Learning Algorithms and Systems Laboratory
School of Engineering

Master Thesis

Learning to Control Planar Hitting Motions of a Robotic Arm in a Mini-Golf-like Task

at
École Polytechnique Fédérale de Lausanne
by

Klas Kronander

Lausanne 2010

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

LINKÖPINGS UNIVERSITET
Linköping University
INSTITUTE OF TECHNOLOGY

Learning to Control Planar Hitting Motions of a Robotic Arm in a Mini-Golf-like Task

Master Thesis carried out at the Learning Algorithms and Systems Laboratory
at École Polytechnique Fédérale de Lausanne
by

Klas Kronander

Supervisor: **Seyed Mohammad Khansari Zadeh**
LASA, EPFL

Professor: **Aude Billard**
LASA, EPFL

Lausanne, November 25th, 2010

Abstract

In this thesis we tackle the problem of goal-oriented adaptation of a robot hitting motion. We propose the parameters that must be learned in order to use and adapt a basic hitting motion to play minigolf. Then, two different statistical methods are used to learn these parameters. The two methods are evaluated and compared. To validate the proposed approach, a minigolf control module is developed for a robotic arm. Using the different learning techniques, we show that a robot can learn the non-trivial task of deciding how the ball should be hit for a given position on a minigolf field. The result is a robust minigolf-playing system that outperforms most human players using only a small set of training examples.

Acknowledgments

First of all, I would like to thank my supervisor Mohammad Khansari for all time and effort he has spent to support me during this project. Mohammad has been an invaluable source of knowledge, encouragement and feedback throughout this project, and I truly can not imagine a better supervisor.

I would also like to thank Prof. Aude Billard for giving me the opportunity to do my master thesis at LASA, and for giving invaluable feedback and encouragement during our meetings.

A good portion of the time in this project has been spent developing a software module for the RobotToolKit software package. I have been fortunate enough to have access to Eric, the maker of the RobotToolKit. Without his support and tips, the minigolf module would never have reached its current state.

I direct my gratitude to Martin Duvanel for teaching me to use the stereovision system.

I would like to thank J.B.Keller who built the golf club tool for the WAM as well as the minigolf fields.

Lastly, I would like to thank all the people at LASA for welcoming me to the lab and for being an awesome group of people.

Contents

Chapter 1

Introduction

Traditional approaches to controlling robots rely on hard-coding desired motions in some programming environments. These techniques require an in-depth knowledge about the robot, the programming environment, and the task, is often non-trivial and may become completely non-intuitive when working with high Degrees of Freedom (DoF) robots. Furthermore, the new wave of motivation to move robots from research laboratories into society further accentuate the need to develop more intuitive techniques to control robots even by an unexperienced user. Programming by Demonstration[1] (PbD) [1, 2] is an approach that tackles this problem.

In PbD, the robot is taught to perform a task by observing a set of demonstrations provided by a trained agent (human or robot). Demonstrations to a robot may be performed in different ways; back-driving the robot, teleoperating it using motion sensors, or capturing a task via vision sensors. PbD has higher aims than to simply record a motion and replay it exactly as recorded[2]. Ideally, the robot would be able to capture the important parts of a task, and be able to perform the task in a new situation while respecting these constraints. More specifically, the learning process consists of extracting the relevant information from the demonstrations and encoding this information in a motion model. Using a set of basic motion models learned in this way, more advanced robot motions can be achieved by combining these. These so-called motion primitives [1, 3] can be seen as a basis, from which multiple desired robot tasks can be formed.

The performance of each PbD based approach directly depends on the choice of learning technique and the motion model. In robotic literature, different models and subsequently learning techniques have been proposed, including but not limited to: spline-based methods [4, 5], non-linear time dependent regression techniques [6, 3], time-dependent Dynamical Systems (DS) [7], and non-linear autonomous Dynamical Systems [8, 9]. These methods have been successfully developed to learn motion primitives such as discrete (point-to-point) motions [8, 6],

[1] also referred to apprenticeship learning, learning by imitation, or learning from demonstration

[2] This approach is implemented in industrial robotics. Some robots can have a task showed to them by manually teleoperating the movement of the robot, and then replaying it exactly as recorded.

rhythmic motions [7], hitting motions [10, 11], etc.

The ability to generalize a task and to adapt a learned motion to unseen situations is a desired for robot motion primitives since the number of demonstrations is usually few and limited[3]. One context in which adaptability is crucial is agile sport games such as tennis or minigolf where a succesful player must be able to strike a ball at a variety of speeds and angles. Learning of all these situations by adapting a basic hitting motion is a non-trivial problem, and may not even be possible. Consider a beginning minigolf player. The first skill that the player must learn is how to hit the ball. Once this skill has been learned, the player can easily hit in a different direction and with a differnt speed, using only a slightly different technique. The next skill the player must learn is at what angle and whith what speed she should hit the ball so as to sink[4] the ball. In this thesis we aim at exploring possible techniques to endow a robot with the ability to properly adapt a basic motion in the context of a minigolf task.

To reach this goal, we propose the necessary parameters that can be used to perform such adaptation, and investigate two possible statistical learning techniques to learn these parameters. Though not limited to, in this thesis we test and verify our method on motion primitives modeled with non-linear autonomous Dynamical Systems [11]. We validate our approach in a robotic experiment of playing minigolf with a 7 DoF Barrett WAM [12] robotic arm. A framework allowing simulations as well as real experiments with the WAM arm is set up. This framework is then used to evaluate the methods on different minigolf fields.

1.0.1 The minigolf task and hitting parameters

Minigolf [5] is a game where the players compete in sinking a golf ball in a hole on a terrain with various features and obstacles. The goal is to sink the ball with as few hits as possible [13]. Depending on the features of the terrain, the task of estimating how to hit the ball is a difficult task that for humans takes lots of practice to master. The task considered in this work is sinking the ball in one shot.

Consider the following scenario: a player finds the ball placed somewhere on the minigolf terrain and is faced with the problem of hitting it so that it goes into the hole in one shot. To solve this task, the following questions must be addressed:

1. How should the ball be approached?

2. At what angle and speed should the ball be hit?

The first question concerns the general motion pattern that is used when approaching the ball. Given that a general motion pattern has been learned, the next question is how should this motion be adapted so as to achieve the goal of the task: sinking the ball. In [14], it is shown that human players usually follows the same pattern when approaching the ball, even if the hitting point, hitting

[3] The number of demonstrations should be limited so that to be within the tolerance of a user.
[4] To sink the ball is golf terminology for putting the ball in the hole.
[5] Also commonly referred to as mini-golf, miniature golf, midget golf, crazy golf and Putt-Putt

angle and hitting speeds change. In this thesis we make a more strict distinction between these two questions by assuming that the minigolf tasks consist of two subtasks that can be learned independently of each other:

1. Approach the ball.

2. Use the correct hitting angle and speed.

To solve the second subtask, a human player would probably use a strategy based on the relative position of the ball and hole and the features of the field to get an initial guess of how the ball should be hit. If this guess results in a successful attempt, the player knows what parameters to use for that particular ball position. Once the player has a few successful hitting settings for some different ball positions, she can guess at what angle and speed the ball should be hit in a new situations, i.e. previously untested ball positions . How to learn what hitting parameters to use based on a set of successful examples is the main question of this thesis.

1.0.2 Modeling of the hitting motion

As stated in the previous section, one of the skills needed to play minigolf is to reach the ball. In robotics, this corresponds to having a basic motion model governing the hitting motion. This motion model must of course be chosen such that the hitting angle and speed can be adjusted. In this work, we consider a hitting motion modeled with Dynamical Systems (DS). We use the approach presented in [11], were a point-to-point motion with non-zero velocity at the target can be encoded in a DS while guaranteeing global convergence at the hitting point.

1.1 Objectives

For a recitation of the original project description, refer to appendix A.

The goal of this master thesis is to investigate techniques that ca be used for learning how to adapt a hitting motion to new situations.

Two statistical methods to model the mapping from input space to parameter space are considered and compared against each other. These methods are Gaussian Process Regression(GPR) and Gaussian Mixture Regression(GMR).

The performance of proposed approach are validated in robot experiments of playing minigolf in different challenging fields on the 7 Degrees of Freedom (DoF) Barrett WAM Arm and its simulator are used. To perform the experiments, a minigolf module is developed that can be used to communicate and control both the robot and its simulator.

To summarize, the project consists of two parts:

- Theoretic part: Evaluation of GPR and GMR when applied to this particular problem.

- Implementation part: The conception of the minigolf module for the WAM.

1.2 Outline

In chapter 2, a detailed description of the minigolf task and its challenges is presented. Chapter 2 also gives an overview of the work presented in [8] and [11], which is used for the default hitting motion considered. In chapter 3 the models used to learn the hitting parameters are presented. This is followed by an in depth presentation of the developed software in chapter 4. The different setups that were used to evaluate both the overall system performance and the different models for the hitting parameters are presented in chapter 5. Results from experiments using this setup and a discussion obout these are given in chapter 6 followed by summary in 7.

