{\partial \theta_g} = 0$, where $g \in \{1, \ldots, P\}$:

$$\frac{\partial L}{\partial \theta_g} = \frac{1}{2}\sum_{j=1}^{N} 2\left(y_j - \sum_{k=1}^{P} \theta_k\phi_k(\boldsymbol{x}_j)\right)\phi_g(\boldsymbol{x}_j) = \tag{4.8}$$

$$= \left[\sum_{j=1}^{N} \phi_g(\boldsymbol{x}_j)y_j - \sum_{j=1}^{N} \phi_g(\boldsymbol{x}_j)\sum_{k=1}^{P} \theta_k\phi_k(\boldsymbol{x}_j)\right] = 0. \tag{4.9}$$

In matrix form,

$$\begin{bmatrix}\phi_g(\boldsymbol{x}_1) & \cdots & \phi_g(\boldsymbol{x}_N)\end{bmatrix}\begin{bmatrix}y_1\\ \vdots \\ y_N\end{bmatrix} -$$

$$-\begin{bmatrix}\phi_g(\boldsymbol{x}_1) & \cdots & \phi_g(\boldsymbol{x}_N)\end{bmatrix}\begin{bmatrix}\begin{bmatrix}\phi_0(\boldsymbol{x}_1) & \cdots & \phi_P(\boldsymbol{x}_1)\end{bmatrix}\cdot\begin{bmatrix}\theta_1\\ \vdots \\ \theta_N\end{bmatrix}\\ \vdots \\ \begin{bmatrix}\phi_0(\boldsymbol{x}_N) & \cdots & \phi_P(\boldsymbol{x}_N)\end{bmatrix}\cdot\begin{bmatrix}\theta_1\\ \vdots \\ \theta_N\end{bmatrix}\end{bmatrix} = 0. \tag{4.10}$$

Let us now define

$$Q(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N) := \begin{bmatrix}\phi_0(\boldsymbol{x}_1) & \cdots & \phi_P(\boldsymbol{x}_1)\\ \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_N) & \cdots & \phi_P(\boldsymbol{x}_N)\end{bmatrix}, \tag{4.11}$$

$$\boldsymbol{\theta} := \begin{bmatrix}\theta_1\\ \vdots \\ \theta_N\end{bmatrix}, \tag{4.12}$$

and consider the multivariate case for $\boldsymbol{y}$:

$$\boldsymbol{y} := \begin{bmatrix}y_1\\ \vdots \\ y_N\end{bmatrix}. \tag{4.13}$$

Note that $\boldsymbol{x}_i$ is a row vector with D elements but $\boldsymbol{y}$ is a column vector of N components. All components of $\nabla_{\boldsymbol{\theta}} L$ may be grouped into a single matrix equation

$$\nabla_{\boldsymbol{\theta}} L = \begin{bmatrix}\dfrac{\partial L}{\partial \theta_0}\\ \vdots \\ \dfrac{\partial L}{\partial \theta_P}\end{bmatrix} = Q^\intercal \boldsymbol{y} - Q^\intercal Q\boldsymbol{\theta} = 0. \tag{4.14}$$

Assuming that the points of the dataset and the basis are unique, then Q has rank P. Hence, $Q^\intercal Q$ is invertible because $N \geq P$, so

$$\boldsymbol{\theta} = (Q^\intercal Q)^{-1} Q^\intercal \boldsymbol{y}. \tag{4.15}$$

Therefore, the approximated function is:

$$f(\boldsymbol{x}) = \begin{bmatrix}\phi_0(\boldsymbol{x}) & \cdots & \phi_P(\boldsymbol{x})\end{bmatrix}\boldsymbol{\theta} = Q(\boldsymbol{x})\boldsymbol{\theta}. \tag{4.16}$$

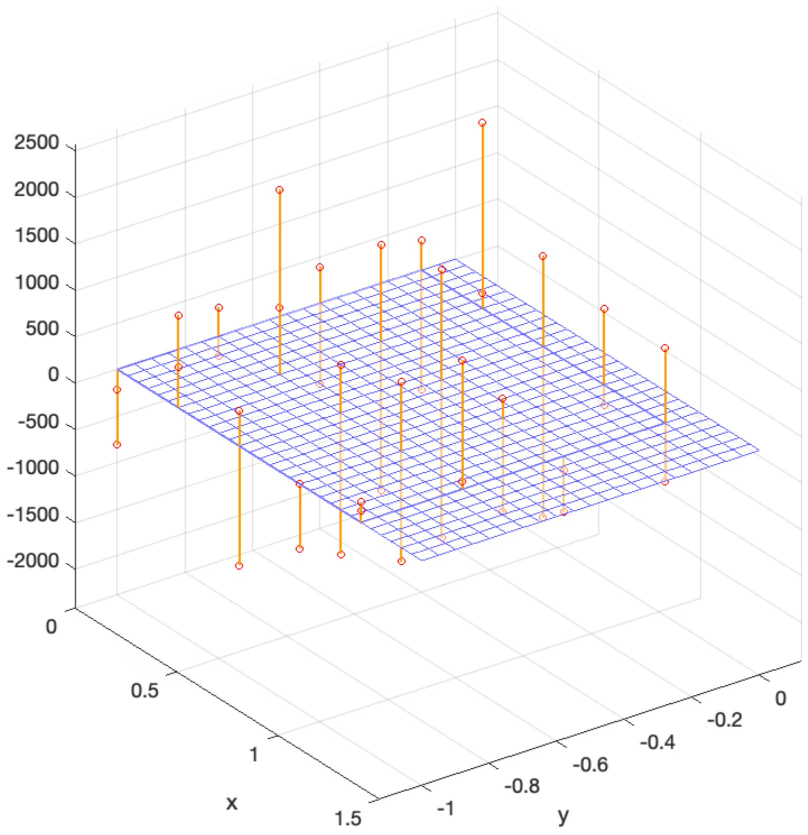


Figure 4.3: Basis theta values overfitting example in a function approximation from $\mathbf{R}^2$ to $\mathbf{R}$. The underlying function maps each (x, y) belonging to the blue grid to a random value between -1 and 1. The red circles are the values of the θ_k coefficients at their respective basis point and the orange lines are the projections onto the resulting approximated function.

A common problem in all function approximation methods is *overfitting*. Gaussian RBF is no exception to this problem (See Figure 4.3). Overfitting occurs when the approximated function has a good fit of the training data points, but its accuracy is poor at any other point. For example, an approximation that models small stochastic deviations is not desirable because it misses the smoothness of the underlying process. In such cases, a term that penalizes complexity is added to the function approximation. This way of mitigating overfitting problems is called *regularization*.

In Least Squares, the so-called L_2 regularization method penalizes the absolute value of the parameters $\boldsymbol{\theta}$ by adding the squared sum of the parameters to the loss function, which becomes

$$L_2(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \frac{\lambda}{2} \left\|\boldsymbol{\theta}\right\|^2, \tag{4.17}$$

where $\lambda \geq 0$ is a parameter that controls the amount of regularization. The new term is carried along the calculation of the gradient:

$$\frac{\partial L_2}{\partial \theta_g} = \frac{\partial L}{\partial \theta_g} + \lambda \theta_g. \tag{4.18}$$

Thus,

$$\nabla_{\boldsymbol{\theta}} L_2 = \nabla_{\boldsymbol{\theta}} L + \boldsymbol{\theta} = Q^\intercal \boldsymbol{y} - Q^\intercal Q \boldsymbol{\theta} + \lambda \boldsymbol{\theta} = 0 \tag{4.19}$$

Solving for $\boldsymbol{\theta}$,

$$\boldsymbol{\theta} = (Q^\intercal Q - \lambda I)^{-1} Q^\intercal \boldsymbol{y} \tag{4.20}$$

As can be easily deduced from the previous equation, the greater the value of λ is, the smaller the parameters $\boldsymbol{\theta}$ become, with $\lambda = 0$ having no regularization and $\lambda \to \infty$ taking the parameters to the trivial solution $\boldsymbol{\theta} = \mathbf{0}$, therefore $f(\boldsymbol{x})$=0.

4.2.3 Analysis of Gaussian-RBF parameters

Figure 4.4 shows a comparison of the approximated function by Gaussian RBF under different values of standard deviation σ and regularization parameter λ, in the case where all the data is interpolated. Figure 4.5 shows the same comparison, but the base span is smaller so some portions of the approximated function is extrapolated. In both cases, low values of the regularization term produces high variations in the value of the θ_k coefficients, which is even more noticeable with higher values of σ. A good value of λ seems to be between $\lambda = 0.5$ and $\lambda = 1$. On the other hand, small values of σ, that is, smaller than around half the distance between bases, result in poor approximation between the kernels, because the kernels are not width enough to cover these spaces, unless the values of the θ_k coefficients are very large and the error at the center of the kernels becomes even larger. Larger values of σ seem to have an effect in regions where there are already other kernels, so it is not desirable either because of the seemingly arbitrary values of the resultant ϕ_k. The optimum value for σ will depend on each specific function being approximated. In general, values between $\sigma = 0.5d$ and $\sigma = 1d$, with d being the distance between bases, yield the most accurate approximations.

4.3 State-space control of time-axis control form

In this section the mathematical derivation of the control law applied to a linear system is deduced from state-space control theory[9]. This control law is to be applied to the control-state part of the time-axis state equations (3.15).

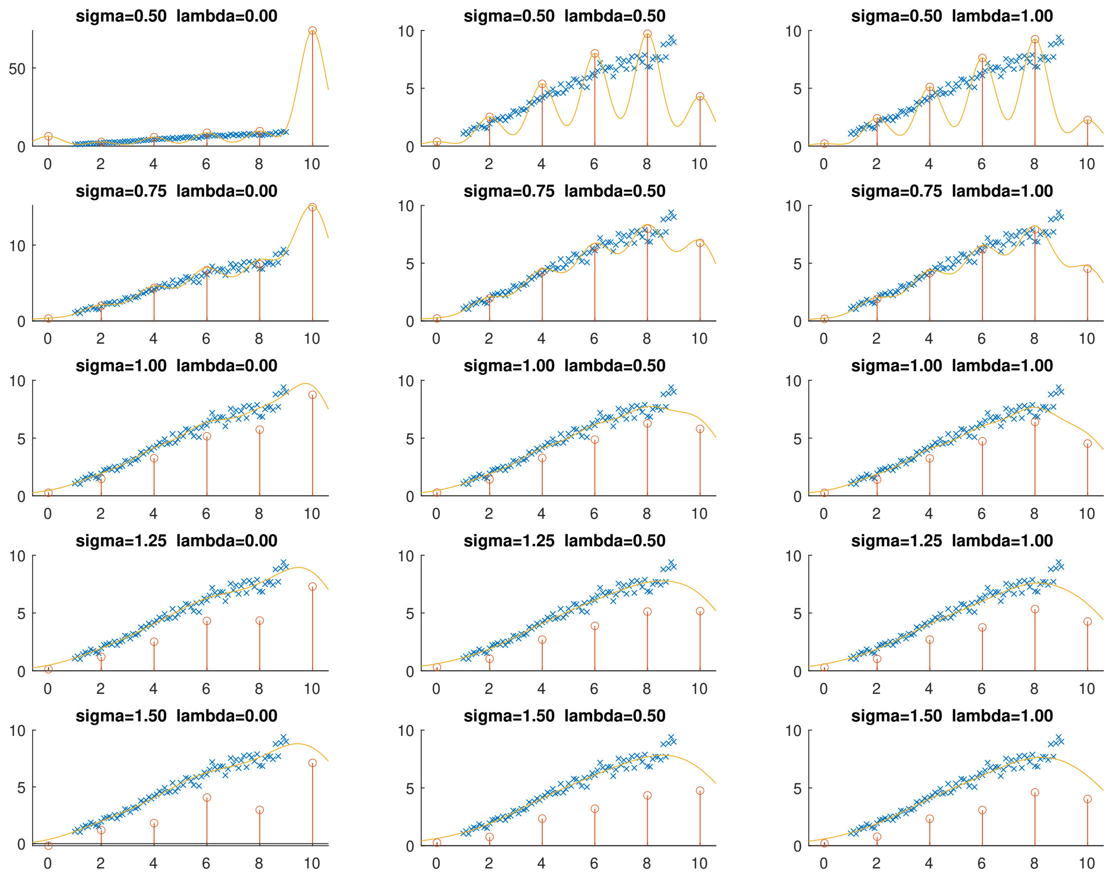


Figure 4.4: Analysis of the λ parameter in function approximation method by Gaussian RBF when all the data is bound to the RBF bases.

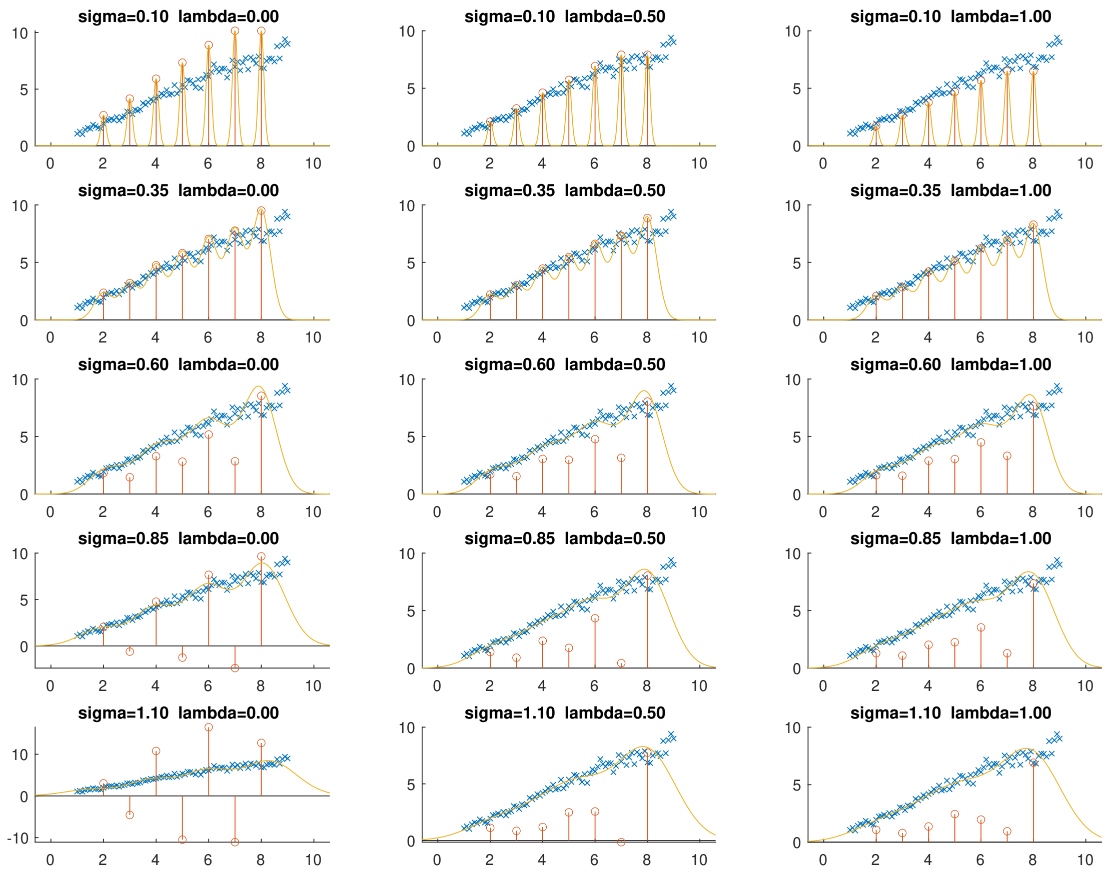


Figure 4.5: Analysis of the λ parameter in function approximation method by Gaussian RBF when some of the data is not bounded by the RBF bases.