1.3 Notation

Below, the notation of elements that reappear throughout the text are summarized.

	Symbol	**Dim.**
End effector position	x	$\mathbb{R}^3$
End effector velocity	$\dot{x}$	$\mathbb{R}^3$
End effector position before starting hitting motion	x^0	$\mathbb{R}^3$
Hitting point	x^*	$\mathbb{R}^3$
Default hitting direction	ψ_0	$\mathbb{R}^3$
Hitting angle w.r.t. default hitting direction	θ	$\mathbb{R}$
Hitting speed	v	$\mathbb{R}$
Ball position relative to hole (input)	ξ	$\mathbb{R}^2$
Number of demonstrations of default hitting motion	N	$\mathbb{R}$
Mean (used in various contexts)	μ	various
Covariance matrix (used in various contexts)	Σ	various
Number of hitting parameter training points	M	$\mathbb{R}$

Chapter 2

A closer look at the minigolf task

The minigolf task was briefly introduced in chapter 1. In this chapter the task and its difficulties are formalized in a more detailed manner. Given a minigolf field, the skill that the robot should acquire is to hit a golf ball so that it goes into a hole located at some distance from the robot. Contrary to minigolf as normally played by humans, where the player is allowed to hit the ball several times before sinking it[1], the task considered here is to sink the ball in one shot. As mentioned in section 1.0.1, in this work we consider that this task has to challenges:

1. Approach the ball.
2. Use the correct hitting angle and speed.

Throughout this work, we consider these two tasks as two clearly distinguishable subtasks. The first task is considered given for this work, but brief description of this subtask and how to learn it is included nonetheless, as it is important to the understanding of the adaptation task.

2.1 The hitting motion

A crucial part of minigolf (or any other ball-game) is how to reach the ball. This task is similar to point-to-point motions tasks, which have been extensively studied, see e.g. [6, 15, 16].In this thesis, we use Dynamical Systems (DS) for to model the hitting motion, as proposed by [11]. This modeling has several advantages that make it particularly powerful when compared to alternatives:

- The hitting motion is guaranteed to converge at the hitting point from any point in space.

[1] *Sinking* the ball is golf terminology for putting the ball in the hole, i.e. completing the task successfully.

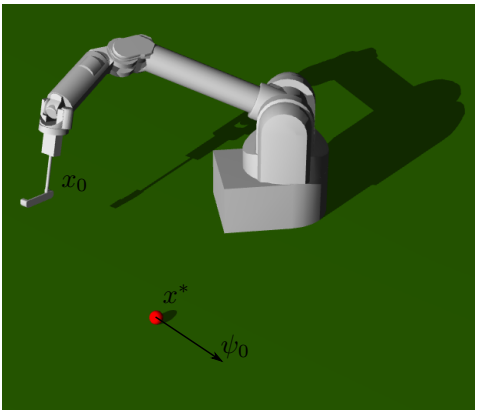


Figure 2.1: The WAM robotic arm with end effector in the rest position x^0 before starting a hitting motion with direction ψ_0 towards the ball at position x^*.

- Inherent robustness to spatial and temporal perturbations [2].
- Modulability, i.e. the hitting speed and angle can be changed.

Figure 2.1 illustrates an example of the minigolf hitting motion. Here the robot starts the motion from an initial point x^0, and approaches the ball at position x^* along the hitting direction ψ_0.

In figure the end effector is in its rest position x^0. This is the starting point of the hitting motion. When the hitting motion is started, the end effector will approach the ball at position x^* along the hitting direction. The hitting motion controls only the trajectory of the end effector. While the orientation of the end effector could be learned similarly as the trajectory, in this thesis we control the orientation of the end effector by assuming that the golf club should be perpendicularly aligned to the hitting direction at all times.

2.1.1 Encoding hitting motions with DS

Consider a set of N demonstrated hitting motions in the form of observed end effector position and velocity at each time instant, $\{x, \dot{x}\}_{t=0,n=1}^{T^n,N}$. We wish to use these demonstrations to model a dynamical system $\dot{x} = f(x)$ so that all trajectories converge to the default hitting direction[3]. Let $\psi_0 = (\psi_x, \psi_y, 0)$ denote this direction, represented as a unit vector. In this work we only consider planar hitting motions, hence the third component in ψ_0 vanishes.

In [8], an algorithm called Stable Estimator of Dynamical Systems (SEDS) is proposed. It is shown how this algorithm can be used to encode demonstrations of

[2] Temporal perturbation causes the robot execution to be delayed (e.g. when slowed down because of friction in the gears) while spatial perturbations causes the robot to depart from its original trajectory (e.g. when slipping or hitting an object).[9]

[3] Here we assume that the N demonstrated hitting motions were demonstrated by approaching the ball from the same direction, thus defining the default hitting direction.

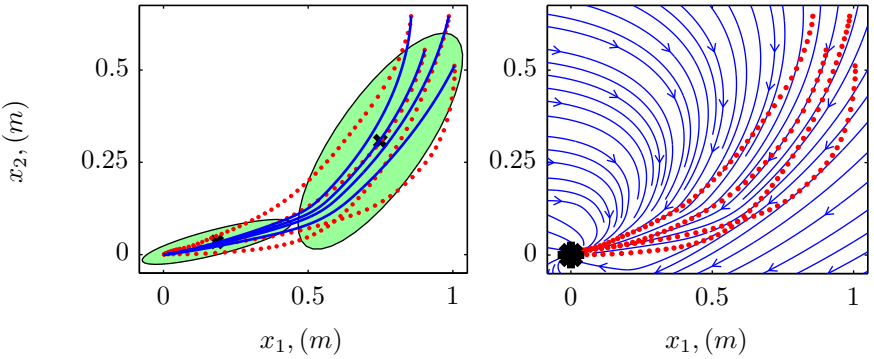


Figure 2.2: Demonstrations and GMM of a hitting motion (left) and resulting DS (right). Four demonstrated trajectories as dotted red lines. Based on these demonstrations, a GMM have been optimized using the SEDS algorithm. The centers of the two Gaussians in the GMM are represented (left) by black crosses, and the covariance is represented by the green elipsoids. The blue lines in (left) show the result of GMR on this model when starting from the same starting points as the demonstrations. The Dynamics of the resulting system is represented by streamlines(right) and plotted along with the demonstrations. Notice how all streamlines lead to the attractor (the ball), regardless of the starting position.

the DS in a Gaussian Mixture Model (GMM) optimized under stability constraints. Then, the trajectory is reproduced through Gaussian Mixture Regression (GMR):

$$\dot{x} = f(x) = \sum_{k=1}^{K} \frac{\pi^k \mathcal{N}(x; \mu_x^k, \Sigma_{xx}^k)}{\sum_{i=1}^{K} \pi^i \mathcal{N}(x; \mu_x^i, \Sigma_{xx}^i)} (\mu_{\dot{x}}^k + \Sigma_{\dot{x}x}^k (\Sigma_x^k)^{-1}(x - \mu_x^k)) \tag{2.1}$$

where π^k,$\mu^k = \begin{pmatrix} \mu_x^k \\ \mu_{\dot{x}}^k \end{pmatrix}$ and $\Sigma^j = \begin{pmatrix} \Sigma_{xx}^j & \Sigma_{x\dot{x}}^k \\ \Sigma_{\dot{x}x}^k & \Sigma_{\dot{x}\dot{x}}^k \end{pmatrix}$ represent the prior, the mean vector and the covariance matrix of component k in the GMM, and

$$\mathcal{N}(x; \mu_x^k, \Sigma_{xx}^k) = p(x|\mu_x^k, \Sigma_{xx}^k) = \frac{1}{\sqrt{(2\pi)^D |\Sigma_{xx}^k|}} e^{(-\frac{1}{2}(x-\mu_x^k)^T (\Sigma_{xx}^k)^{-1}(x-\mu_x^k))} \tag{2.2}$$

is the probability density function of component k in the GMM.

Use of the SEDS algorithm to estimate the parameters in (2.1) renders a dynamical system that enjoys global asymptotic stability at the attractor x^* [8]. Figure 2.2 gives an example of a two dimenstional hitting motion being encoded in a DS with the SEDS algorithm. Note however that this is not suitable for a hitting motion. Using the DS as in (2.1) to hit a ball would fail, since the motion would have zero velocity at the hitting point.

To overcome this problem, [11] proposed a reformulation of the original structure of DS presented above. In this formulation, a DS is modeled as a multiplication of a so called *target field* and *strength factor*. The target field is defined from the DS in (2.1) as:

$$E(x) = \frac{f(x)}{\|f(x)\|} \tag{2.3}$$

This entity preserves the directional information given by the DS in (2.1). Next, let $v(x)$ denote the strength factor of the hitting motion. Like the target field, the strength factor can be modeled with a GMM, this time considering the data set $\{x, \|\dot{x}\|\}_{n=1,t=1}^{T^n,N}$ and optimizing the model parameters with some suitable algorithm, for example Expectation Maximization [17]. The strength factor can then be reacquired for each end effector position according to:

$$v(x) = \sum_{k=1}^{K} \frac{{}_v\pi^k \mathcal{N}(x; {}_v\mu_x^k, {}_v\Sigma_{xx}^k)}{\sum_{i=1}^{K} {}_v\pi^i \mathcal{N}(x; {}_v\mu_x^i, {}_v\Sigma_{xx}^i)} ({}_v\mu_v^k + {}_v\Sigma_{vx}^k ({}_v\Sigma_x^k)^{-1}(x - {}_v\mu_x^k)) \tag{2.4}$$

where ${}_v\pi^k, {}_v\mu^k = \begin{pmatrix} {}_v\mu_x^k \\ {}_v\mu_v^k \end{pmatrix}$ and ${}_v\Sigma^k = \begin{pmatrix} {}_v\Sigma_{xx}^k & {}_v\Sigma_{xv}^k \\ {}_v\Sigma_{vx}^k & {}_v\Sigma_{vv}^k \end{pmatrix}$ represent the prior, the mean vector and the covariance matrix of component k in the GMM.

Now that the target field and strength factor have been defined, we can construct the hitting motion as the target field modulated by the strength factor:

$$\dot{x} = g(x) = v(x)E(x) \tag{2.5}$$

This DS renders hitting motions in the direction defined by the direction approach with a hitting speed as per (2.4) while retaining the stability[4] and robustness properties of the original DS [11].

2.1.2 Changing the hitting speed

In the previous section we defined the default hitting motion. Recall the default hitting speed given by $v(x^*)$. In this section it is shown how to alter the speed of the hitting model in (2.5). Consider first the factor $s^d = \frac{v^d}{v(x^*)}$ by which the default hitting speed must be multiplied to acquire a desired hitting speed $v^d = s^d v(x^*)$. For the continuous version of DS, as presented in (2.5), we can safely go ahead and multiply the whole strength factor with our constant s^d without adventuring stability or altering the trajectory [18]. However, when we move to the time discrete domain, altering the speed in this way will change the trajectory depending on the sampling time [18]. Thus, it is not clear how the strength factor should be altered so as to minimize the perturbation of the trajectory while ensuring that the desired hitting speed is achieved at the hitting point.

A common approach when moving from a continuous representation to a discrete implementation is to assume a sampling frequency that is high enough that in practice, we need not worry about any of the effects introduced by discretizing. This approach is used for implementing the default hitting motion in this work. To alter the hitting speed we have two choices:

[4] A more correct phrase is asymptotic convergence to the hitting point (rather than stability) since the end-effector has non-zero velocity at the target.

1. Modulate the entire hitting motion with a constant.
2. Modulate the entire hitting motion with a monotonic function.

More formally the choice we have is specifying a modulator $s(\|x-x^*\|)$ altering the hitting motion in (2.5) as $\dot{x} = s(\|x-x^*\|)g(x)$. The first approach corresponds to simply setting $s = s^d$ everywhere. The second option corresponds to choosing some function s that gradually applies the modulation to the motion, achieving s^d only at the hitting point. Figure 2.4 shows the effect that these two alternatives have on the speed of the end effector throughout the hitting motion. To get a smooth, natural hitting motion without unnatural jerky movements, s can be chosen such that:

$$s(\|x-x^*\|) \text{ is monotonic for } \|x-x^*\| \in [0, \|x^0-x^*\|] \tag{2.6a}$$

$$s(0) = s^d \tag{2.6b}$$

$$s(\|x^0-x^*\|) = 1 \tag{2.6c}$$

$$s(\|x-x^*\|) \text{ is smooth} \tag{2.6d}$$

For an example of a suitable choice of s, refer to figure 2.3.

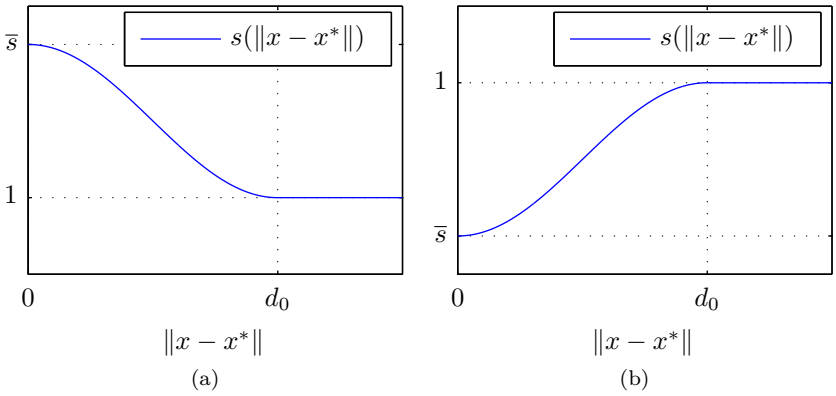


Figure 2.3: Two examples of good modulators. In 2.3a, the desired speed is higher than the default speed. In 2.3b the contrary. The distance d_0 is the distance between the initial position of the end effector, x^0 and the hitting point. These modulators are constructed by letting an appropriately shifted sine gradually increase or decrease during half a period.

The two approaches each have advantages. Modulating the whole motion with a constant has the benefit of exactly preserving the relative speed variations present in the default model. The main advantage of the monotonic function modulator is that it could be used in a system suffering from the effects of discretization . In such a system, changing the speed at an early stage in the motion, when the end effector follows a curved trajectory, could potentially introduce a significant

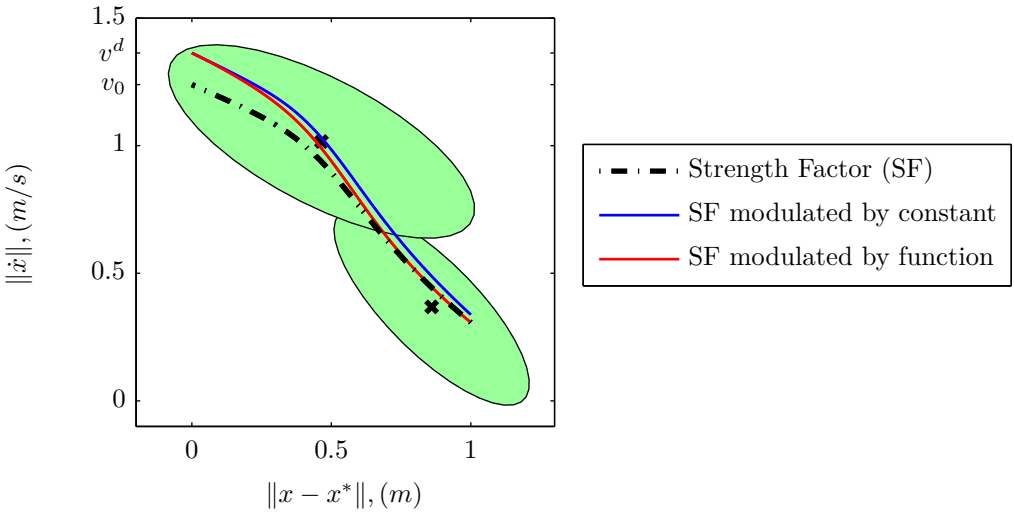


Figure 2.4: The hitting model strength factor, acquired through GMR from the GMM represented by the center (black dots) and covariance matrices(green elipsoids).The dashed line shows the strength factor for the default hitting motion when started 1 meter from the ball. The red and blue lines show the result when changing the hitting speed from the default hitting speed v_0 to the desired hitting speed v^d by multiplying the whole motion with a constant (blue) and multiplying the motion with a smooth monotonic modulator function (red).

deviation from the original target field. Applying the speed change mainly in the end phase of such a hitting motion would be beneficial, since the end effector would then move almost in a straight line. Another advantage of using a monotonic function modulator is that it could be used to adjust the hitting speed pattern of the default model, should this not be satisfactory. Also, for the case where the hitting speed is increased, applying the maximum amplification only at the hitting point is more energy efficient, and may reduce the wear of the robot.

Applying the modulator to the default hitting motion, we can now change the speed by adjusting $\overline{s}$:

$$\dot{x} = s(\|x - x^*\|)g(x) \tag{2.7}$$

2.1.3 Changing the hitting direction

In (2.7) the original DS in (2.5) was altered to have a desired hitting speed. Thus, one of the necessary properties of the hitting model has been acquired. This model can be used according to (2.7) to hit the ball with desired speed in the default direction, ψ_0. In this section we show how to alter the DS in (2.7) to make it possible to hit the ball in any direction, as presented in [11].

Consider a new hitting direction ψ_θ defined by:

$$\psi_\theta = R(\theta)\psi_0 \tag{2.8}$$

with

$$R(\theta) = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.9}$$

Thus, the ψ_θ is rotated counterclockwise around the z-axis by the angle θ. With the new direction formalized like this, the change of (2.7) is merely a rotation of frames of references. First step is to rotate the end effector position clockwise around the $z-axis$ by angle θ so that it aligns with the original frame of reference. Then, the corresponding velocity output from $g(R^T(\theta))$ must be rotated back to the new frame of reference. More formally, the following DS allows us to hit with a desired speed as in (2.7) and in an arbitrary direction defined by the angle θ against the default hitting direction.

$$\dot{x} = s(\|x - x^*\|)R(\theta)g(R^T(\theta)x) \tag{2.10}$$

Note that the scalar-to-scalar mapping s need not be rotated, since it is only dependent on the distance between the end effector and the hitting point.