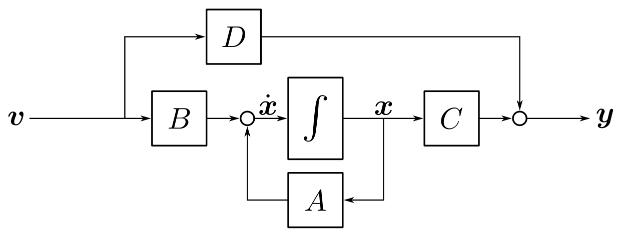


Figure 4.6: Graphical representation of a linear system.

Linear systems

The state and output equations for a linear system, depicted in Figure 4.6, are

$$\begin{cases} \dot{\boldsymbol{x}} = & A(t)\boldsymbol{x}(t) + B(t)\boldsymbol{u}(t) \\ y(t) = & C(t)\boldsymbol{x}(t) + D(t)\boldsymbol{u}(t). \end{cases} \tag{4.21}$$

where $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^\mathsf{T}$ is the state vector and $\boldsymbol{u}$ is the input vector. Considering that A is constant ($A \neq A(t)$), the solution to this system under null input $\boldsymbol{u} = \mathbf{0}$ is the transition equation

$$\boldsymbol{x} = \boldsymbol{x}_0 e^{A(t-t_0)}. \tag{4.22}$$

$\boldsymbol{x}$ is controllable if the gramian controllability matrix

$$Q = \begin{bmatrix} B \mid AB \mid A^2B \mid \cdots \mid A^nB \end{bmatrix} \tag{4.23}$$

is rank $n \coloneqq \dim(\boldsymbol{x})$. Now, the general expression for a univariate system under Laplacian transform is

$$y = \frac{b_n s^n + \ldots + b_1 s + b_0}{s^n + \ldots + a_1 s + a_0} u, \tag{4.24}$$

which may be expressed in phase variables by choosing the first state variable

$$x_1 = \frac{1}{s^n + a_{n-1}s^{n-1} + \ldots + a_0}, \tag{4.25}$$

and the rest of the variables

$$\begin{aligned} x_2 = &\ \dot{x}_1 \\ x_3 = &\ \dot{x}_2 \\ &\vdots \\ x_n = &\ \dot{x}_{n-1}. \end{aligned} \tag{4.26}$$

This is the state equation (4.21) in phase variables. Matrix-wise, (4.21) is expressed in phase form with

$$A = \begin{bmatrix} 0 & 1 & \cdots & & 0 \\ 0 & 0 & \ddots & & \vdots \\ \vdots & & \ddots & 1 & \\ & & & 0 & 1 \\ -a_0 & -a_1 & \cdots & -a_{n-2} & -a_{n-1} \end{bmatrix} ;\ B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}. \tag{4.27}$$

Design of the feedback loop

Considering that the characteristic polynomial of the system is

$$p_r(s) = s^n + a_{n-1}s^{n-1} + \ldots + a_1 s + a_0, \tag{4.28}$$

and that the desired characteristic polynomial of the system without feedback loop is

$$p_r(s) = s^n + \alpha_{n-1}s^{n-1} + \ldots + \alpha_1 s + \alpha_0, \tag{4.29}$$

a state-feedback loop is added to the system with coefficients

$$K = \begin{bmatrix} k_1 & k_2 & \ldots & k_n \end{bmatrix}. \tag{4.30}$$

The block diagram of the original system with a state-feedback control loop is shown in Figure 4.7. Operating with matrices, an equivalent system is obtained that has the form of the

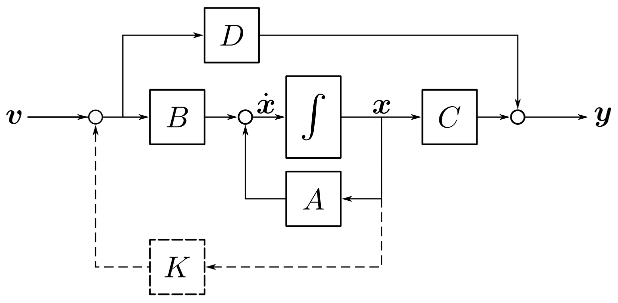


Figure 4.7: Feedback control loop (dashed lines) attached to the linear system.

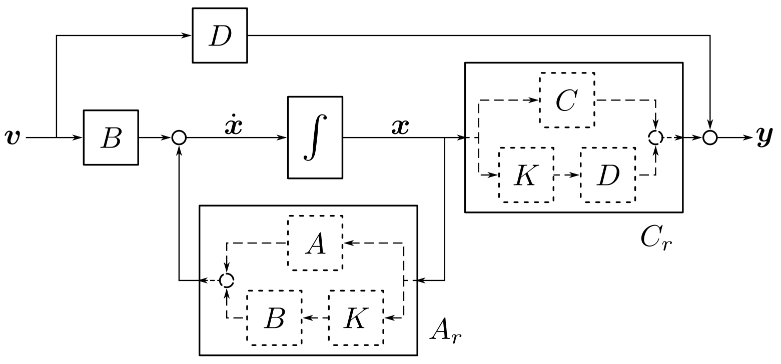


Figure 4.8: Graphical representation of the feedback-controlled linear system converted into the canonical representation.

canonical state and output equations (4.21), as depicted in Figure 4.8:

$$\begin{cases} \dot{x} = & A_r(t)x(t) + B(t)u(t) \\ y(t) = & C_r(t)x(t) + D(t)u(t). \end{cases} \tag{4.31}$$

where

$$A_r := A + BK, \tag{4.32}$$

and

$$C_r := C + DK. \tag{4.33}$$

The design of the feedback loop is not affected by the output equation, so the output equation is dropped. Simplifying A_r,

$$\begin{aligned} A_r &= \begin{bmatrix} 0 & 1 & \cdots & & 0 \\ 0 & 0 & \ddots & & \vdots \\ \vdots & & \ddots & 1 & \\ & & & 0 & 1 \\ -a_0 & -a_1 & \cdots & -a_{n-2} & -a_{n-1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} k_1 & k_2 & \cdots & k_n \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 & \cdots & & 0 \\ 0 & 0 & \ddots & & \vdots \\ \vdots & & \ddots & 1 & \\ & & & 0 & 1 \\ k_1 - a_0 & k_2 - a_1 & \cdots & k_{n-1} - a_{n-2} & k_n - a_{n-1} \end{bmatrix}. \end{aligned} \tag{4.34}$$

Therefore, calculating the characteristic polynomial of A_r and equalizing to $p_r(s)$, the following set of simultaneous equations of order n is obtained:

$$a_{i-1} - k_i = \alpha_{i-1}. \tag{4.35}$$

Solving for k_i, the input

$$u(t) = \begin{bmatrix} k_1 & k_2 & \cdots & k_n \end{bmatrix} \boldsymbol{x}(t) \tag{4.36}$$

is applied on the original system (4.21), with $k_i = a_{i-1} + \alpha_{i-1}$. This way the system has the dynamic behavior as desired at (4.29).

Application to the time-axis control form of the unicycle

The state equation for the unicycle in time-axis control form (3.20) is, considering μ_1 >0:

$$\frac{d}{dt}\begin{bmatrix} z_3 \\ z_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} z_3 \\ z_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}\mu_2. \tag{4.37}$$

The controllability gramian yields a controllable system:

$$Q = \begin{bmatrix} B \mid AB \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{4.38}$$

The feedback loop constants are $K = \begin{bmatrix} k_1 & k_2 \end{bmatrix}$, hence

$$A_r = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}\begin{bmatrix} k_1 & k_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ k_1 & k_2 \end{bmatrix}. \tag{4.39}$$

The desired characteristic polynomial is

$$p_r(s) = \left(\frac{1}{(s-p_1)(s-p_2)}\right)^{-1} = \left(\frac{1}{s^2 - (p_1 - p_2)s + p_1 p_2}\right)^{-1}. \tag{4.40}$$

Equalizing to the characteristic polynomial of A_r and solving for K,

$$\begin{aligned} \alpha_0 = &\quad p_1 p_2 \\ \alpha_1 = &\quad -(p_1 + p_2) \end{aligned}, \tag{4.41}$$

which results in

$$\begin{aligned} k_0 = &\quad p_1 p_2 \\ k_1 = &\quad -(p_1 + p_2) \end{aligned}. \tag{4.42}$$

Therefore, the feedback system becomes

$$\frac{d}{dt}\begin{bmatrix} z_3 \\ z_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -p_1 p_2 & p_1 + p_2 \end{bmatrix}\begin{bmatrix} z_3 \\ z_2 \end{bmatrix}, \tag{4.43}$$

and the desired input is

$$\begin{aligned} \mu_1 &\quad > 0 \\ \mu_2 &\quad = \begin{bmatrix} k_1 & k_2 \end{bmatrix}\begin{bmatrix} z_3 \\ z_2 \end{bmatrix} = -p_1 p_2 z_3 + (p_1 + p_2) z_2. \end{aligned} \tag{4.44}$$

For negative time scales $\mu_1 < 0$ (Section 3.1.6.3), the control policy becomes

$$\begin{aligned} \mu_1' &= -\mu_1 \\ \mu_2' &= -(p_1 + p_2) z_2 - p_1 p_2 z_3 \end{aligned}. \tag{4.45}$$

Similarly to the equations for positive time scales, the control policy is only valid for $\mu_1' > 0$, that is, $\mu_1 < 0$.

Chapter 5

Learning algorithm

This is the core chapter of the thesis. It contains the main contributions of the research, which are the theoretical developments that follow.

The learning algorithm of sensorimotor mapping is divided in two parts, or stages. The first stage deals with the manoeuvrability of the robot. During this stage, the Jacobians of the system around the initial position are explored. The initial position of the system is selected by the designer or operator of the robot. This position is considered the origin in chained-form state space. The behavior of the Jacobian with respect to the control inputs is analyzed and a virtual input is constructed, such that the virtual input effectively moves the robot in the forbidden direction of the non-holonomic constraint. The second stage deals with the exploration of the sensor space with the purpose of constructing a dataset to calculate an approximated function between the sensor mapping and some generalized coordinates in chained-form.

Intuitively, the first stage is akin to obtaining the kinematics of the robot, whilst the second stage resembles learning of the sensor configuration, yet both remain entangled as a single unknown for the controller.

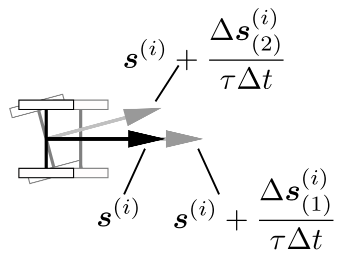
5.1 Stage 1: Virtual input

In this section, the first stage of the learning algorithm is described. In this stage, the proposed controller learns efficient patterns to navigate the sensor space, despite the non-holonomic constraints. It is necessary that the controller learns how each control input affects the sensor observations to maximize mobility. Nevertheless, in the unicycle example, there are only two control inputs for three degrees of freedom, so a method to navigate the third degree of freedom is required. This section describes how to design a *virtual* control input $\boldsymbol{u}_{(3)}$ that maximizes movement along this third degree of freedom, which corresponds to the forbidden direction imposed by the non-holonomic constraint, while minimizing variations of the sensor observations along the directions immediately controllable by the two regular control inputs $\boldsymbol{u}_{(1)}$ and $\boldsymbol{u}_{(2)}$ at the initial state.

5.1.1 Jacobian

The first step is to retrieve the Jacobians (Section 4.1) of the system.

Let p be the state of the system, identified by sensor observation $\boldsymbol{s}^{(p)} \in \mathbf{R}^3$. Let $\dot{\boldsymbol{s}}^{(p)}_{(\vartheta)}$ be the derivative of the sensor with respect to time when the system is applied control input $\boldsymbol{u}_{(\vartheta)}$, with $\vartheta \in \{1, 2\}$, $\boldsymbol{u}_{(1)} = \begin{bmatrix} 1 & 0 \end{bmatrix}^\mathsf{T}$ and $\boldsymbol{u}_{(2)} = \begin{bmatrix} 0 & 1 \end{bmatrix}^\mathsf{T}$, as described in Section 2.1. Then, the



$s^{(i)} + \dfrac{\Delta s^{(i)}_{(2)}}{\tau\Delta t}$
$s^{(i)}$
$s^{(i)} + \dfrac{\Delta s^{(i)}_{(1)}}{\tau\Delta t}$

Figure 5.1: Starting from the sensor observation at $s^{(i)}$, two more sensor observations are required to obtain the Jacobian at $s^{(i)}$, one for each control input.