2.2 Choosing the hitting parameters

In the previous section, it was shown how a DS hitting model can be used to hit in different directions and with different speeds than what was used when training the model. In this section we will concentrate on how to chose these parameters when facing the minigolf task.

2.2.1 Different fields require different parameters

Consider the simplest possible minigolf field: a flat field without obstacles. Such a field depicted in figure 2.5. In this case the choice of hitting angle is trivial - the ball should simply be hit in a straight line towards the hole. Choosing the speed is also a fairly easy task as there is generally a wide range of valid speeds that will sink the ball in combination with a certain angle for this type of field. The vector $\xi \in \mathbb{R}^2$ denotes the relative position of the hole to the ball projected in the xy-plane. This vector completely specifies the *situation* that the player has to adapt to when choosing the hitting parameters. As can be seen in figure 2.5, to play the flat field, the player simply has to align the hitting direction ψ_θ with this vector.

Now consider the more advanced field in figure 2.6. Here, ξ is aligned with ψ_0, which for the flat field would mean that the hitting angle should be zero. Due to the slope of the field, however, hitting the ball along ξ in this case will fail. To compensate for the slope, a hitting angle must be used, rendering a curved trajectory of the ball. Changing ξ means that a new angle and speed must be selected accordingly. Thus, in order to play minigolf on this field, or on the simpler field in figure 2.5, the player needs to be able to estimate the hitting angle, θ and hitting speed,v given the situation on the field, ξ.

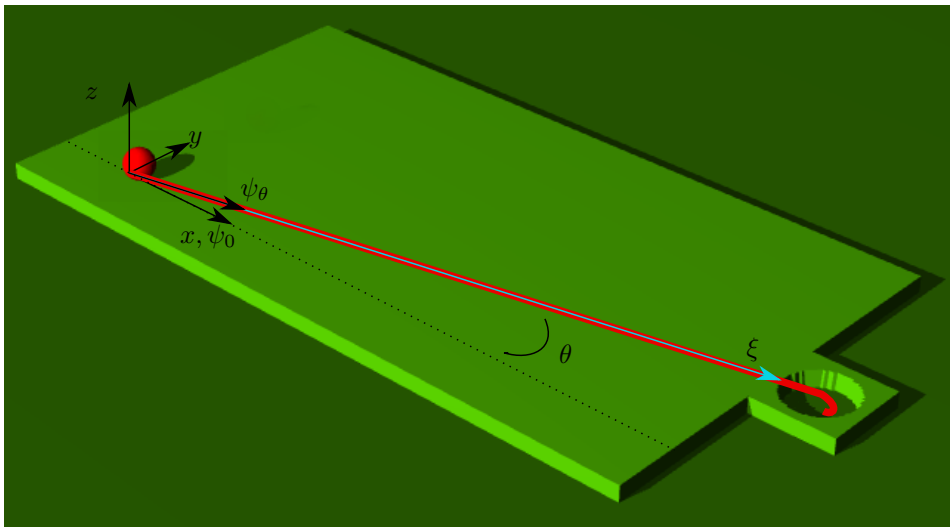


Figure 2.5: Situation on a flat minigolf field. A reference coordinate system is placed on the ball. The 2-dimensional vector ξ (represented by the light blue arrow) describes the situation completely. With the default hitting direction along the x-axis, hitting the ball with angle θ in combination with a range of suitable speeds will result in sinking the ball.

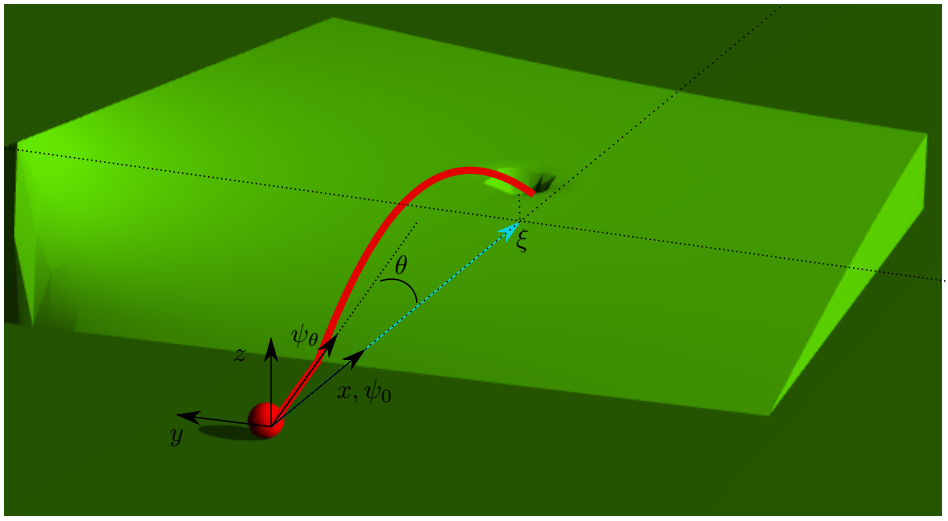


Figure 2.6: Situation on an advanced field. The ball trajectory of a successful attempt is indicated by the red line. Note that the vector describing the situation, ξ is the projection of the vector between the ball and the hole projected in the xy-plane. In contrast to the flat field in figure 2.5, hitting the ball along the ξ will fail.

2.2.2 The function approximation approach

The conclusion from the discussion in the previous section is that to solve the task of correctly estimating the hitting parameters, the player must learn a field-specific nonlinear mapping from the position of the ball to the hitting parameters. More formally, the mapping:

$$h : \xi \in \mathbb{R}^2 \mapsto (\theta, v) \in \mathbb{R}^2 \tag{2.11}$$

must be learned. It should be noted that the problem of choosing hitting angle and speed for the situation in figure 2.6 is generally redundant, meaning that there are several different combinations of (θ, v) that will lead to sinking the ball. Two such possibilities are illustrated in figure 2.7. In order to take the function approximation approach suggested above,we limit the problem to only one of the possible combinations for each input. While this approach will not cover all the different possibilities a player has when faced with the minigolf task, it does supply enough information to solve the task, since it is clearly enough to learn only one way to hit the ball for each situation.

Moreover, concentrating on only one solution at each input has the benefit of dramatically reducing the complexity of the problem. To see why, consider the possible solutions to the situation in figure 2.6. The redundancy of the problem can be interpreted as different strategies in the choice of parameters. One strategy is to hit the ball such that it has the lowest speed possible when reaching the hole. Another strategy is to try to hit the ball as straight as possible, which would imply a higher hitting speed. Similarly, all different solutions has a corresponding strategy. Consequently, to find all possible solutions, one would have to learn one mapping for each strategy.

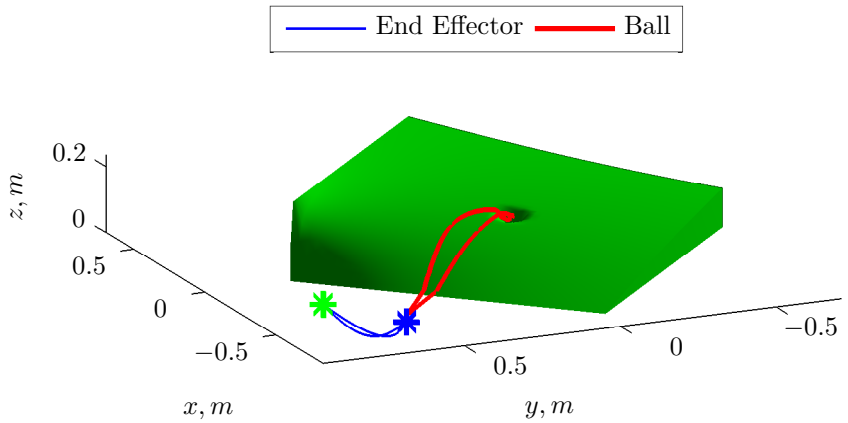


Figure 2.7: The figure illustrates a typical situation for advanced fields: for a given relative position between the ball and the hole, there are several combinations of hitting speed and hitting angle that will lead to sinking the ball. In this work, we call each such combination a strategy. Two different strategies are applied in this figure, one with a high hitting speed and a less curved trajectory, and one where compensating for the fields slope by launching the ball at a bigger angle allows for lower hitting speed.

Chapter 3

Learning the hitting parameters

The conclusion from the previous chapter was that in order to solve the problem of using the correct hitting angle and hitting speed, we must learn at least one mapping $(\theta, v) = h(\xi)$ (refer to section 2.2.2). For the rest of this thesis the situation on the field ξ, which is the input to the learning problem, will be referred to as simply the *input*. In this chapter, the different methods used to learn $(\theta, v) = h(\xi)$ are presented and discussed. We do not go into more detail about learning the default hitting motion, as this is considered given for this work. For a detailed description on how to learn the default hitting motion used for this work, refer to [11].