Jacobian for the unicycle (Section 2.2) is

$$J := \left.\frac{\partial \dot{s}}{\partial u}\right|_{n\times m} = \begin{bmatrix} \dfrac{\partial \dot{s_1}}{\partial u_1} & \dfrac{\partial \dot{s_1}}{\partial u_2} \\ \dfrac{\partial \dot{s_2}}{\partial u_1} & \dfrac{\partial \dot{s_2}}{\partial u_2} \\ \dfrac{\partial \dot{s_3}}{\partial u_1} & \dfrac{\partial \dot{s_3}}{\partial u_2} \end{bmatrix} = \begin{bmatrix} j_1 & j_2 \end{bmatrix}. \tag{5.1}$$

The state equation of the system in sensor coordinates at state p is

$$\dot{s}^{(p)}_{(\vartheta)} = g(s^{(p)})\, u_{(\vartheta)} = g_1(s^{(p)})\, u_{(\vartheta),1} + g_2(s^{(p)})\, u_{(\vartheta),2}. \tag{5.2}$$

Considering control input $u = \tau_1 u_{(1)} = \begin{bmatrix}\tau_1 & 0\end{bmatrix}^\mathsf{T}$, then $\dot{s}^{(p)}_{(1)} = g_1(s^{(p)})\tau_1$ so

$$\frac{\partial \dot{s}^{(p)}_{(1)}}{\partial u} = \frac{\partial}{\partial u} g_1(s^{(p)})\, u_1 + \frac{\partial}{\partial u} g_2(s^{(p)})\, u_2 = g_1(s^{(p)})\,\frac{\partial u_1}{\partial u_1} = \frac{\dot{s}^{(p)}}{\tau_1} \approx \frac{\Delta s^{(p)}}{\tau_1 \Delta t}. \tag{5.3}$$

Similarly for $u = \tau_2 u_{(2)} = \begin{bmatrix}0 & \tau_2\end{bmatrix}^\mathsf{T}$ and substituting for $\dot{s}^{(p)}_{(2)}$, we have that

$$J^{(p)} = \begin{bmatrix} j^{(p)}_1 & j^{(p)}_2 \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \dot{s}^{(p)}}{\partial u_1} & \dfrac{\partial \dot{s}^{(p)}}{\partial u_2} \end{bmatrix} \approx \begin{bmatrix} \dfrac{\Delta s^{(p)}_1}{\tau_1\,\Delta t} & \dfrac{\Delta s^{(p)}_2}{\tau_2\,\Delta t} \end{bmatrix}, \tag{5.4}$$

which is the formula for retrieving the approximated Jacobian of the system at state p with discrete sensor observations (Figure 5.1). Thus, the Jacobian is obtained by retrieving three sensor observations: At state $s^{(p)}$ and at state $s^{(p)}$ after applying a brief input $u_{(\vartheta)}$ to the system for every input $\vartheta \in \{1,2\}$. After every movement, the system must backtrack the control inputs by applying $-u_{(\vartheta)}$ for the same amount of time. This action nullifies the possible effect of trajectory history in the calculation of the state of the system after retrieving the Jacobian.

The procedure to design the virtual input continues by retrieving the Jacobian at periodic intervals along the trajectory defined by applying control input $u_{(\varphi)}$ for each φ, starting from the initial state $s^{(0)}$, as depicted in Figure 5.2. The goal is to find a state $s^{(\star)}$ whose Jacobian $J^{(\star)}$ contains a component $j^{(\star)}_\psi$ with $\psi \in \{1,2\}$ such that

$$\det \begin{bmatrix} j^{(0)}_1 & j^{(0)}_2 & j^{(\star)}_\psi \end{bmatrix} = \pm 1, \tag{5.5}$$

assuming that the Jacobians are normalized. Then, the input $u_{(\psi)}$ applied at state $s^{(\star)}$ is the forbidden direction by the non-holonomic constraint. Therefore, the optimal value of $j^{(\star)}_\psi$ in

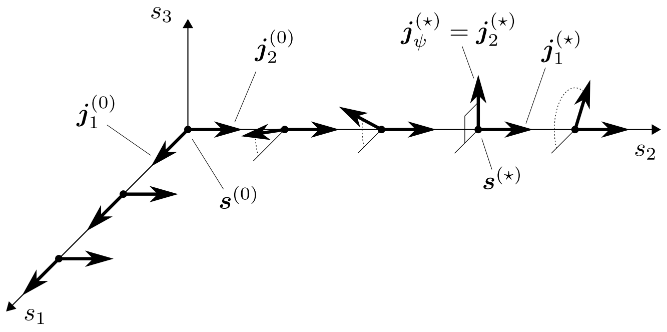


Figure 5.2: The Jacobian is obtained at several points along the axis defined by each control input at $\boldsymbol{s}^{(0)}$ in search for the most orthogonal Jacobian component $\boldsymbol{j}_\psi^{(\star)}$ at $\boldsymbol{s}^{(\star)}$.

(5.5), considering discrete sensor observations and finite number of Jacobians, is given by

$$(\boldsymbol{s}^{(\star)}, \psi) = \underset{\boldsymbol{s}^{(p)}, i \in \{1,2\}}{\arg\max} \left| \det \begin{bmatrix} \boldsymbol{j}_1^{(0)} & \boldsymbol{j}_2^{(0)} & \boldsymbol{j}_i^{(p)} \end{bmatrix} \right|. \tag{5.6}$$

$\boldsymbol{s}^{(\star)}$ and ψ are the necessary data to construct the virtual input.

5.1.2 Construction of virtual input

Let $\boldsymbol{u}_{(\boldsymbol{s}^{(\star)})}$ be the control input required to take the system from state $\boldsymbol{s}^{(0)}$ to $\boldsymbol{s}^{(\star)}$ and $\Delta t_{(\boldsymbol{s}^{(\star)})}$ the time required to reach $\boldsymbol{s}^{(\star)}$. Then, the virtual input $\boldsymbol{u}_{(3)}$ is defined as the sequence of inputs

$$\boldsymbol{u}_{(3)}\Delta t \Longleftrightarrow \boldsymbol{u}_{(\star)}\Delta t_{(\star)}; \quad \boldsymbol{u}_{(\psi)}\Delta t; \quad -\boldsymbol{u}_{(\star)}\Delta t_{(\star)}. \tag{5.7}$$

Virtual input $\boldsymbol{u}_{(3)}$ effectively moves the non-holonomic system in the forbidden direction as if the non-holonomic constraint did not exist. We will use inputs $\boldsymbol{u}_{(1)}$, $\boldsymbol{u}_{(2)}$ and virtual input $\boldsymbol{u}_{(3)}$ to navigate the sensor space freely in the next stage. Conceptually, it is similar to a holonomic system with an additional input.

5.2 Stage 2: Exploration of sensor space

In this stage, the learning controller explores the sensor space by applying control inputs to obtain a dataset for function approximation from sensor space to chained-form space, where the system can be feedback controlled.

5.2.1 Sensor space data sampling

Since the trajectory of non-holonomic systems depends on the history of the control inputs, the sensor space may not be explored randomly. There must be some rule to the applied inputs such that the learning controller can assure that the state in chained form remains consistent with the sensor observations. In other words, the sensor observations will always be trustworthy with respect to the state of the system, but the equivalent state in chained-form will be incoherent unless the sensor space is explored systematically. The reason is that with arbitrary control inputs, it is possible that by applying the same control inputs to the chained-form equivalent system as the ones applied to the actual system, then the same sensor observation is mapped to many states in chained form, or conversely, many sensor observations end up being mapped to a single state in chained form. In these cases, the mapping between sensor space and chained-form control space is not isomorphic and hence the method cannot provide reliable sensor readings for feedback control.

With that in mind, the rules for sensor exploration are

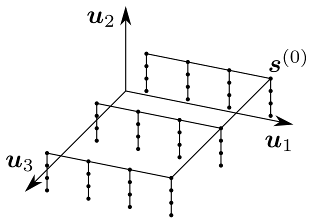


Figure 5.3: Example of fixed pattern of exploration of sensor space during stage 2. In this example, $u_{(3)}$ is applied first, then $u_{(1)}$ and $u_{(2)}$, each of them in a loop to explore the sensor space systematically.

- The starting position is arbitrary inside the region of exploration and it is decided by the operator.
- The system must backtrack all its movements to the starting position during learning.
- In order to reduce the cumulative errors of the actuators, applying the virtual input should be minimized because this control input is a sequence of other control inputs.
- The system may only be controlled with *pure* control inputs, that is, only a single element in $\boldsymbol{u} = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^\mathsf{T}$ may be non-zero at any given time.

Guided by these rules, the following algorithm was developed.

```
loop $\Delta t_1 : 0 \ldots \Delta T_1$
    apply input $\boldsymbol{u}_{(3)}$ for $\Delta t_1$;
    loop $\Delta t_2 : 0 \ldots \Delta T_2$
        apply input $\boldsymbol{u}_{(\psi)}$ for $\Delta t_2$;
        loop $\Delta t_3 : 0 \ldots \Delta T_3$
            apply input $\boldsymbol{u}_{(\boldsymbol{s}^{(\star)})}$ for $\Delta t_3$;
            dataset $\leftarrow$ dataset $\cup$ $(\boldsymbol{s}, (\boldsymbol{u}_{(3)}; \boldsymbol{u}_{(\psi)}; \boldsymbol{u}_{(\boldsymbol{s}^{(\star)})}))$;
            backtrack $\boldsymbol{u}_{(\boldsymbol{s}^{(\star)})}$;
        backtrack $\boldsymbol{u}_{(\psi)}$;
    backtrack $\boldsymbol{u}_{(3)}$;
```

Algorithm 1: Pseudo code for sampling the sensor space. $\boldsymbol{u}_{(\psi)}$ and $\boldsymbol{u}_{(\boldsymbol{s}^{(\star)})}$ correspond to the inputs obtained in Section 5.1.

Figure 5.3 shows an example of exploring a sensor space where each axis represents a control input, *i.e.* when a control input is applied, it is shown as a movement along the corresponding axis.

5.2.2 Function approximation

After exploration of the sensor space by fixed pattern, the dataset $\{(\boldsymbol{s}_i, \boldsymbol{z}_i)\}$ is available, where $i \in [1, \ldots, N]$, N is the number of sensor observations $\boldsymbol{s}_i$ and $\boldsymbol{z}_i$ is the state of the equivalent system in chained-form. The mapping

$$\boldsymbol{z} = \boldsymbol{\phi}(\boldsymbol{s}) = \begin{bmatrix} \phi_1(\boldsymbol{s}) \\ \phi_2(\boldsymbol{s}) \\ \phi_3(\boldsymbol{s}) \end{bmatrix} \tag{5.8}$$

is inferred from the dataset $\{(\boldsymbol{s}_i, \boldsymbol{z}_i)\}$ by Gaussian RBF as described in Section 4.2.

The number of kernels should not be greater than the number of sensor observations to avoid problems related to underspecification. Since the sensor space is sampled by following a grid, the number of kernels in each dimension is chosen to be at most the number of sensor observations in the corresponding dimension. For example, if 5 sensor observations are taken in each dimension of the unicycle system, the number of kernels should be at most $5 \times 5 \times 5 = 125$. The selection of the location of the kernels follows a lattice in sensor space. The size of the lattice is the same as the smallest cube or hypercube that encloses all the sensor observations.

At this point, the learning process is finished. The system should be linearly controllable in the regular regions (*i.e.* not singular, Section 3.1.5) if the method was successful. The linear controller φ acts on the equivalent system on chained-form, so it must use the transformed sensor observations:

$$\boldsymbol{u} = \varphi(\phi(\boldsymbol{s})). \tag{5.9}$$

For evaluation purposes of the validity of ϕ, the linear controller described in Section 4.3 is used in this thesis, but any other linear controller should be equally valid.

Chapter 6

Simulation

The approach described previously was developed and tested in a simulated environment under the Matlab toolset[1]. In this chapter, the architecture and code of the implementation of the approach are described in detail, followed by the exposition and analysis of the results of the simulated experiment.

6.1 Implementation

There are two directories used in the Matlab implementation: matlab and matlab/Orthogonal_spaces, where there is the Matlab source code for the simulation and some other auxiliary source code files written during the development of the implementation. The final source files and their relation are shown in the communication diagram of Figure 6.1.

The most important file of the implementation is the script showcase.m, which controls the flow of the simulation and dispatches calls to the rest of the functions. unicycle.m hosts the simulated system in a Matlab object, ensuring that the internal state of the unicycle is not accessible by the controllers. The execution flow of showcase.m goes over the following points, in the same order:

1. Initialization of parameters, auxiliary variables, and configuration values.
2. Calling max_orth(), which handles the first stage of the method. max_orth() explores each axis separately, samples differential sensor readings to obtain the Jacobian at each point, and calculates the optimum values for $\boldsymbol{u}_{(s^{(\star)})}$ and $\Delta t_{(s^{(\star)})}$ of the virtual input u_3.
3. Calling xplr_axis() to obtain the fixed sequence of inputs necessary to explore the sensor space, based on the output of max_orth().
4. Looping over each input recursively as indicated by xplr_axis(), including the virtual input, and storing the sensor observation values and the state values of the system, assumed in chained form, in a dataset.
5. Obtaining the set of points of a grid in sensor space that will be used to locate the kernels of the Gaussian RBF by calling basis_grid(). The size of the grid depends on the maximum and minimum values of each dimension in the dataset. Therefore, the bases are placed in a three-dimensional grid that extends to the smallest box containing all data in sensor space.
6. Calling gaussian_RGB(), where the function approximation by least squares is computed on the dataset obtained previously. gaussian_RGB() returns ϕ as a function object. ϕ should be called with one or more sensor observations and it returns the approximated values in chained-form space.