3.1 Learning approaches

Estimation of the mapping h can be performed in various ways. One approach is to consider at a set of good examples for which one knows the successful combinations of hitting parameters, and then to guess the parameters for a new situation based on these examples. This is commonly referred to as supervised learning, and has the obvious disadvantage of being completely dependent on a human assistant providing good examples for training. On the other hand this approach is attractive in that, as will be presented in chapter 6, a useful model can be built based only a small set of training points. Moreover, robust and computationally efficient learning algorithms exist for solving this type of problems.

Another alternative is to take a reinforcement learning approach and let the robot explore hitting combinations, take feedback (e.g. observing the path of the ball with computer vision) and evaluate each explored combination of (ξ, θ, v). Based on the evaluation of explored parameters, the robot should then use these evaluations to find a good parameter setting for next attempt. One way to do this is through Cost-Regularized Kernel Regression, proposed in [19]. While this approach is attractive in that it allows the robot to learn the task autonomously, it

has the major drawback that convergence to a good solution is slow. Similarly to the first approach, the technique presented in [19] needs good examples to produce a useful model. However, since these good examples must be found by exploring parameter space, a vast number of attempts is needed before a good model can be built. This is not feasible for implementation on a real robot unless the setup of the environment(i.e. moving the ball and hole etc) is automated.

In this work, we take the supervised learning approach. There are two major reasons for this choice:

- As outlined in section 2.2.2, we have drastically reduced the complexity of the learning problem by only considering only one combination of hitting parameters for each input, i.e. concentrating on learning one strategy. In supervised learning, the teacher is responsible for providing the robot with training examples from the same strategy. Taking the reinforcement learning approach we either have to consider all strategies, or incorporate a mechanism to classify them. Either of these alternatives considerably increases the problem complexity.

- By choosing the supervised learning approach, we hope not only to find good estimates of h, but we also believe that taking this approach can give specific information about the minigolf task and what is important to learn it (e.g. if it is useful or even necessary to consider the hitting parameters as dependent variables).

3.2 Training data

As mentioned in section 2.2.2, the problem of estimating the hitting parameters based on the situation on the field is a redundant problem. There are several different strategies a player can choose between when deciding how to hit the ball. Note that within each strategy, there is a range of different angles and speeds that leads to sinking the ball. It is important to distinguish this redundancy which is simply due to the fact that the hole is larger than the ball, and the redundancy due to completely different possibilities to hit the ball such that it reaches the hole.

Each different strategy corresponds to its own mapping from input space to the hitting parameters. Thus, to learn all different strategies, one training set per strategy is needed. While in this work only one strategy is learned, it is important to note that any attempt at predicting hitting parameters for a new input is bound to fail if the training examples are samples of different strategies. Hence, care must be taken to ensure that the same strategy is used for all the training examples.

Consider a set of M observations of good examples[1] $\{\xi^m, \theta^m, v^m\}_{m=1}^M$. Following the assumption that we are looking for a function $(\theta, v) = h(\xi)$, we assume

[1] Note that these examples are *not* the same as the demonstrations of the default hitting motion.

that the training set consists of noisy[2] observations of this function:

$$\{\xi^m, \theta^m, v^m\}_{m=1}^M = \{\xi^m, h_\theta(\xi^m) + \epsilon_\theta, h_v(\xi^m) + \epsilon_v\}_{m=1}^M \tag{3.1}$$

with noise ϵ_θ and ϵ_v corrupting the angle and speed part of respectively.

For notational clarity, we introduce the following notation used specifically for the training data:

$$\{\Xi, \Theta, V\} = \{\xi^m, h_\theta(\xi^m) + \epsilon_\theta, h_v(\xi^m) + \epsilon_v\}_{m=1}^M \tag{3.2}$$

with

$$\Xi = \begin{pmatrix} \xi^1 \\ \vdots \\ \xi^M \end{pmatrix}, \Theta = \begin{pmatrix} h_\theta(\xi^1) + \epsilon_\theta^1 \\ \vdots \\ h_\theta(\xi^M) + \epsilon_\theta \end{pmatrix} \text{ and } V = \begin{pmatrix} h_v(\xi^1) + \epsilon_v^1 \\ \vdots \\ h_v(\xi^M) + \epsilon_v \end{pmatrix} \tag{3.3}$$

3.3 Statistical learning of the hitting parameters

In this work, two different statistical methods are used to learn from good examples $\{\Xi, \Theta, V\}$ to predict parameters for unseen inputs ξ^*. The first one, Gaussian Process Regression, estimates the values $h_\theta(\xi^*)$ and $h_v(\xi^*)$ independently of each other. The second, Gaussian Mixture Regression, is capable of estimating the underlying model both with and without considering dependency between the hitting parameters in the predictive model.

3.3.1 Gaussian Process Regression

Consider first the definition of a Gaussian Process (GP)[20]:

Definition 3.1 (Gaussian Process) *A Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

When using GP:s for function approximation, the function values are considered random values indexed by the corresponding input values. More specifically, we associate to each of the functions h_θ and h_v a distribution over functions:

$$\begin{aligned} h_\theta(\xi) &\sim \mathcal{GP}(m_\theta(\xi), k_\theta(\xi, \xi')) \\ h_v(\xi) &\sim \mathcal{GP}(m_v(\xi), k_v(\xi, \xi')) \end{aligned} \tag{3.4}$$

Where $m(\xi)$ and $k(\xi, \xi')$ are the mean and covariance functions fully specifying the GP. In this work, we consider only the case where $m(\xi) = 0$[3]. Associating the function with a distribution over functions implies that the function itself is drawn from that distribution. This is equivalent to considering the function as a particular realization of the GP to which it is associated. Depending on the covariance function, we will see that the amount of possible functions drops as we condition the distribution over functions on the training set.

[2]The noise on the observations represents the small redundancies caused by the hole being larger than the ball.

[3]This may be perceived as very limiting - in fact it is not. It is standard practice when using GP:s for regression. See e.g. [20]

Distribution over functions

The observations in (3.1) are noisy examples of the respective functions, so they have a joint Gaussian distribution according to definition 3.1. We write the distribution for the vector containing the noisy observations of h_θ from the training set:

$$\begin{pmatrix} h_\theta(\xi^1) \\ h_\theta(\xi^1) \\ \vdots \\ h_\theta(\xi^N) \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{K}_\theta(\Xi,\Xi) + \sigma_n^2\mathbf{I}\right) \tag{3.5}$$

with

$$\mathbf{K}_\theta(\Xi,\Xi) = \begin{pmatrix} k_\theta(\xi^1,\xi^1) & k_\theta(\xi^1,\xi^2) & \dots & k_\theta(\xi^1,\xi^N) \\ k_\theta(\xi^2,\xi^1) & k_\theta(\xi^2,\xi^2) & \dots & k_\theta(\xi^2,\xi^N) \\ \vdots & \vdots & \ddots & \vdots \\ k_\theta(\xi^N,\xi^1) & k_\theta(\xi^N,\xi^2) & \dots & k_\theta(\xi^N,\xi^N) \end{pmatrix}$$

and where the noise ϵ_θ in (3.1) are considered samples from the distribution $\mathcal{N}(0,\sigma_n^2)$. Similarly, since we can write the distribution for the training set and one test point $h_\theta(\xi^*)$ as:

$$\begin{pmatrix} h_\theta(\xi^1) \\ \vdots \\ h_\theta(\xi^N) \\ h_\theta(\xi^*) \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} \mathbf{K}_\theta(\Xi,\Xi) + \sigma_n^2\mathbf{I} & \mathbf{K}_\theta(\Xi,\xi^*) \\ \mathbf{K}_\theta(\xi^*,\Xi) & \mathbf{K}_\theta(\xi^*,\xi^*) \end{pmatrix}\right) \tag{3.6}$$

where

$$\mathbf{K}_\theta(\Xi,\xi^*) = \begin{pmatrix} k_\theta(\xi^1,\xi^*) \\ k_\theta(\xi^2,\xi^* \\ \vdots \\ k_\theta(\xi^N,\xi^*) \end{pmatrix} \text{ and } \mathbf{K}_\theta(\Xi,\xi^*) = (k_\theta(\xi^*,\xi^1), k_\theta(\xi^*,\xi^2), \dots, k_\theta(\xi^*,\xi^N))$$

We now condition $h_\theta(\xi^*)$ on the training data and obtain:

$$h_\theta(\xi^*)|\Xi,\Theta \sim \mathcal{N}(\hat{h}_\theta(\xi^*), \Sigma_\theta^*) \tag{3.7a}$$

with estimate

$$\hat{h}_\theta(\xi^*) = \mathbf{K}_\theta(\xi^*,\Xi)(\mathbf{K}_\theta(\Xi,\Xi) + \sigma_n\mathbf{I})^{-1}\Theta \tag{3.7b}$$

and predictive variance

$$\Sigma_\theta^* = \mathbf{K}_\theta(\xi^*,\xi^*) - \mathbf{K}_\theta(\xi^*,\Xi)(\mathbf{K}_\theta(\Xi,\Xi))^{-1}\mathbf{K}_\theta(\Xi,\xi^*) \tag{3.7c}$$

Note that the same holds true for h_v of course, the only difference is that the θ and Θ above should be replaced by v and V respectively.