[1]https://www.mathworks.com/

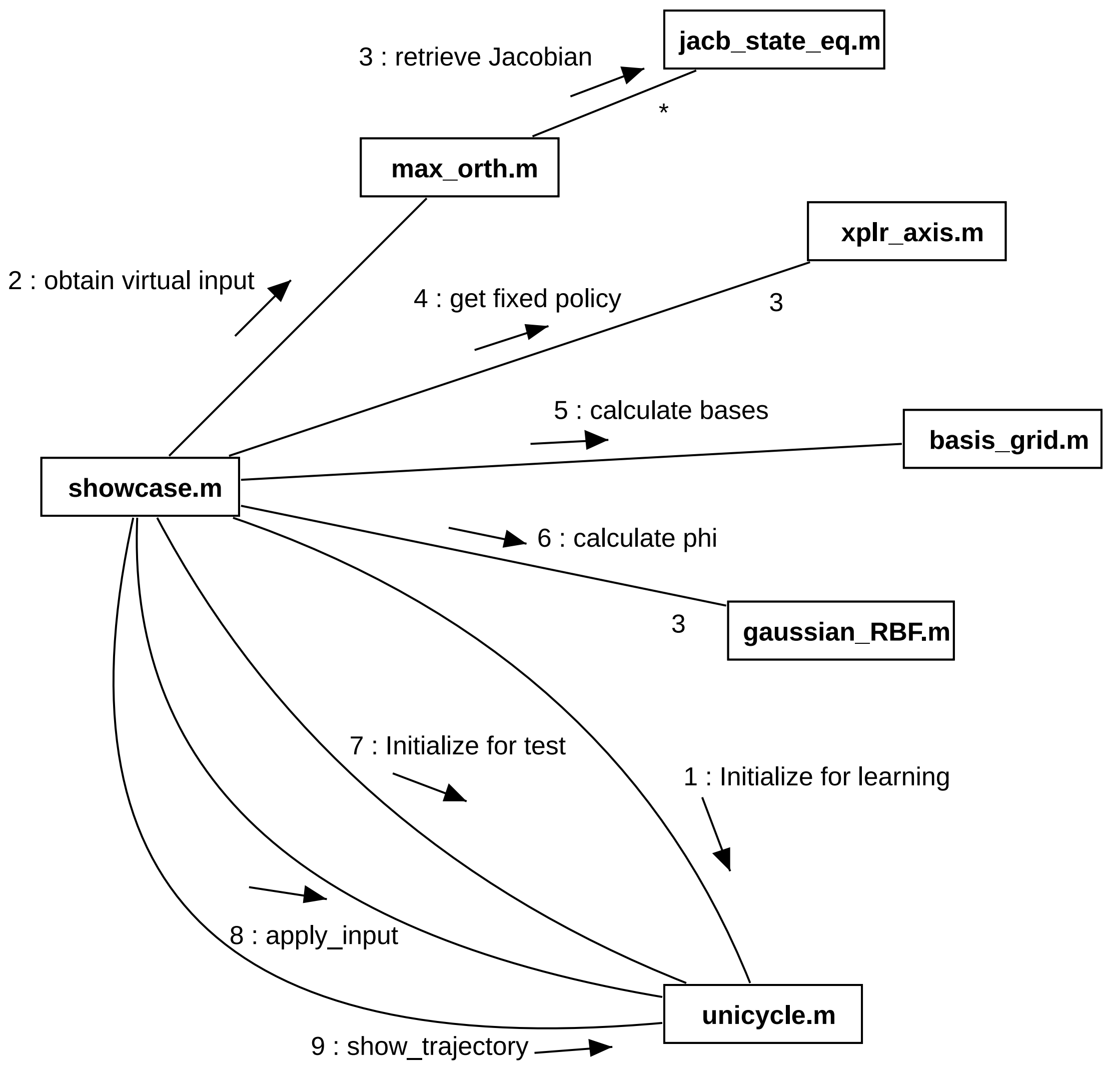


Figure 6.1: UML communication diagram of MATLAB implementation for simulation. Calls to unicycle .apply_input() in the learning stages have been omitted.

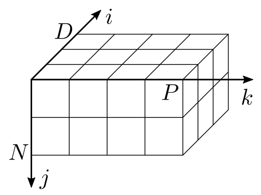


Figure 6.2: Conventions for dimensions in `Gaussian_RBF()` internal variables. The cube represents a $N \times P \times D = 2 \times 4 \times 3$ multidimensional matrix. N: Number of entries in the dataset. P: Number of bases. D: Number of components in the state of the system.

7. Setting up another unicycle object with a different initial state, but same motor and sensor configuration.
8. The new unicycle system is feedback controlled by time-axis control method usingthe previously computed ϕ.
9. Finally, the trajectories followed by the unicycle are plotted and the intermediate variables, dataset and other auxiliary data are stored in Matlab global workspace.

As well as for the simulation, Matlab was used to develop and test the methods of this research. Particularly, the most time-consuming developments in Matlab other than the simulation were the analysis, extension and test of the seminal implementation in the initial stages of this research, and the implementation, test and analysis of the function approximation algorithm by Gaussian RBF, which resulted in the addition of the regularization coefficient as described in Section 4.2.2, to solve issues with overfitting.

The implementation of the function approximation algorithm will be described in more detail due to its complexity.

Implementation of approximated function by Gaussian RBF

The function approximation by least squares of Gaussian RBF implemented in Matlab operates with three-dimensional matrices to simplify the code. Each dimension of these matrices conveniently index each of the 3 ranges $\{1, \ldots, N\}$, $\{1, \ldots, P\}$ and $\{1, \ldots, D\}$, as shown in Figure 6.2. The constants N, P, and D correspond to the number of entries in the dataset, the number of bases and the number of components, respectively. The input arguments need to be converted to this multidimensional convention before operating with them. The output of the function is a function reference to the approximated function as a lambda object and ready to be used. The implementation is found in the file `gaussian_RBF.m` in the form of a Matlab function.

```
function [f,theta] =
        gaussian_RBF(X_training,Y_data,X_basis,sigma,lambda)

N=size(X_training,2);
P=size(X_basis,2);
D=size(X_training,1);
```

There are 5 input arguments which are described here in detail. The convention applied is that the first dimension (rows) is used for the components of the vector and the second dimension (columns) is used for the data points. Therefore, a column vector represents the state at one point, whilst a row vector represents one of the state components at several points.

`X_training` The input component $\boldsymbol{x}_j$ of the training dataset $\{(\boldsymbol{x}_j, y_j)\}$ at point j. The size of this matrix is $D \times N$.

`Y_training` The output component y_j of the training dataset $\{(\boldsymbol{x}_j, y_j)\}$ at point j. This matrix is a row vector $1 \times N$.

`X_basis`	The location of the bases. Same format as `X_training`, but size is $D \times P$.
`sigma`	The standard deviation σ for all bases.
`lambda`	The parameter λ used in regularization. Use `lambda = 0` for no regularization.

The conventions for the matrix dimensions used internally differ from the conventions for the arguments because matrix manipulation becomes easier. Thus, before any operation the arguments should be transformed to the internal convention. Internally, the matrix dimensions are $N \times P \times D$ ($N \times 1 \times 1$ for the training dataset `Y`):

1. (Row vectors) Data points.
2. (Column vectors) Basis points.
3. (Depth vectors) Vector components.

Unfortunately, the original and desirable version of the code for function $Q(\boldsymbol{x}_1, \ldots)$, which is

```
% var_X=syms('X',[N P D]);
% Q(var_X) = exp( -1/(2*sigma)
          * sum( (permute(var_X,[2 3 1])-B).^2 ,3) );
```

cannot be used due to Matlab limitations in using symbolic values for multi-dimensional matrices. Alternatively, the following permutations are applied to the inputs to circumvent the need for symbolic variables.

```
X = permute(X_training, [2 3 1]);
B = permute(X_basis, [3 2 1]);
Y = Y_data';
```

Additionally, `X` and `B` are replicated in their respective unused dimensions to automate the calculations with matrix operators only.

```
X = repmat(X, 1,P,1);
B = repmat(B, N,1,1);
```

With this transformation of the inputs, the equations (4.4), (4.11) and (4.20) can be coded directly: Q is calculated, which results in a $N \times P$ matrix that matches the dimensions of the theoretical development, and the result is used to calculate the linear parameters $\boldsymbol{\theta}$. `theta` has dimensions $P \times 1$, as expected. The regularization parameter λ is also applied in this step.

```
Q = exp( -1/(2*sigma^2) * sum((X-B).^2,3) );
theta = (Q'*Q + lambda*eye(size(Q,2)))\Q' * Y;
```

This concludes the calculation of the $\boldsymbol{\theta}$ coefficients, but `gaussian_RBF.m` can be made easier for users if an actual function is returned, rather than just the coefficients.

Therefore, the lambda function that will hold the approximation is defined and returned. The arguments and output matrices of `f` follow the same conventions used for the input arguments of `gaussian_RBF()`, so Q needs recalculation every time that the approximated function is called to take the input format into account. The transformations are the same as above.

```
function q=Q_f(var_x)
    N=size(var_x,2); % Not the same N as above.
    X = permute(var_x,    [2 3 1]);
    X = repmat(X, 1,P,1);
    B = repmat(preb, N,1,1);
    s = zeros(size(X(:,:,1)));
    for i=1:D
        s = s + ( X(:,:,i)-B(:,:,i) ).^2;
    end
    q = exp(-1/(2*sigma^2)*s);
end
```

Finally, (4.5) is generalized to N input vectors and a reference to the approximated function f is returned. Thanks to the selection of matrix dimensions, it may be coded simply as:

```
f=@(input) (Q_f(input) * theta)';
```

The development of function approximation by Gaussian RBF was tested by cross validation. Cross validation involves randomly partitioning a dataset in training data and testing data. The training data is used for constructing the $\boldsymbol{\theta}$ coefficients and the test data, smaller, is used to assess the correctness of the generalization. The partitioning is repeated until all data is used at least once as testing data. The implementation of cross validation is found in k_fold.m, but it is not used in the final version of the simulation.

6.2 Results

The state equation for the unicycle used as the target simulated robot is

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{6.1}$$

$$\boldsymbol{s} = H(\boldsymbol{q}) \quad . \tag{6.2}$$

where u_1 indicates linear speed and u_2 indicates rotational speed. Equations (6.1) and (6.2) are not exposed to the learning and control algorithms: the only data available to modify is $\boldsymbol{u}$ and the only readable data is $\boldsymbol{s}$. Three variations of H were tested, which were designed to be isomorphic mappings in the region of interest:

$$H_1(\boldsymbol{q}) = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \quad H_2(\boldsymbol{q}) = \begin{bmatrix} \sinh(y) \\ e^x \\ \arctan(\theta) \end{bmatrix}, \quad H_3(\boldsymbol{q}) = \begin{bmatrix} x + e^y \\ e^x - y \\ \theta^3 \end{bmatrix} \quad . \tag{6.3}$$

These were the simulation parameters for the three cases: the initial state was $\boldsymbol{q}_0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^{\mathsf{T}}$, with nine samples per axis and a separation of 0.25 units in the first stage. For constructing the dataset, a total of 5^3 samples, 5 per axis, were taken in the range $[-2, 2]$. The Gaussian RBF approximation had 5^3 bases, a standard deviation of $1.5d$ where d is the minimum distance between bases, and regularization term $\lambda = 0.5$. The feedback controlled stage for assesment had intial position $(x, y, \theta) = (-2, 0.5, \pi/4)$, control poles of $p_1 = p_2 = -5$ and was controlled for 2.5 seconds. The standard deviation for the gaussian kernels for each case were $\sigma_1 = 1.1970$, $\sigma_2 = 0.6643$ and $\sigma_3 = 3.0745$, respectively. The trajectory, observations and gaussian kernel locations in sensor space of the sensor space mapping stage are shown in Figures 6.3 and 6.4.

The system was successfully controlled to the time axis in the three sensor configurations (Figure 6.5). Errors in the z_2-axis of the transformation function $\boldsymbol{\phi}$, corresponding to y in (x, y, θ) space, along the time axis in chained space were negligible ($\phi_2(\tau) = 0 \pm 10^{-13}$ for the three cases). In contrast, errors in the z_3-axis, corresponding to θ in (x, y, θ) space, were significant: $\phi_3(\tau) = 0 \pm 0.0138$ for H_1, $\phi_3(\tau) = 0 \pm 0.0759$ for H_2, and $\phi_3(\tau) = 0 \pm 0.2356$ for H_3. The control law could correct the deviations resulting from inaccuracies in the approximation of $\boldsymbol{\phi}$, which were perceived as perturbations. The perturbations to the controlled trajectory, compared to the ideal mapping approximation with no errors, can be seen in Figure 6.5. Errors resulting from inaccuracies in the actuators were negligible, that is, the positional error comes from rounding errors in floating operations.

The sensor space suffers no transformation in the case of H_1, thus the trajectory in sensor space matches the trajectory in (x, y, θ) space. It can be seen how there is a slight oscillation around axis y. This is attributed to inaccuracies in phi. In the case of H_2, the slight deformation of the sensor space derives in a slight deviation at around $(x, y) = (-1.8, 0.3)$ in (x, y, θ) space

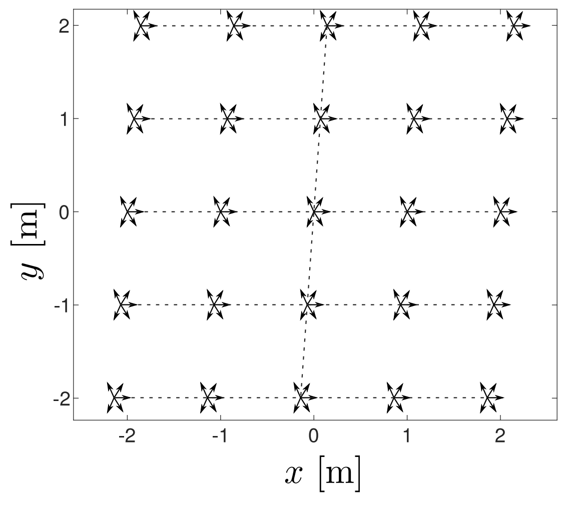


Figure 6.3: The trajectory of the simulated system during sensor space sampling (stage two) in (x, y, θ) coordinates is the same for H_1, H_2 and H_3. Arrows indicate sampled states.

compared to the H_1 case, but appropriately controlled. With respect to H_3, the oscillation in the trajectory cannot be explained by the control poles because the latter are real values. Instead of that, it is explained by in the approximation inaccuracies of the approximated ϕ due to an insufficient number of samples in some regions. These inaccuracies were perceived as perturbations and corrected appropriately by the control law.

An additional simulation was performed to analyse the limits of the method. In the previous simulation, the order of the inputs was not specified in the algorithm, but they were the same in all cases: linear speed and angular speed. In this simulation, the inputs were modified to analyze the effectiveness of the method when the linear component of the input $\boldsymbol{u}$ was slightly affected by rotational movement, and conversely, the rotational component of the input was slightly affected by linear movement. Let R be the ratio between the radius of the left and right wheel:

$$R = \frac{r_l}{r_r} \tag{6.4}$$

In the previous experiment, $R = 1$. The goal of this other experiment was to test the effects of several ratio values on the feedback controlled trajectory. Therefore, the only change to the simulation was in the revisited state equation (3.7)

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dfrac{R+1}{2}\cos\theta \\ \dfrac{R+1}{2}\sin\theta \\ \dfrac{R-1}{2} \end{bmatrix} u_1 + \begin{bmatrix} \dfrac{R-1}{2}\cos\theta \\ \dfrac{R-1}{2}\sin\theta \\ \dfrac{R+1}{2} \end{bmatrix} u_2. \tag{6.5}$$

The rest of the simulation, and specifically, the algorithms described in previous sections, remained unchanged. The ratios tested were 1, 1.2, 1.5, 1.8, 2.2 and 4. The results are shown in Figure 6.6. The learning system was successfully controlled up to $R = 1.8$. When $R = 1$, the results were the same as above, as expected. When $R = 1.2$, the trajectory of the sensor space sampling stage and the sample points have a slight deformation, nonetheless the feedback controller could correct the deformations in the trajectory. When $R = 1.5$, the trajectory is deformed further, so much that the initial state of the feedback controlled system almost matches the time axis. Thus, the feedback controlled system tracks closely the time axis from the starting point. When $R = 1.8$, there is an interesting phenomenon. The trajectory of

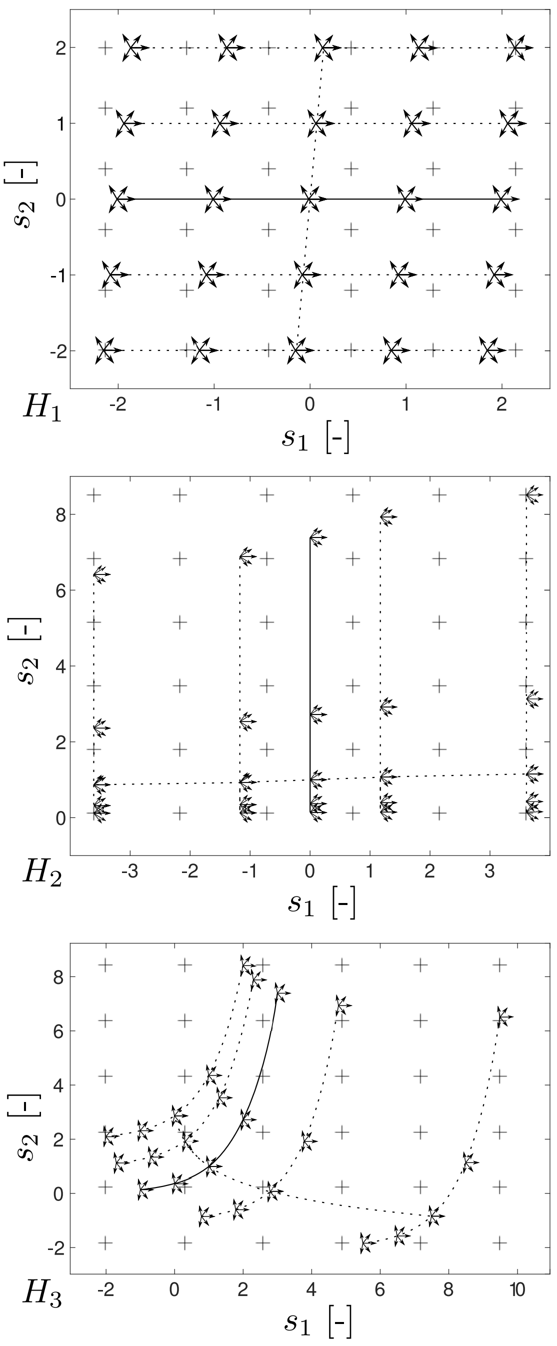


Figure 6.4: Dotted line: Trajectory of the simulated system during sensor space sampling (stage 2) in sensor coordinates for H_1 (top), H_2 (middle) and H_3 (bottom). Solid line: Trajectory of the robot along the line taken as time axis. Arrows: Sampled observations with their rotations indicating units of s_3 in radians. Crosses: Location of Gaussian kernel centers (bases).

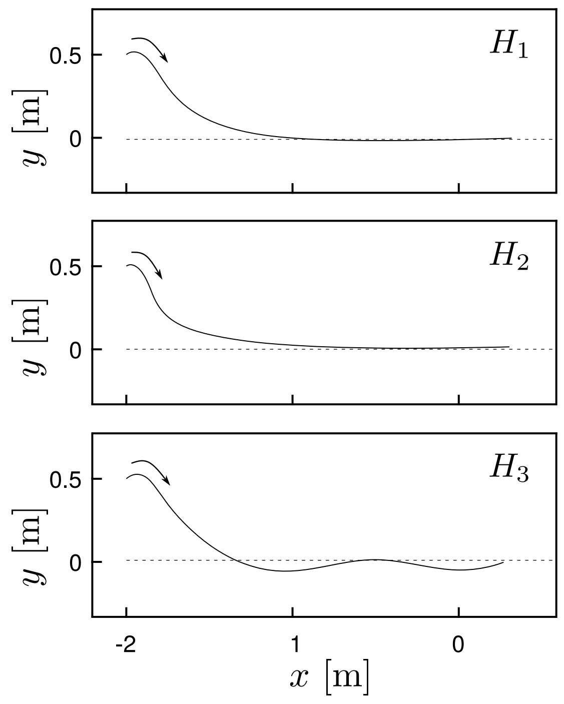


Figure 6.5: Trajectory of the simulated system after learning of sensorimotor mapping for the three sensor configuration transformation functions H_1, H_2 and H_3.

the sensor space sampling stage marginally overlaps with itself. Casually, the trajectory of the feedback controlled system does not pass through any of the problematic regions by a small margin, but it is deformed by a great degree compared to the case $R = 1$. When $R = 2.2$, the region suffering of overlap is much wider and covers the initial state of the feedback controlled trajectory, which becomes uncontrollable due to the sensor mapping not being isomorphic any more in those regions. The function approximation cannot deduce which of the overlapping sampled trajectories should the system follow and the motions become unpredictable. When $R = 4$, the whole sensor space is overlapping and control is impossible.

Consequently, the analysis on different wheel radius ratios confirm the requirement that the function-approximated mapping from sensor space to chained-form space must be isomorphic in the sampled region.

6.3 Comparison with PPO algorithm

The approach as implemented in the simulation was compared with a Reinforcement Learning method called the Proximal Policy Optimization (PPO) algorithm [46].

6.3.1 Introduction to PPO algorithm

Reinforcement Learning (RL) is often used in navigation tasks where the environment of an agent, including the sensorimotor mapping, is not known. The problem setting described at Section 2.1 was adapted to Reinforcement Learning method as much as possible to make a fair comparison with current state of the art methods. The adaptation is not ideal because the problem setting of this research and the problem setting tackled by RL algorithms is not a perfect match, but it could be made close enough to make a meaningful evaluation. PPO was chosen over other methods such as Deep Deterministic Policy Gradient agents (DDPG)[30] or

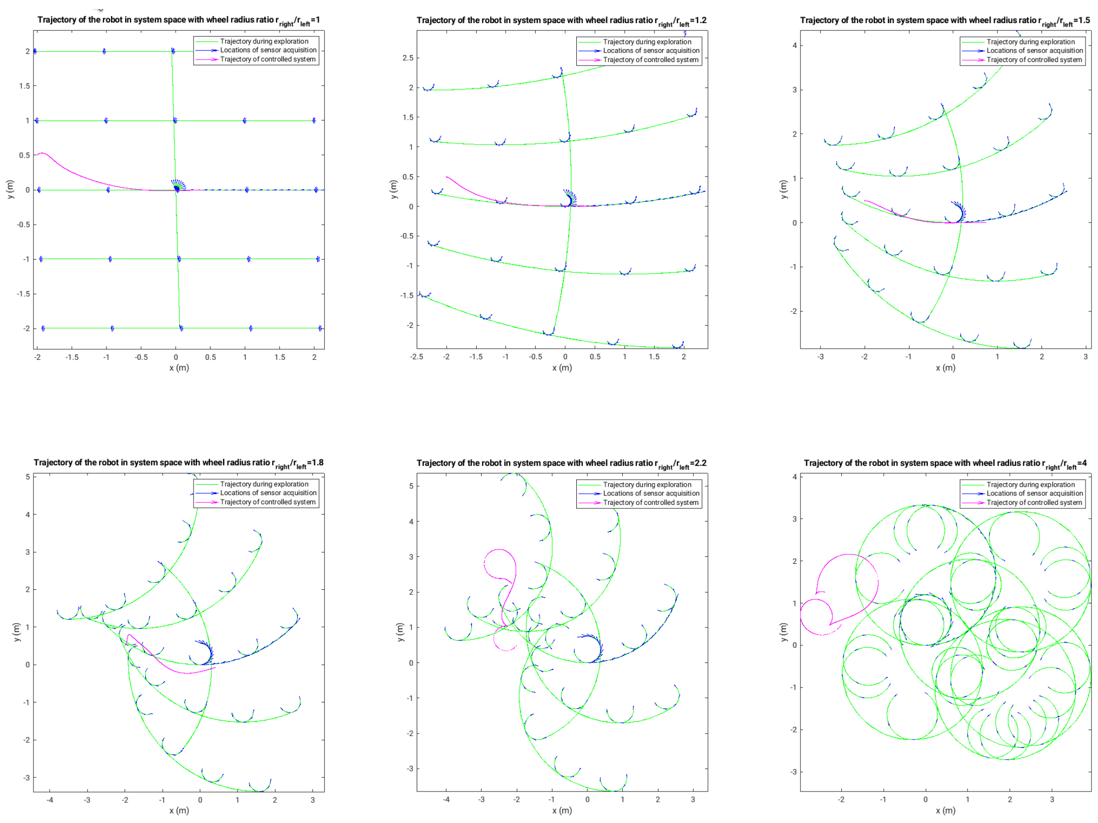


Figure 6.6: Simulation of learning of sensorimotor mapping in a unicycle with left and right wheel radius ratio R. Green: Trajectory during sensor space sampling. Blue: Sample sites. Magenta: Feedback controlled trajectory.

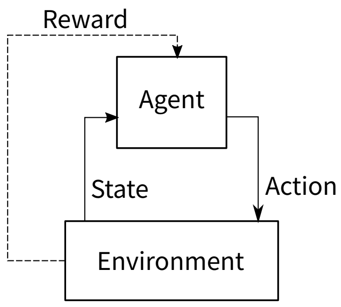


Figure 6.7: Schematic of a Reinforcement Learning agent

Twin-Delayed Deep Deterministic Policy Gradient agents (TD3)[11] due to its simplicity of use and its proven efficacy.

Reinforment Learning is a machine learning algorithm that deals with the problem of maximizing a reward function by trial and error. Every RL algorithm receives two signals from the environment (Figure 6.7): The state signal is used by the RL agent to select the most appropriate action based on the best-known model of the environment, whilst the reward signal is a measure of the performance of the RL agent. The RL agent adjusts its internal model of the environment to improve its performance with respect to the reward signal. RL agents explore the reward response of the environment by exploring the action space by trial and error.

Actor-Critic methods is a RL method where the agent is composed of two separate learning entities which are parameterized with neural networks. In Actor-Critic agents, the *Critic* part estimates the value function. The value function is a magnitude derived from the reward and set for each state. The *Actor* then updates the policy model used for selecting the appropriate action. Critic and Actor are updated often with exploration of the environment by trial and error and by receiving rewards.

The Proximal Policy Optimization algorithm is a class of Actor-Critic RL algorithm that limits the learning rate of the policy functions (Actor and Critic functions) in the regions where the value function has strong variations. The main idea is that overcorrecting the policy functions should not lead to regressive policy update decisions. PPO has the advantage that it supports continuos state values, as opposed to plain RL that only supports discrete state values. Hence, the proposed method was compared with PPO.

6.3.2 Implementation of PPO algorithm

Results of the approach in the simulated environment described in the previous section was compared to the Proximal Policy Optimization (PPO) algorithm. The Matlab implementation of the PPO algorithm found in the Reinforcement Learning Matlab Toolbox was used[2]. This implementation allows for great flexibility in the environments and in the agents. The Matlab Help Center instructions for creating PPO agents was followed[3]. The parameters for the PPO agent were set as indicated in Table 6.1. A custom Matlab environment was coded based on the simulation of the unicycle system, with the same characteristics as the unicycle object created in Section 6.1. Only the first sensor configuration, *i.e.* $H = H_1$ was tested. With the purpose of matching the capabilities of the time-axis linear controller, the PPO agent only controlled the rotational speed u_2 while fixing the linear speed to $u_1 = 0$, therefore control by the PPO agent was only required on the state-control part of the time axis (Section 3.1.6). The agent was stopped as soon as the distance to the desired state $\boldsymbol{z}_d = H(\mathbf{0})$ incremented, that is, as soon as the unicycle system stopped approaching the origin. Since $u_1 = 1$ at any moment, this means that the agent was stopped in every case after crossing the coordinate $x = 0$. The reward

[2]https://nl.mathworks.com/products/reinforcement-learning.html
[3]https://www.mathworks.com/help/reinforcement-learning/ug/ppo-agents.html

Parameter	Value	Description
(x_0, y_0, θ_0)	$\mathcal{N}(\begin{bmatrix}-2\\0.5\\\pi/4\end{bmatrix}, 1/10)$	Initial position of the agent for each trial
r_i	$100\left\Vert\boldsymbol{z}\right\Vert^{-1}$	Reward function
stop	$r_i < r_{i-1}$	Terminal condition
T_s	0.1	Sample time
H	H_1	Sensor configuration
	256	Experience horizon
	64	Mini batch size
γ	0.997	Discount factor
u_1	1	Linear speed of the agent
u_2	*Action by agent*	Rotational speed of the agent

Table 6.1: Parameters of the PPO agent

of the agent was inversely proportional to the distance to the origin, thus the reward function was designed to promote reaching the desired state $\boldsymbol{z}_d = \boldsymbol{0}$.

6.3.3 Results of comparison with PPO algorithm

Several training attempts were needed to obtain a converging PPO agent with reasonable performance. In the final attempt, training was stopped after 138 episodes with a total of 2926 sensor observations and an average reward of 5358 units over the last five agents (Figure 6.8). The last 5 agents were evaluated to controlling the unicycle simulated system from $(x_0, y_0, \theta_0) = (-2, 0.5, \pi/4)$. The 5 agents were run 5000 times each. The average of the closest distance to the origin by the PPO agents was $\mu_{\text{PPO}} = 0.0878$ and standard deviation $\sigma_{\text{PPO}} = 0.0878$. Figure 6.9 shows the trajectory of one of the PPO agents compared to the trajectory of the proposed method. These values contrast to the closest distance to the origin of $\mu_{H_1} = 0.0114$ units in the proposed method.

The differences between PPO and the proposed method are several. To start with, the mobile robot only needs to be positioned once at the initial state with the proposed method, whereas with PPO, the robot needs to be repositioned at the initial state for every training episode, and again for different locations in the desired region of operability. Moreover, training of the proposed method takes less time because the number of observations is far smaller than in PPO training. The proposed method is also safer because it avoids unexpected explotation in the process of sample collection. Furthermore, the computation requirements of the proposed method are small enough to execute the learning algorithms in embedded systems with limited computation and power resources. These differences make the proposed method more suitable to real-world uses than PPO. In contrast, PPO assumes fewer restrictions on the system, so PPO can be used in a wider range of systems. PPO can, in principle, deal with convoluted kinematics and sensor configurations that are problematic in the proposed method, as shown in Section 6.2. For these reasons, PPO is more suitable than the proposed method in simulated environments with weaker requirements than those indicated in the problem setting (Section 2.1).