Choosing covariance function

As in all Bayesian estimation, a certain prior belief or knowledge [21] about the data generating process is assumed in GPR. One such prior is the very choice of using GPR - by choosing to do so we have already made assumptions about the data[4]. Another prior is the choice of covariance function. As is clear from equations (3.7a)-(3.7c), the covariance function k_θ has a major impact on the behavior of the function.

The most commonly used covariance function is the Squared Exponential (SE) covariance function and variants thereof. In this work, a variant of the classic SE covariance function that allows encoding of different degrees of correlation of function values for the different dimension in input space is used. When optimizing the parameters of such a covariance function, an Automatic Relevance Determination (ARD) takes place. This means that if changes in one of the input dimension has little or no effect on the corresponding observations, then the same should hold true for the predicted values. This covariance function will be referred to as the ARD covariance function.

Definition 3.2 (ARD covariance function) *The Automatic Relevance Determination covariance function for two-dimensional input is defined as:*

$$k(\xi, \xi') = \sigma e^{-(\xi-\xi')^T M(\xi-\xi')}$$

where

$$M = \begin{pmatrix} l_1 & 0 \\ 0 & l_2 \end{pmatrix}$$

The parameters l_1 and l_2 are the lenghtscales (or kernel width) of the covariance function. The parameter σ is the signal variance.

The infinite differentiability of the ARD covariance implies mean square infinite differentiability of the GP, so any function drawn from this distribution over functions will have smooth characteristics.

The covariance function and its importance to (3.7a) is best illustrated by a simple example:

Example 3.1: GPR

For the purpose of simplicity in this example, we consider only one training point ξ with noise-free observation $\theta = h_\theta(\xi)$ and one test point ξ^* for which we want to find $\hat{h}_\theta(\xi^*)$. From (3.7b) we have:

$$\hat{h}_\theta(\xi^*) = k(\xi^*, \xi)k^{-1}(\xi, \xi)\theta$$

Defining $(\Delta\xi_1, \Delta\xi_2)^T = \xi - \xi^*$ we get for the covariance functions involved:

$$k(\xi^*, \xi) = \sigma e^{-(\Delta\xi_1^2 l_1 + \Delta\xi_2^2 l_2)} \quad , \quad k^{-1}(\xi, \xi) = 1$$

[4] According to (3.4) we have assumed that the function values have a joint Gaussian distribution.

we insert these into the expression for the estimate $\hat{h}_\theta(\xi^*)$:

$$\hat{h}_\theta(\xi^*) = \sigma e^{-(\Delta\xi_1^2 l_1 + \Delta\xi_2^2 l_2)}\theta$$

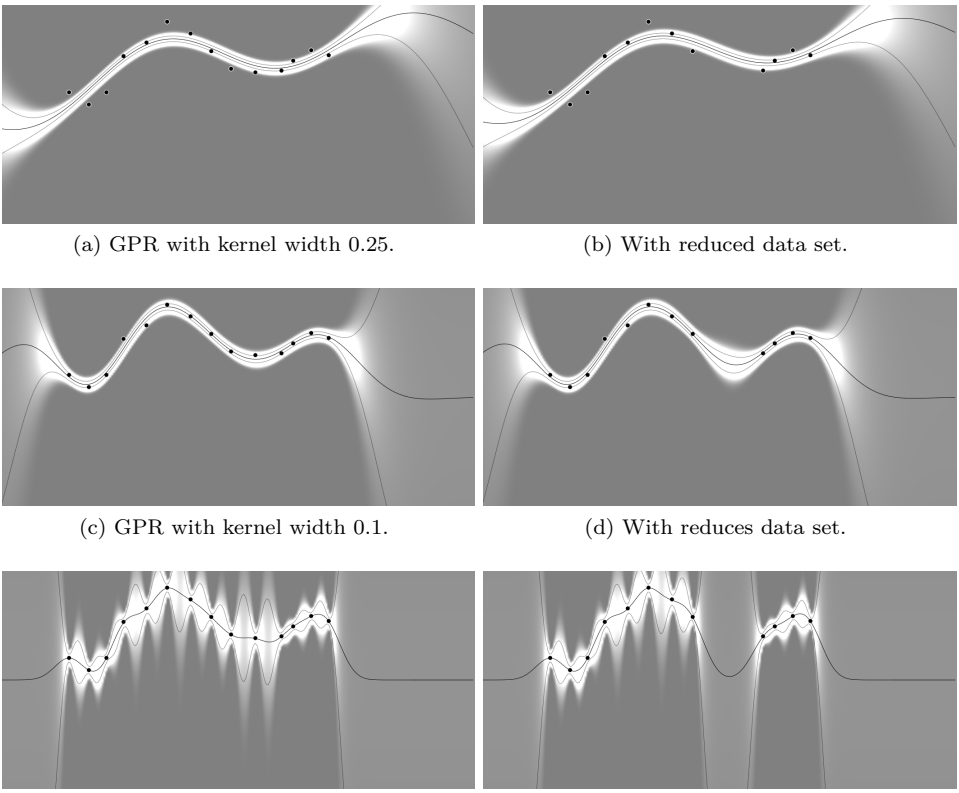
It is now easy to see one of the central assumptions when using this type of covariance function: that points close to each other in input space should have covariant function values. The speed at which the covariance between the test point and the training point decays is controlled with l_1 and l_2. It is clear from the final equation that for a distant test point, the estimate will be very close to zero. This effect is also illustrated in figure 3.1.

In figure 3.1 an example of a two dimensional dataset learned with GPR is illustrated. Note how the short lengthscale in figure 3.1e gives excellent reproduction at the test points, while it is very sensitive to the reduction of the data set, as is seen in figure 3.1f. Note also how the shorter lengthscales in figures 3.1c and 3.1e results in a quick drop of the prediction to zero once we move outside the training range. This effect is not as drastic in 3.1b thanks to the longer lengthscale. However, this comes at the cost of being less able to account for local variations, as is clear when comparing figures 3.1a, 3.1c and 3.1e.

Optimizing the hyperparameters

In the definition in 3.2 there are three free parameters, σ_θ, l_1 and l_2. These parameters, along with the observation noise variance σ_n make out the hyperparameters of the model. As discussed in the previous section, changing these parameters can have drastic effects on the prediction. The question is thus how to chose the hyperparameters. Since we have assumed a Gaussian distribution over functions, with observations corrupted by Gaussian noise, we are fortunate enough to be able to explicitly calculate the log marginal likelihood of the training data [20]. This allows us to use Bayesian model selection, i.e. we select the hyperparameters so as to maximize the marginal likelihood of the training data. To this end, we use a conjugate-gradient based search algorithm available in [22].

Experiments with the small data sets used in this work indicate that the marginal likelihood suffers from local optima. In practice, the algorithm is depending on the user to supply it with a good prior guess as to what the hyperparameters should be.



(a) GPR with kernel width 0.25.

(b) With reduced data set.

(c) GPR with kernel width 0.1.

(d) With reduces data set.

(e) GPR with kernel width 0.05

(f) With reduced data set

Figure 3.1: Estimating a nonlinear data set (represented by black dots) with GPR. The center line indicate the mean of the predictive distribution (i.e. the regression curve) and the two enveloping lines indicate the confidence range of one standard deviation (corresponding to $\sim 66\%$). The predictive confidence is also represented by the background gray scale, where white indicates high confidence. Figures (a), (c) and (e) show GPR with kernel widths 0.25, 0.1 and 0.03 respectively, and (b), (d) and (f) show how the model changes when the data set is altered by creating a gap in input space. The figures were generated with MLdemos [23].

3.3.2 Gaussian Mixture Regression

Gaussian Mixture Regression (GMR) is a regression technique that uses a probability density modeled as a Gaussian Mixture Model (GMM) to make predictions at query points. In this section we describe how a GMM can be used to encode a statistical model of training data, and how estimations can be found using GMR on this model.

Gaussian Mixture Models

To estimate the probability density of the training data, a convex combination of multivariate Gaussian probability density functions is used:

Definition 3.3 *The probability of a point $\gamma \in \mathbb{R}^D$ belonging to the GMM is given by:*

$$p(\gamma) = \sum_{k=1}^{K} \pi^k \mathcal{N}(\gamma; \mu_k, \Sigma_k)$$

where

$$\mathcal{N}(\gamma; \mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^D |\Sigma_k|}} e^{-\frac{1}{2}\left((\gamma-\mu_k)^T \Sigma_k^{-1} (\gamma-\mu_k)\right)}$$

and

$$\sum_{k=1}^{K} \pi_k = 1$$

The GMM is parametrized by $K + DK + \frac{D(D+1)}{2}K$ scalar values corresponding to the priors, means and covariance[5] matrices of the K Gaussians in the model. Given the number of Gaussians, or *states* in the model, the parameters can be optimized to maximize the likelihood of the training set. A popular choice of algorithm to optimize the parameters is the Expectation Maximization (EM) algorithm [17]. EM is a local search technique that guarantees monotone increase of the likelihood of the training data [6]. The EM algorithm needs an initial guess of the parameters. To get a good starting guess, one option is to cluster the training data using the k-means technique. All GMMs in this work[6] are optimized by EM initialized by k-means.