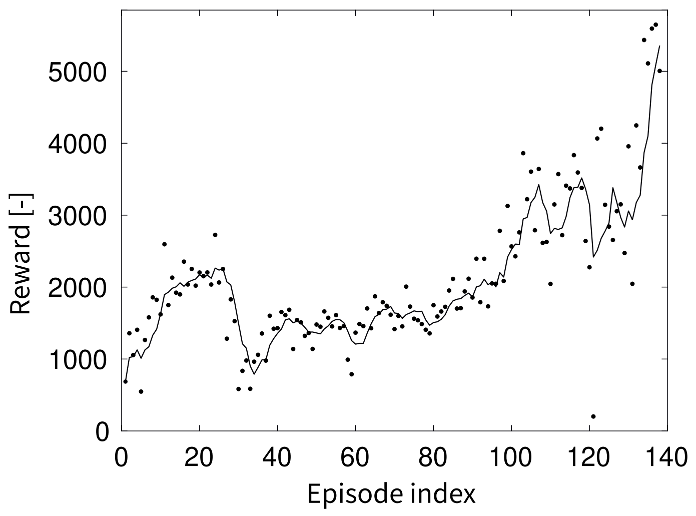


Figure 6.8: Evolution of rewards during training of the PPO agents. Dots: Rewards for each PPO agent. Line: Average reward over the last five trained PPO agents.

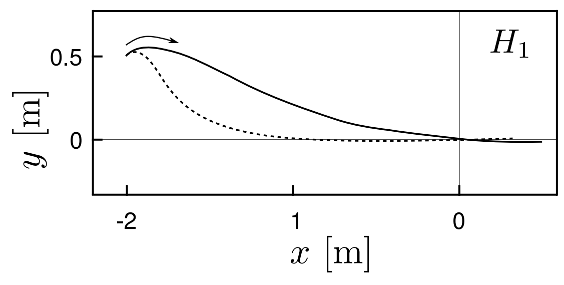


Figure 6.9: Comparison of trajectories between the proposed method (dotted line) and the PPO agent (solid line) for the sensor configuration H_1.

Chapter 7

Experiment

In addition to the simulation of the method in a Matlab implementation, an experiment on a real robot was also realised. In this chapter, the implementation details of the experiment on a real robot are described and the results reported.

7.1 Implementation

The robotic platform used for experimental assessment was the mobile robot Pioneer 3-DX (Figure 7.1), which features two feedback-controlled wheels with a high resolution encoder and a swivel caster for balance [1]. As for the sensors, a 5K PTZ camera was used. This camera features very high resolution images for more accurate sampling of the Jacobians, and may be controlled remotely for quick reconfiguration of the sensor output, which was useful during the development of the experimental software. A pair of colour beacons were installed on the robot to simplify the image processing algorithm that detects the position and orientation of the robot. The beacons were installed on the robot as shown in Figure 7.2.

The software implementation was programmed in C++ and it relies on a set of custom-made executables connected to each other by CORBA[2]. CORBA was chosen over other frameworks such as OpenRTM or ROS due to its easy installation, multi-platform support, flexibility and low number of dependencies with other softwares. CORBA is an ISO standard for the interconnection of computer applications in a network. It is commonly used in high-performance systems and designed by important industry actors around the world. The main characteristic of CORBA is that it automates and simplifies the connection of software modules by defining a common interface between the caller and the callee (Figure 7.3). The interface is described with a C++-like language called IDL, which is translated by the CORBA implementation to the programming language that the module will be implemented on. Here I used the OmniORB[3] open-source implementation of CORBA to implement the modules in C++.

In robotic applications, the control loop may be described as sensor output, signal computations, and control input. This makes for three modules, as depicted in Figure 7.4: The sensor signal module, where the signal from the transducers is transformed and sent over CORBA, the control module, which reads sensor observations and computes the control law, and the actuator module, which transforms the control signal to electrical signals on the physical actuators. Thus, the sensor module and the control module communicate with each other by the following IDL interface:

```
typedef double readings[3];
interface sensor_3D {
    readings sample();
```

[1]https://www.cyberbotics.com/doc/guide/pioneer-3dx
[2]https://www.corba.org
[3]http://omniorb.sourceforge.net

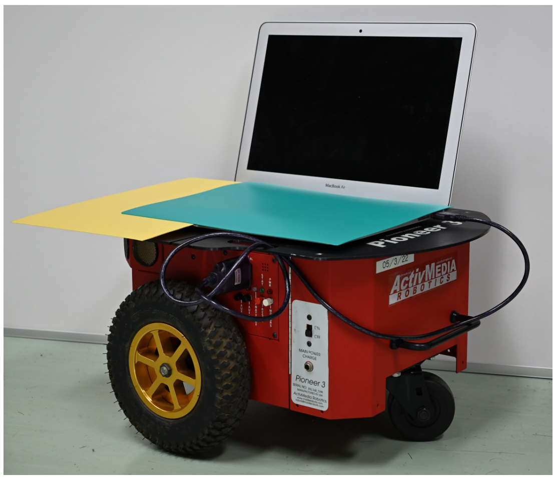


Figure 7.1: The Pioneer 3-DX robot.

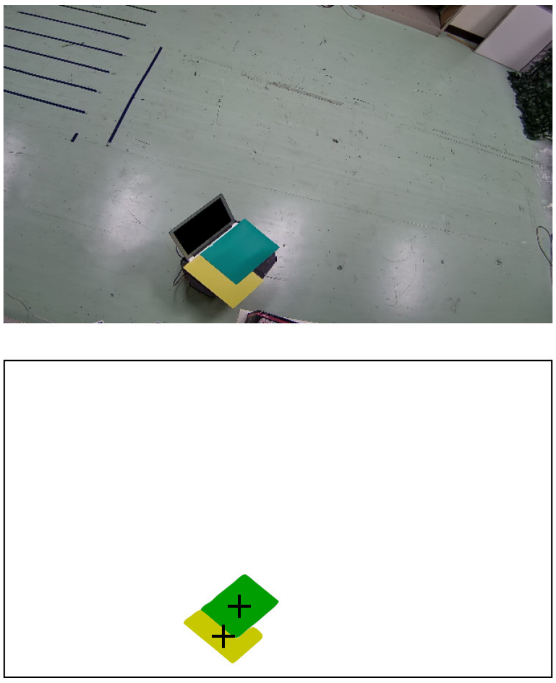


Figure 7.2: Above: Image frame during the experiment, as seen by the camera. Below: Output of the image processing algorithm. The resulting position of the green and yellow beacons are indicated with a cross.

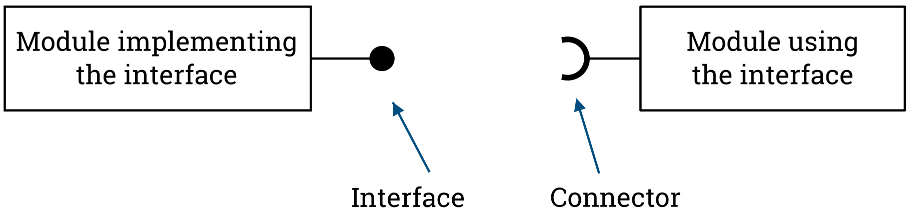


Figure 7.3: CORBA standard denotation for interface implementations (circle) and interface users (arc).

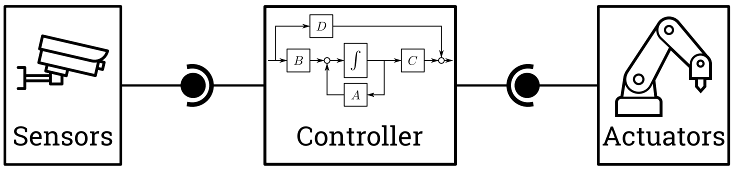


Figure 7.4: Control loop for a system using CORBA between the controller and the input/output devices.

```
};
```

The sensor module provides just enough data for a sensor observation, corresponding to $\boldsymbol{s} = \begin{bmatrix} s_1 & s_2 & s_3 \end{bmatrix}^\mathsf{T}$. On the other side, the interface between the control module and the actuator module follows the IDL definition:

```
typedef double input[2];
interface unicycle {
    void apply_input(in input u, in double duration_secs);
};
```

The data is limited to the input signal $\boldsymbol{u} = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^\mathsf{T}$ and an additional field duration_secs that limits the time that the input signal is applied for security reasons and to simplify the design of the control module. CORBA functions are synchronous, that is, the caller waits until the callee has finished executing the task. This feature was used to force the control module to wait until a sensor observation becomes ready and to wait until the robot has finished executing the control command. The control module is an user of both of the aforementioned interfaces. The sensor and control modules are implementations of the corresponding interface.

Several modules were implemented for each interface, as shown in Figure 7.5, to cover the needs of development, testing and final experiment. Each module is described now.

Sensor monitor

The sensor monitor module is a simple program that samples the sensor values through the sensor_3D interface with a period of one second and prints the values to the console. This module is implemented in file camera.test.cc. Use cases for this module are depicted in Figures 7.6, 7.11 and 7.13.

Camera

The task of the camera module is to grab images from a camera, process the images and provide the coordinates of the robot with an internal, privately specified robot mapping through the sensor_3D interface. Figure 7.6 shows the module connectivity for development and testing of this module, which is implemented in file camera.cc.

The first approach to image processing was to use the image analysis features of OpenCV. The beacons were circular-shaped and the algorithm for detecting the position of the beacons was the Hough transform implemented by OpenCV. The beacon was correctly detected. Unfortunately, this approach was not satisfactory because the deformations of the beacon caused by perspective transformations resulted in the alternating detection of different circles, as shown in Figure 7.7. Consequently, detection of the true position of the beacon was ambiguous. Retrieving the Jacobian under these conditions was impossible because the Jacobian needs precise and highly repetitive measurements of the position, despite the accuracy being not so important.

In a second and final approach, OpenCV was merely used to retrieve the image frames from the camera. The numerics library Armadillo[4] was used to process the image as follows: First,

[4]http://arma.sourceforge.net/

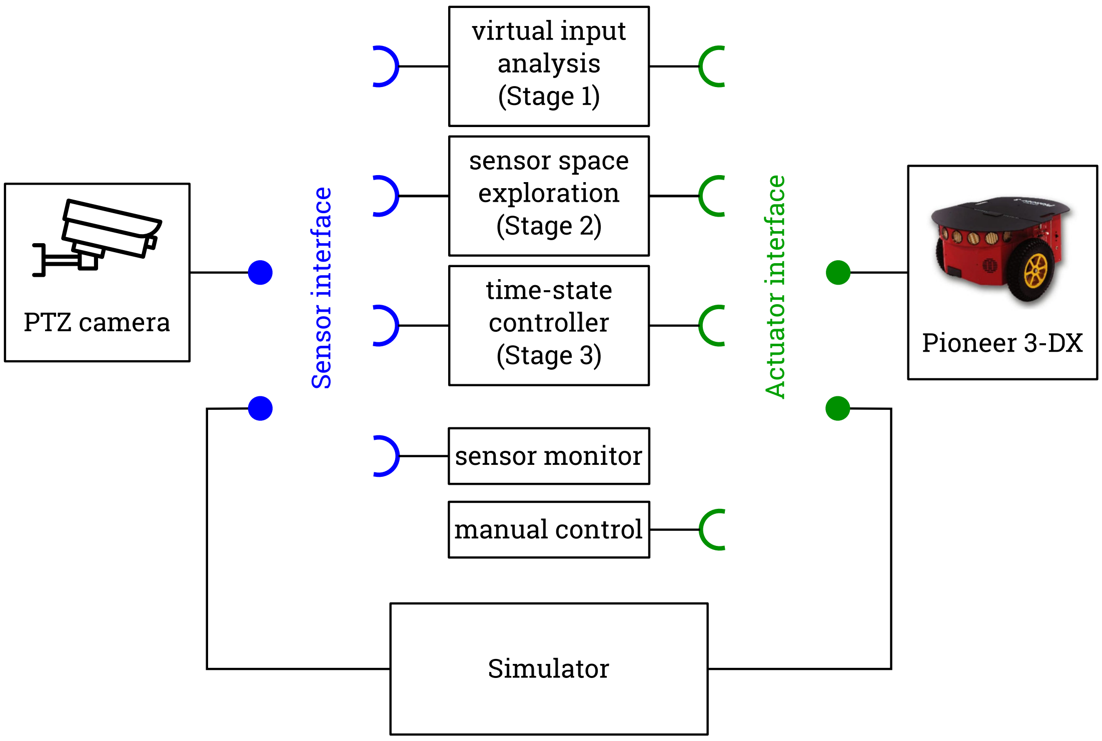

virtual input analysis (Stage 1)
sensor space exploration (Stage 2)
time-state controller (Stage 3)
sensor monitor
manual control
PTZ camera
Sensor interface
Actuator interface
Pioneer 3-DX
Simulator

Figure 7.5: CORBA facilitates mix-and-match capabilities between interface users and interface implementations. This diagram depics all the CORBA modules developed and their relations.

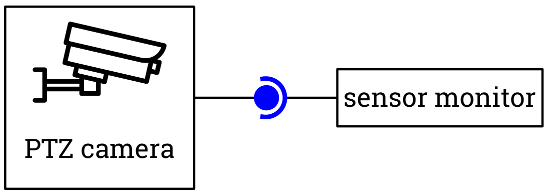


Figure 7.6: CORBA setup for the development of the camera module.

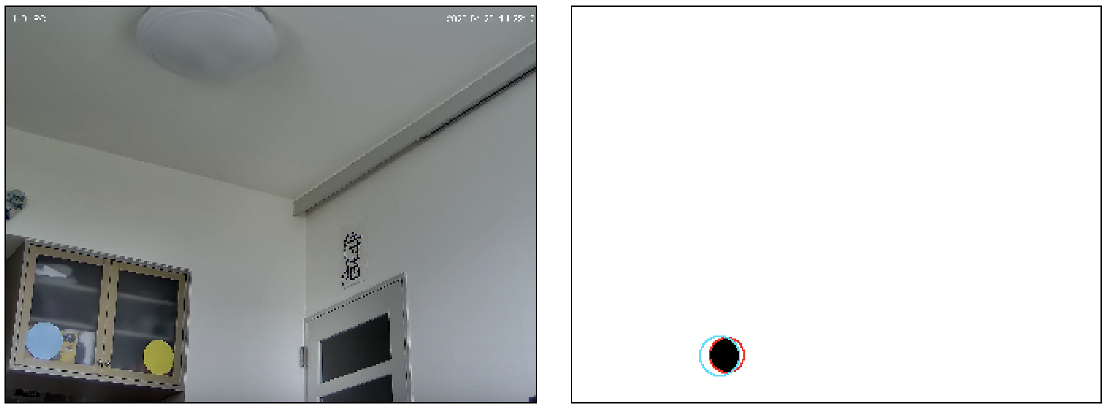


Figure 7.7: Image and processing output of the first, unsuccessful approach to detect the position of the beacons based on the Hough transform.

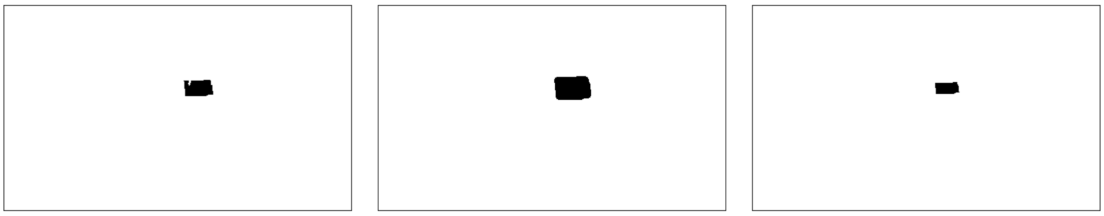


Figure 7.8: Noise removal steps in the image processing algorithm. Left: original segmented image. Center: Image after expanding the area of the black pixels. Right: Image after contracting the region of the black pixels.

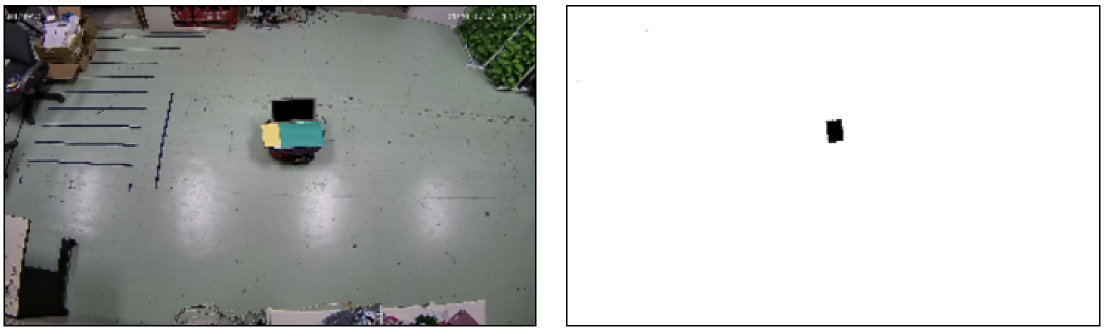


Figure 7.9: Camera frame taken during the experiment (left) and the image processing outcome for the yellow beacon (right).

the image was segmented by discriminating the hue, saturation and value components of each pixel and for each beacon. Then, the noise was cleaned using an expand and then contract filter, as shown in Figure 7.8. Lastly, the centroid of the remaining pixels was calculated. The position of the beacon was given by the centroid.

The coordinates exposed to the CORBA interface sensor_3D were the x and y camera coordinates of the green beacon, and the angle between the centroids of the green and yellow beacons, as shown in Figure 7.9.

Manual control

The manual control module is another simple program that uses the CORBA interface unicycle. This module is implemented in source file unicycle_test.cc. It takes two command line arguments representing the desired linear speed and the desired angular speed. These values are then commanded to the module implementing the interface unicycle with a fixed duration of 1 second. This module is useful for repositioning the robot manually, albeit somewhat cumbersome, and to stop the robot immediately in case of emergency (linear and rotation speeds set to zero). The use cases of this module are depicted in Figures 7.10, 7.11 and 7.13.

Robot

The Pioneer 3-DX robot is compatible with the C++ library *ARIA*, which is the de facto library for all Pioneer robot models. This library has an overwhelming amount of utilities for controlling the Pioneer robots and for developing robot applications, specially for navigation purposes. However, this functionality comes at the price of complexity and third-party library dependencies. The CORBA module that implements the unicycle interface started as a simple wrapper of the ARIA library[5] to the unicycle interface. However, due to difficulties in installation

[5]https://github.com/reedhedges/AriaCoda

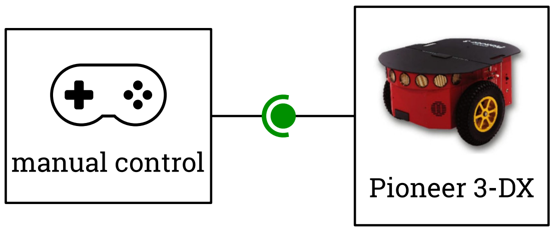


Figure 7.10: CORBA setup for the development of the robot module.

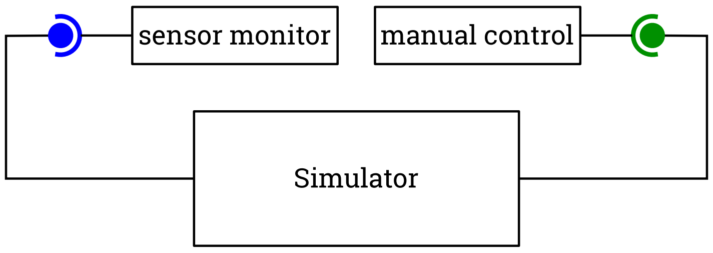


Figure 7.11: CORBA setup for the development of the simulator.

in the onboard computer and the complexity of the library, this approach was ditched in favour of a custom-made implementation of the protocol stack of the Pioneer robot, called AROS[6]. Implementation of the protocol stack resulted easier than setting up the library in the onboard computer. The onboard computer was an Apple MacBook Air with operating system OSX. Many libraries with complex dependences have difficulties with setting up in OSX, and the ARIA library was no exception. By developing the protocol stack on standard C++ and with no external dependencies, it was possible to control the robot from Linux-based computers or OSX-based computers easily and with deterministic behaviors.

The protocol stack is implemented under the directory pioneer_aros and the module that connects it to the CORBA framework is in unicycle.cc. Figure 7.10 depicts the CORBA connection diagram during the development of the Pioneer module.

Simulator

The simulator CORBA module was added to simplify development of the control modules. It consists of two parts: the state equation and the sensor transformation. The simulator applies the input commands on the unicycle CORBA interface to a dynamic equation solver applied on (6.1). The resulting state is then transformed by a given sensor transformation function (6.2) and shared with the controller via the sensor_3D CORBA interface. Thus, the simulator implements the two CORBA interfaces. The simulator is implemented in file simulator.cc. During development, the sensor monitor module and the manual control module were attached to the interfaces for testing, as shown in Figure 7.11.

Controllers

The controllers for learning the virtual input, mapping the sensor space and testing the mapping with a feedback controller were developed in three independent modules against the simulator module, as depicted in Figure 7.12.

The MATLAB code that handles the learning algorithms was ported to C++ using the Armadillo numerics library. The port is almost straightforward and the simulator module was

[6]www.inf.ufrgs.br/~prestes/Courses/Robotics/manual_pioneer.pdf

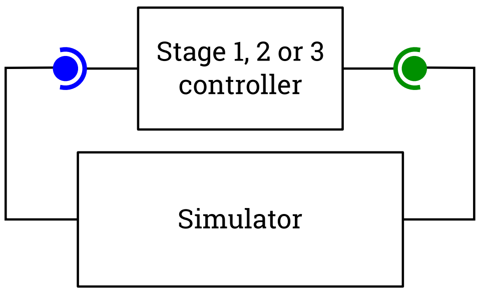


Figure 7.12: CORBA setup for the development of the controllers for each stage: virtual input analysis, sensor space exploration, and feedback controller.

used to assess that the algorithms behaved exactly the same as the Matlab counterpart. Thus, the differences between the Matlab simulation and the experiment on the real robot should be attributed to differences between the simulated robot and the real robot, which is precisely the purpose of realizing a real experiment. The ported code is found in func_approximate.cc and func_approximate.hh.

The three control modules are now described separately:

1. The module implemented in file controller_IOD.cc takes care of finding a virtual input as indicated in Section 5.1. There is no input to the program. After sampling the Jacobians in the neightborhood of the initial state, the program will output the input vector $\boldsymbol{u}_{(s^{(*)})}$ and the input duration $\Delta t_{(s^{(*)})}$ needed by the virtual input u_3.

2. The module implemented in file controller_sensorspace.cc requires the output from the previous module as program arguments. The module then proceeds to sample the sensor space by following a predefined pattern as indicated in Section 5.2. Data from the sensor space and data from the applied inputs are gathered into two datasets and written in files dataset_sensor.m and dataset_linear.m. These files are compatible with Matlab file format for analysis and plotting of the dataset.

3. The last module is implemented in file controller_control.cc. This module reads the mentioned dataset files and computes the function approximations ϕ_1, ϕ_2 and ϕ_3 with the Gaussian RBF method described in Section 4.2. The module then proceeds to operate the robot with a feedback controller with the same pole configuration as in the Matlab simulation $p_1 = p_2 = -5$.

7.2 Results

During the experiment, the CORBA modules were connected as depicted in Figure 7.13 to perform the experiment. The camera module, which implements the sensor_3D interface, was connected to the controller modules sequentially and also to the sensor monitor to ensure that the readings were reliable and responsive. Likewise, the pioneer module, which implemented the unicycle interface, was connected to the controller modules sequentially and also to the manual control module to ensure that the robot could be stopped in case of error in the control modules or some other emergency. The CORBA modules were executed in a distributed environment of three computers: The pioneer module ran on the onboard computer. The camera module ran on its own computer due to the heavy computations required by the image processing algorithm. The rest of the modules were run on the terminal computer from where the experiment was being operated.

The parameters of the controllers were similar to those in the simulation. 6 samples per axis with a separation of 0.3 units in the first stage and 4 samples per axis for the second

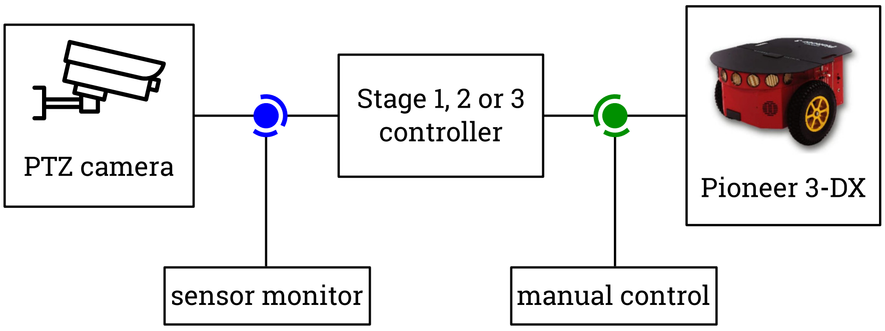


Figure 7.13: CORBA setup for the execution of the experiment of the real robot. The sensor signal was being monitored at all times and the manual control module was ready to override the controllers to ensure safety of the experiment.

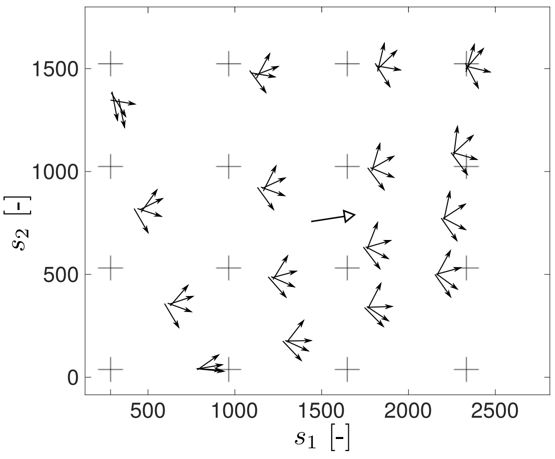


Figure 7.14: Sampled points in camera coordinates for the dataset. The empty-filled arrow indicates coordinates of the initial position for stages one and two. The crosses indicate the location of Gaussian kernel centres. The angle of the arrows indicate the value of s_3 in radians. Note that the camera y-axis is inverted.

stage, thus a reduced number of observations compared to the simulation. The mapping $\boldsymbol{\phi}$ was approximated using 4^3 bases with a standard deviation of 0.45 and regularisation term $\lambda = 0.5$. The feedback controller had a double pole at -5; the same poles as in the simulation.

The unicycle-like robot executed exploration of the sensor space and feedback control to the time axis successfully, as depicted in Figure 7.14. The output of the first stage was $\boldsymbol{u}_{(\star)} = \boldsymbol{u}_{(2)}$ and $\Delta t_{(\star)} = 1.5$ seconds. Four feedback controlled tasks were tested from different starting points as shown in Figure 7.15, all of them converging from the bottom-left hand side of the diagram to the approximated position of the time axis. As in the simulations, no back and forth control was added because the purpose of the feedback controller was just to evaluate the mapping. Some imperfections are noticeable in the sample observations from the sensor space exploration stage. They were attributed to perspective deformation, lens aberrations, signal noise, image processing lag, partial occlusion of beacons and cumulative positional errors.

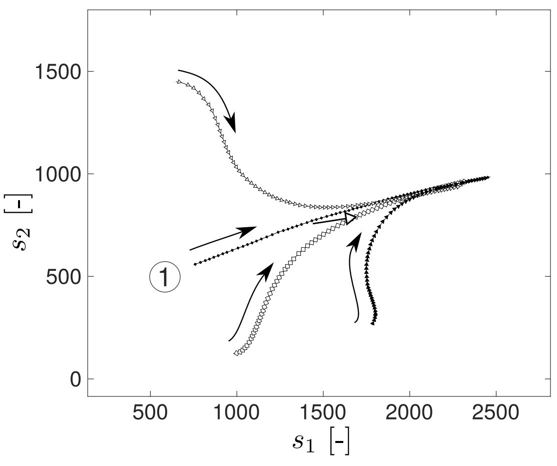


Figure 7.15: The feedback controlled trajectories for the experiment on the real robot. The trajectory marked with ① starts at approximately the time axis and follows it. The other three trajectories converge to the time axis as well. The initial position for stages one and two is indicated with an empty-filled arrow.

Chapter 8

Conclusion

This research has proposed a method to tackle controllability of non-holonomic driftless systems with partially unknown kinematics and unknown sensor configuration (*i.e.* unknown sensorimotor mapping). The problem setting is novel because state of the art researches focus on either the sensorimotor mapping or controllability of non-holonomic systems, but not both. The main contribution is the automatic design of a virtual input to overcome non-holonomic constraints and a fixed-pattern method to explore the sensor space, with the previously obtained virtual input, for enabling controllability of non-holonomic driftless systems with unknown sensorimotor mapping. The result is a method that improves the reach of mobile robotics by facilitating the flexibility in the design and deployment of these robots.