Conditioning the GMM

Setting $\gamma = (\gamma^{\mathcal{I}}, \gamma^{\mathcal{O}})^T$, where $\mathcal{I}$ and $\mathcal{O}$ signifies input and output, we can write, for one of the Gaussian components in definition 3.3 [15]:

$$p(\gamma|k) = \mathcal{N}(\gamma; \mu_k, \Sigma_k) = \mathcal{N}\left(\begin{pmatrix} \gamma^{\mathcal{I}} \\ \gamma^{\mathcal{O}} \end{pmatrix}; \begin{pmatrix} \mu_k^{\mathcal{I}} \\ \mu_k^{\mathcal{O}} \end{pmatrix}, \begin{pmatrix} \Sigma_k^{\mathcal{I}} & \Sigma_k^{\mathcal{I}\mathcal{O}} \\ \Sigma_k^{\mathcal{O}\mathcal{I}} & \Sigma_k^{\mathcal{O}} \end{pmatrix}\right) \tag{3.8}$$

[5] Covariance matrices are symmetric, hence the number of scalar parameters $\frac{D(D+1)}{2}$ as opposed to D^2K as the size of the matrix would otherwise imply.

[6] Note that the GMMs defining the trajectory planning for the default hitting motion are considered given and are not part of this work. These GMMs are optimized differently, as described in [8] and [11].

Now we condition the output on the input:

$$p(\gamma^{\mathcal{O}}|\gamma^{\mathcal{I}},k)=\mathcal{N}\left(\gamma^{\mathcal{O}}|\gamma^{\mathcal{I}};\mu_k^{\mathcal{O}|\mathcal{I}},\Sigma_k^{\mathcal{O}|\mathcal{I}}\right) \tag{3.9a}$$

with

$$\mu_k^{\mathcal{O}|\mathcal{I}}=\mu_k^{\mathcal{O}}+\Sigma_k^{\mathcal{O}\mathcal{I}}(\Sigma_k^{\mathcal{I}})^{-1}\Sigma_k^{\mathcal{I}\mathcal{O}}(\gamma^{\mathcal{I}}-\mu^{\mathcal{I}}) \tag{3.9b}$$

and

$$\Sigma_k^{\mathcal{O}|\mathcal{I}}=\Sigma_k^{\mathcal{O}}-\Sigma_k^{\mathcal{O}\mathcal{I}}(\Sigma_k^{\mathcal{I}})^{-1}\Sigma_k^{\mathcal{I}\mathcal{O}} \tag{3.9c}$$

We now consider the full GMM with all its components, and estimate the probability of the output conditioned on the input by marginalizing out the association between the data point and the Gaussians:

$$p(\gamma^{\mathcal{O}}|\gamma^{\mathcal{I}})=\sum_{k=1}^{K}p(k)p(\gamma^{\mathcal{O}}|\gamma^{\mathcal{I}},k) \tag{3.10}$$

with

$$p(k)=\frac{\pi_k p(\gamma^{\mathcal{O}}|\gamma^{\mathcal{I}},k)}{\sum_{i=1}^{K}\pi_i p(\gamma^{\mathcal{O}}|\gamma^{\mathcal{I}},i)} \tag{3.11}$$

The probability in (3.10) is a linear combination of K Gaussians as in (3.9a)-(3.9c), which means that it is a Gaussian itself [24], with mean and covariance according to:

$$p(\gamma^{\mathcal{O}}|\gamma^{\mathcal{I}})=\mathcal{N}(\gamma^{\mathcal{O}}|\gamma^{\mathcal{I}};\mu^{\mathcal{O}|\mathcal{I}},\Sigma^{\mathcal{O}|\mathcal{I}}) \tag{3.12a}$$

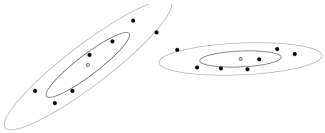
$$\mu^{\mathcal{O}|\mathcal{I}}=p(k)\mu_k^{\mathcal{O}|\mathcal{I}} \tag{3.12b}$$

$$\Sigma^{\mathcal{O}|\mathcal{I}}=p^2(k)\Sigma_k^{\mathcal{O}|\mathcal{I}} \tag{3.12c}$$

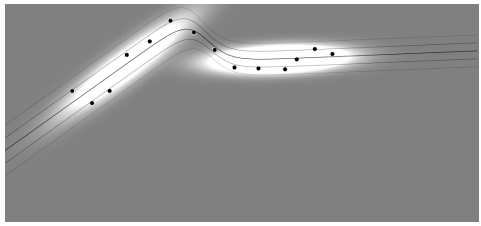
Now, to model the probability density for the hitting angle and the input, we simply set $\gamma=(\xi_1,\xi_2,\theta)$ and construct the GMM according to 3.3 and optimize the parameters with EM. Then, we set $\gamma^{\mathcal{I}}=(\xi_1,\xi_2)^T$ and $\gamma^{\mathcal{O}}=(\theta)^T$. And use (3.12b) for some unseen input ξ^* to get $\hat{h}_\theta(\xi^*)$. To get $\hat{h}_v(\xi^*)$ the θ are simply replaced by v.

To model the dependency between the hitting angle and hitting speed, a joint probability is used by setting $\gamma=(\xi_1,\xi_2,\theta,v)^T$, $\gamma^{\mathcal{I}}=(\xi_1,\xi_2)^T$ and $\gamma^{\mathcal{O}}=(\theta,v)$. Then, (3.12b) is used to get $\hat{h}(\xi^*)$.

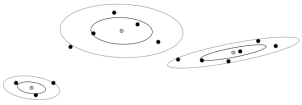
Figure 3.2 shows GMMs with 2,3 and 4 Gaussians fitted to a nonlinear data set similar to the data sets considered in this work. In figure 3.2b, we see that using only two Gaussians is enough to capture the main characteristics of this data set. Figures 3.2d and 3.2f illustrates that adding more Gaussians doesn't necessarily imply a better model.



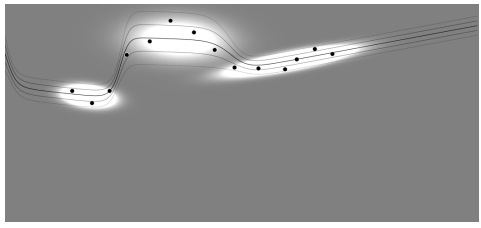
(a) GMM with two Gaussians.



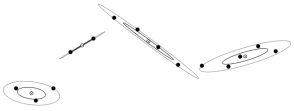
(b) GMR on GMM with two Gaussians.



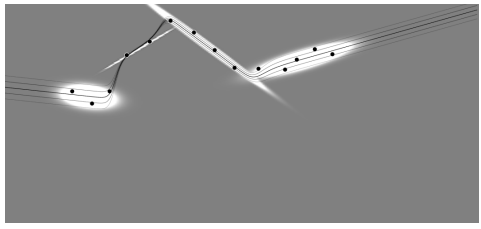
(c) GMM with three Gaussians



(d) GMR on GMM with three Gaussians



(e) GMM with four Gaussians



(f) GMR on GMM with four Gaussians

Figure 3.2: Figures (a), (c) and (e) shows a GMMs with two, three and four Gaussians respectively, fitted to the data set represented by black dots. In (b), (d) and (f), the regression curve for each GMM is plotted, along with lines at one and two standard deviations from the mean, representing confidence intervals of $\sim 66\%$ and $\sim 95\%$.

3.4 Comparison

While both GPR and GMR are powerful methods capable of capturing a wide range of functions, they have some important practical differences. In this section we discuss how some of those differences might affect how well they will perform in the context of learning hitting parameters.

3.4.1 Generalization outside range of training data

One expected drawback with GPR is that query points far from any training point will be almost zero, as discussed in example 3.1. GMR on the other hand, though it produces predictions of poor quality for inputs far from the training data, will not set the predictions to zero. This difference is illustrated in figure 3.4. To see why this might be beneficial to the case of estimating the hitting angle in minigolf, consider the following example:

Example 3.2

A player plays minigolf on a flat field. She has three training examples located along a line according to figure 3.3. When faced with the new input, she can choose one of the following two options:

1. Reason that the new input is too far from the training data and that the information from the training data is therefore useless. Hence, using the hitting angle zero is as good a guess as any.

2. Reason that even though the new input is outside the range of the training data, maybe the best guess is to try to generalize the model outside of the training range.

As is clear from figure 3.3, in this case it would be better to try to generalize the function outside the training range. Even if doing so means a very bad guess of the hitting angle that does not result in sinking the ball, it is certainly more likely to be successful than to hit the ball at zero angle.