The proposed method has been successfully tested in a simulated environment and in a real experiment. The simulated test was realised on Matlab and the real experiment was realised on a Pioneer3-DX robot with CORBA as middleware platform. Additionally, the Proximal Policy Optimization algorithm was tested in Matlab under similar problem settings for comparing to the proposed method.

8.1 Limitations

There are several limitations that were identified in the proposed research. Firstly, the region of controllability is bound to the explored region in sensor space during the second stage (Section 5.2). This problem can be mitigated by changing the algorithm of function approximation because the used algorithm, which was Gaussian RBF (Section 4.2), is restricted to interpolated observations. Another limitation is that in heavily distorted sensor spaces, such as cameras installed very close to the floor where the effect of the perspective is strong, the accuracy of the deduction of the virtual input (Section 5.1) can result in problems with dead-reckoning tracking in chained form during the second stage (Section 5.2). Thirdly, learning is performed offline, which is not always possible or desirable, because it was not possible to rely on methods that required better knowledge of the dynamic system and the environment, due to the generality of the problem requirements (Section 2.1). Fourth, sampling the sensor space is affected by the problem known as the curse of dimensionality, whereby each additional dimension in the problem space increments exponentially the number of necessary observations. Despite the curse of dimensionality, the proposed method still has better performance than reinforcement learning methods (Section 6.3.3). Lastly, the method requires that there is no minimum turning radius ρ_{min}. In other words, it must be capable of turning without moving. So, car-like vehicles, which have a limitation $\rho > \rho_{min}$, are not supported by the proposed method.

8.2 Future works

Despite this work incrementing the flexibility of installing non-holonomic robots, there is still a lot of room for improving the reach of these robots. It looks like the assumptions made in Section 2.1 can be relaxed further in potential future works. For example, this method does not support non-holonomic driftless systems whose control inputs are collinear at the designated initial state, that is, when $\boldsymbol{j}_1^{(0)} \parallel \boldsymbol{j}_2^{(0)}$ (See Section 2.2 for a description of this notation). To solve this novel problem, an additional stage before acquiring the virtual input (Section 5.1) could explore linear combinations of control inputs $\boldsymbol{u}_1 = \begin{bmatrix} u_1 & 0 \end{bmatrix}^\intercal$ and $\boldsymbol{u}_2 = \begin{bmatrix} u_2 & 0 \end{bmatrix}^\intercal$, such as in $\boldsymbol{u}_\lambda = \lambda_1 \boldsymbol{u}_1 + \lambda_2 \boldsymbol{u}_2$, to find the control inputs that make sensor observations vary as orthogonally as possible ($\boldsymbol{j}_1^{(0)} \perp \boldsymbol{j}_2^{(0)}$).

Another possible future work is the design of an online controller that does not need pre-training and can learn the sensorimotor even during its first initial approach to the desired state $\boldsymbol{s}_{(d)}$. The fundamental idea is that the sensorimotor mapping is approximated at every control cycle with a neural network [43], which can extrapolate approximations unlike kernel-based approximation methods, followed by a control policy given by the continuation method [48] over the best approximation of the sensorimotor mapping at each control step. If the controller approximates correctly the sensorimotor mapping, then it should not be a problem to get gradually closer to $\boldsymbol{s}_{(d)}$, and even faster as it gets closer because with closer observations, the approximation can get more accurate as well. Recently, [13] proposed a particularisation to the continuation method based on parametric functions that can solve other limitations in the method proposed in this thesis, such as restrictions in the minimum radius of rotation of the mobile robot. [13] is a strong candidate for the control law in potential implementations of online controllers.

Bibliography

[1] Khoukhi Amar and Shahab Mohamed. Stabilized Feedback Control of Unicycle Mobile Robots. *Int. J. Adv. Robot. Syst.*, 10(187), apr 2013.

[2] Minoru Asada, Takamaro Tanaka, and Koh Hosoda. Adaptive binocular visual servoing for independently moving target tracking. In *Proc. - IEEE Int. Conf. Robot. Autom.*, volume 3, pages 2076–2081, 2000.

[3] A. Astolfi. Exponential stabilization of a car-like vehicle. In *IEEE Int. Conf. Robot. Autom.*, pages 1391–1396, 1995.

[4] Alexey V. Borisov, Ivan S. Mamaev, and Ivan A. Bizyaev. Historical and critical review of the development of nonholonomic mechanics: the classical period. *Regul. Chaotic Dyn.*, 21(4):455–476, jul 2016.

[5] R.W. Brockett. Asymptotic stability and feedback stabilization. In *Differ. Geom. Control Theory*, pages 181–191. Birkhauser, 1983.

[6] Andrea Censi and Richard M Murray. Bootstrapping bilinear models of Simple Vehicles. *Int. J. Rob. Res.*, 34(8):1087–1113, jul 2015.

[7] HM Choset, S Hutchinson, KM Lynch, and G Kantor. *Principles of robot motion: theory, algorithms, and implementation.* 2005.

[8] B. D'Andrea-Novel, G. Bastin, and G. Campion. Modelling and control of non-holonomic wheeled mobile robots. In *IEEE Int. Conf. Robot. Autom.*, number April, pages 1130–1135, 1991.

[9] Sergio Dominguez, Pascual Campoy, Jose Maria Sebastian, and Agustin Jimenez. *Control en el Espacio de Estado.* Prentice Hall, Madrid, 2006.

[10] Wenjie Dong, Wei Liang Xu, and Wei Huo. Trajectory tracking control of dynamic nonholonomic systems with unknown dynamics. *Int. J. robust nonlinear Control*, (9):905–922, 1999.

[11] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *35th Int. Conf. Mach. Learn. ICML 2018*, volume 4, pages 2587–2601. International Machine Learning Society (IMLS), feb 2018.

[12] Mirosław Galicki. The planning of optimal motions of non-holonomic systems. *Nonlinear Dyn.*, 90(3):2163–2184, nov 2017.

[13] Ida Góral and Krzysztof Tchoń. Lagrangian Jacobian Motion Planning : A Parametric Approach. *J. Intell. Robot. Syst.*, 85:511–522, 2017.

[14] Volker Graefe. Calibration-free robots for cost-effective, dependable and robust automation. *Proc. IEEE Int. Conf. Autom. Logist. ICAL 2008*, pages 262–267, 2008.

[15] Volker Graefe and André Maryniak. The Sensor-control Jacobian as a basis for controlling calibration-free robots. In *IEEE Int. Symp. Ind. Electron.*, volume 2, pages 420–425. IEEE, 1998.

[16] Koh Hosoda and Minoru Asada. Versatile visual servoing without knowledge of true Jacobian. In *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, volume 1, pages 186–193. IEEE, 1994.

[17] Mariusz Janiak, K Tchon, and Krzysztof Tchoń. Constrained motion planning of nonholonomic systems. *Syst. Control Lett.*, 60(8):625–631, 2011.

[18] Zhong Ping Jiang and Henk Nijmeijer. A recursive technique for tracking control of nonholonomic systems in chained form. *IEEE Trans. Automat. Contr.*, 44(2):265–279, 1999.

[19] Yuichi Kobayashi, Kentaro Harada, and Kentaro Takagi. Automatic controller generation based on dependency network of multi-modal sensor variables for musculoskeletal robotic arm. *Rob. Auton. Syst.*, 118:55–65, aug 2019.

[20] Yuichi Kobayashi, Eisuke Kurita, and Manabu Gouko. Integration of Multiple Sensor Spaces With Limited Sensing Range and Redundancy. *Int. J. Robot. Autom.*, 28(1), 2013.

[21] Ilya Kolmanovsky and N Harris Mcclamroch. Developments in nonholonomic control problems. *IEEE Control Syst.*, 15(6):20–36, dec 1995.

[22] Risi Kondor. Regression by linear combination of basis functions, 2004.

[23] F. Lamiraux and J. P. Laumond. Smooth motion planning for car-like vehicles. *IEEE Trans. Robot. Autom.*, 17(4):498–502, 2001.

[24] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Reinforcement learning in continuous action spaces through sequential Monte Carlo methods. *Adv. Neural Inf. Process. Syst. 20 - Proc. 2007 Conf.*, 2007.

[25] Alex X. Lee, Sergey Levine, and Pieter Abbeel. Learning Visual Servoing with Deep Features and Fitted Q-Iteration. In *ICLR*, mar 2017.

[26] E Lefeber, A. Robertsson, and H. Nijmeijer. Linear controllers for exponential tracking of systems in chained form. *Int. J. Robust Nonlinear Control*, 10:243–263, 2000.

[27] Erjen Lefeber, Anders Robertsson, and Henk Nijmeijer. Linear controllers for tracking chained form systems. In *Lect. Notes Control Inf. Sci.*, pages 183–199. 2004.

[28] Gaofeng Li, Shan Xu, Lei Sun, and Jingtai Liu. Kinematic-free position control for a deformable manipulator. In *2016 35th Chinese Control Conf.*, pages 10302–10307. IEEE, jul 2016.

[29] Han Li, Tianding Chen, Hualiang Teng, and Yingtao Jiang. A graph-based reinforcement learning method with converged state exploration and exploitation. *C. - Comput. Model. Eng. Sci.*, 118(2):253–274, 2019.

[30] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Int. Conf. Learn. Represent.* International Conference on Learning Representations, ICLR, sep 2016.

[31] Jihao Luo and Panagiotis Tsiotras. Control design for chained form systems with bounded inputs. *Syst. Control Lett.*, 39(2):123–131, 2000.

[32] Angel Martínez-Tenor, Juan Antonio Fernández-Madrigal, Ana Cruz-Martín, and Javier González-Jiménez. Towards a Common Implementation of Reinforcement Learning for Multiple Robotic Tasks, feb 2017.

[33] Naoyuki Masuda and Toshimitsu Ushio. Control of Nonholonomic Vehicle System Using Hierarchical Deep Reinforcement Learning. In *Int. Symp. Nonlinear Theory Its Appl.*, pages 26–29, 2017.

[34] Tadasuke Matsumoto, Shigeki Nakaura, and Mitsuji Sampei. Adaptive Control for a Class of Driftless System via Time State Control Form Tokyo Institute of Technology. *Control*, 1(3):495–496, 2006.

[35] W. Thomas Miller. Sensor-Based Control of Robotic Manipulators Using a General Learning Algorithm. *IEEE J. Robot. Autom.*, 3(2):157–165, apr 1987.

[36] Jonathan Mugan. Robot Learning: A Sampling of Methods. Technical report, 2005.

[37] Richard M. Murray and S. Shankar Sastry. Steering nonholonomic systems in chained form. In *IEEE Conf. Decis. Control*, volume 2, pages 1121–1126. IEEE, 1991.

[38] David Navarro-Alarcon, Andrea Cherubini, and Xiang Li. On Model Adaptation for Sensorimotor Control of Robots. In *Proc. 38th Chinese Control Conf.*, pages 2548–2552. IEEE, jul 2019.

[39] David Pierce and Benjamin Kuipers. Learning to Explore and Build Maps. In *AAAI*, pages 1264–1271, 1994.

[40] David Pierce and Benjamin J. Kuipers. Map learning with uninterpreted sensors and effectors. *Artif. Intell.*, 92(1-2):169–227, may 1997.

[41] Adam Ratajczak, Krzysztof Tchó, A Ratajczak, · K Tchó, and K Tchó. Parametric and Non-parametric Jacobian Motion Planning for Non-holonomic Robotic Systems. *Springer*, 77(3-4):445–456, 2013.

[42] L. Rifford. Stabilization problem for nonholonomic control systems. In *Geom. Control Nonsmooth Anal.*, pages 260–269. jul 2008.

[43] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. *Parallel Distrib. Process. (Cambridge, MA MIT Press.*, 1986.

[44] Mitsuji Sampei. A Control Strategy for a Class of Non-Holonomic Systems - Time-State Control Form and its Application. In *IEEE Conf. Decis. Control*, pages 1120–1121, 1994.

[45] Mitsuji Sampei, Hiromitsu Kiyota, Masanobu Koga, and Masahiro Suzuki. Necessary and Sufficient Conditions for Transformation of Nonholonomic System into Time-State Control Form. In *Proc. 35rd Conf. Decis. Control*, pages 4745–4746, 1996.

[46] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, jul 2017.

[47] William D Smart and Leslie Pack Kaelbling. Effective reinforcement learning for mobile robots. In *IEEE Int. Conf. Robot. Autom.*, volume 4, pages 3404–3410, 2002.

[48] H J Sussmann. A continuation method for nonholonomic path-finding problems. Technical report, 1993.

[49] Krzysztof Tchon and Joanna Ratajczak. Dynamically consistent Jacobian inverse for nonholonomic robotic systems. *Nonlinear Dyn.*, 85:107–122, 2016.

[50] Hado Van Hasselt and Marco A. Wiering. Reinforcement learning in continuous action spaces. *Proc. 2007 IEEE Symp. Approx. Dyn. Program. Reinf. Learn. ADPRL 2007*, (Adprl):272–279, 2007.

[51] Chaoli Wang, Yingchun Mei, Zhenying Liang, and Qingwei Jia. Dynamic feedback tracking control of non-holonomic mobile robots with unknown camera parameters. *Trans. Inst. Meas. Control*, 32(2):155–169, 2010.

[52] L.E. Weiss and A.C. Sanderson. Dynamic sensor-based control of robots with visual feedback. In *IEEE Work. Intell. Control*, 1985.

[53] Yuxiang Wu and Yu Wang. Asymptotic tracking control of uncertain nonholonomic wheeled mobile robot with actuator saturation and external disturbances. *Neural Comput. Appl.*, 32(12):8735–8745, 2020.

[54] Fang Yang and Chao Li Wang. Adaptive stabilization for uncertain nonholonomic dynamic mobile robots based on visual servoing feedback. *Acta Autom. Sin.*, 37(7):857–864, jul 2011.

[55] Junjie Zeng, Rusheng Ju, Long Qin, Yue Hu, Quanjun Yin, and Cong Hu. Navigation in Unknown Dynamic Environments Based on Deep Reinforcement Learning. *Sensors*, 19(18):3837, sep 2019.

[56] Wei Zeng, Qinghui Wang, Fenglin Liu, and Ying Wang. Learning from adaptive neural network output feedback control of a unicycle-type mobile robot. *ISA Trans.*, 61:337–347, mar 2016.