It should be noted that while for simple field such as the one in example 3.2 it is clearly better to try to make use of whatever training data one has, this is not generally true. If the field in figure 3.3 were replaced by some highly nonlinear structure, it might well be so that guessing a hitting angle of zero is better than desperately trying to make use of distant training data. Also, there is a safety benefit of setting parameters to zero instead of making a bad guess. Experiments with GMM shows that generalizing outside the training range can result in unexpected hitting directions, adventuring the safety of a human assistant working with the robot.

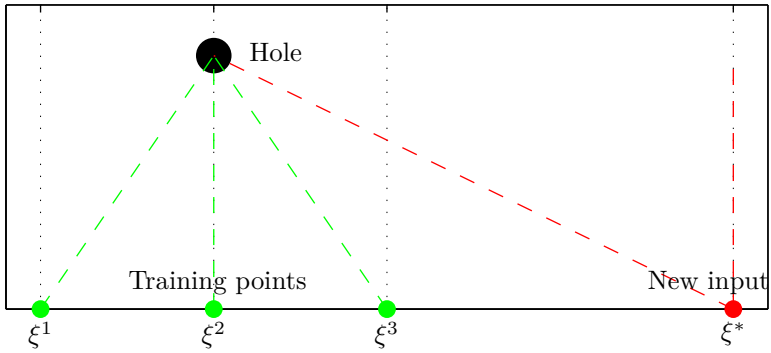


Figure 3.3: Flat field with training data and new input. The green dashed lines indicate the hitting directions for the training data. For the new input, which is distant and outside the range of the training set, which hitting direction to use is not clear cut. Two possibilities are shown by the red dashed lines.

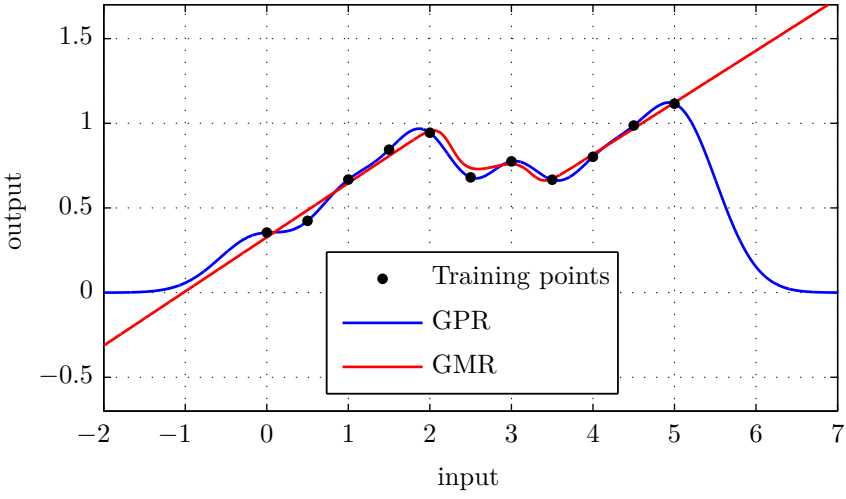


Figure 3.4: A training set with corresponding regression curves from GPR and GMR. Note how GPR drops to zero quickly outside the training range, while the GMR curve is a continuation of the "last seen" pattern. The difference in local precision is also clearly visible, as the GPR curve passes through all the training data points.

3.4.2 Local precision

One appealing aspect of GPR is the possibility to encode a measure of how much the training data should be trusted. This is done by varying the noise σ_n in (3.5). This adjustment in combination with the important lengthscale offers great flexibility in terms of model complexity. As outlined in 3.3.1, the hyperparameters are learned by optimizing the marginal likelihood of the training data. In general terms, this optimization tries to fit the training data well, but it also punishes high complexity. The result is usually a model that captures the main characteristics of the data without overfitting.

Increasing model complexity in a GMM is done by increasing the number of Gaussians in the mixture. With a small number of training data, care must be taken not to overfit the GMM to the training data by using to many Gaussians. Figure 3.2f is an example where too many Gaussians have been used, introducing artifacts not present in the training data. In practice, for small data sets, the number of Gaussians must be kept quite low. Consequently, the reproductions at the training input can be significantly different from the actual training data. Given that the training data is good, it is clearly favorable to have the training data well reproduced, so in this aspect GPR has a considerable advantage over GMR.

3.4.3 Encoding dependency between hitting parameters

In section 3.3.1 we used the probabilities $p(\theta, \xi)$ and $p(v, \xi)$ to predict $E(\theta^*|\xi^*)$ and $E(v^*|\xi^*)$. In section 3.3.2 we did the same thing but with different probability distributions, but we also used $p(\theta, v, \xi)$ to find $E(\theta, v|\xi)$.

An ideal model of the mapping would not need to encode any dependency between the parameters. The purpose of the function h is to predict for a test input ξ^* the hitting angle $\hat{h}_\theta(\xi^*)$ and the hitting speed $\hat{h}_v(\xi^*)$. Since for an ideal such estimator, these predictions would always be good, there is no reason to consider any dependency between the hitting parameters. However, this is assuming a perfect model, and this is certainly not what GPR and GMR (or any other technique for that matter) will deliver. The question is thus if considering the dependency between the hitting parameters *improves the quality of the predictions*. If this is the case, then this favors GMR since this is the only method out of the two considered that is capable of modeling this dependency. Of course, it might still be so that the local precision of the GPR still outweighs the added benefit of considering dependency in GMR. In section 6.2.4, we present results that aims at answering this question.

Chapter 4

Practical framework

As previously mentioned in section 1.1, one of the goals of this master thesis is to develop a minigolf module for the Barrett WAM arm. In this chapter, the WAM robot and the RobotToolKit interface are introduced. This is followed by a presentation of the minigolf module that has been developed for the WAM.

4.1 The Barrett WAM

The Barrett WAM is an advanced robotic arm developed by Barrett Technology. It has 7 independently controllable degrees of freedom. Four actuators are placed in the base of the robot and 3 in the wrist, making the WAM very light compared to conventional robotic arms. The maximum end effector speed of 3 m/s [12] is largely sufficient for the minigolf task. To prepare the WAM for the minigolf task, a golf club tool was designed, see figure 4.1. The golf club tool was designed and built by J.B. Keller at LASA.

The WAM can be controlled by means of a PC integrated in the base of the robot, or by a separate PC over a realtime CAN-bus that gives access to the actuators and sensors in the WAM. In this work, we use the latter.

Figure 4.1: The Barrett Whole Arm Manipulator with the custom made golf club tool attached to the end effector.

4.1.1 The RobotToolKit

RobotToolKit is a modular software package developed by Eric Sauser at LASA. New robots, environments, interfaces and control modules can easily be added.

An extensive function library allows for effortless integration of calculations such as inverse kinematics, inverse dynamics etc. A robot module, simulator and realtime control interface for the WAM are included in the package.

For simulator use, the control module interacts with a world specified by the user in a realistic simulated environment. Objects can be added to the world, and their physical properties such as friction and moments of inertia can be specified.

When operating on the real robot, the RobotToolKit uses the xenomai [25] linux realtime kernel, updating the control signals sent to the WAM at a frequency of 500 Hz.

4.2 The minigolf module

The minigolf module for the WAM is a control module written for the RobotToolKit. It uses the extensive function library available in the RobotToolKit to implement the hitting motion and its adaptation as described in sections 2.1 and 2.1.2. It was designed to have platform on which the techniques to learn the hitting parameters can be evaluated. Besides the basic functionality which is predefined in appendix A, more features have been added when considered necessary or convenient. Some key features are:

- Control the Robot to perform a hitting motion governed by a DS retrieved by GMR.
- Load different default hitting models model and hit the ball with a hitting angle and speed specified by the user.
- Load a GMM representing a hitting parameter training set and use it for regression to automatically adapt the hitting parameters to the current situation.
- Get ball and hole positions from the world by communicating over the network with a stereovision system running on another computer.

4.2.1 Operating modes and states

At each iteration, the module will execute code depending on the current state. The state is determined by an internal variable that can be altered both by the user and automatically, for example when one state is done. The different states that the module has are:

Idle When this state is activated the robot is in gravity compensation mode. This means that a human assistant can effortlessly move the WAM to any joint configuration. In this state, the WAM will remain in any joint configuration it is left.

Rest position In this state, the robot will move the end effector to a predefined rest position, and align the golf club so that it is perpendicular to the hitting direction. This is normally the state of the robot before starting a hitting motion.

Hitting motion This is the state in which the robot performs the hitting motion. The robot remains in this state until the ball has been hit. When the ball is hit, the robot automatically switches to the braking state. Figure 4.2 gives an overview of how the state of the module switches at different stages of the hitting task.

Braking When the ball has been hit, the hitting model as defined in section 2.1 can no longer be used to govern the motion of the robot, and the control scheme must be switched. In this state, the robot lifts the golf club slightly, so as to ensure that the ball is not touched more than once. At the same time, all joints are smoothly braked. When all the joints are stationary, the idle state is automatically activated.

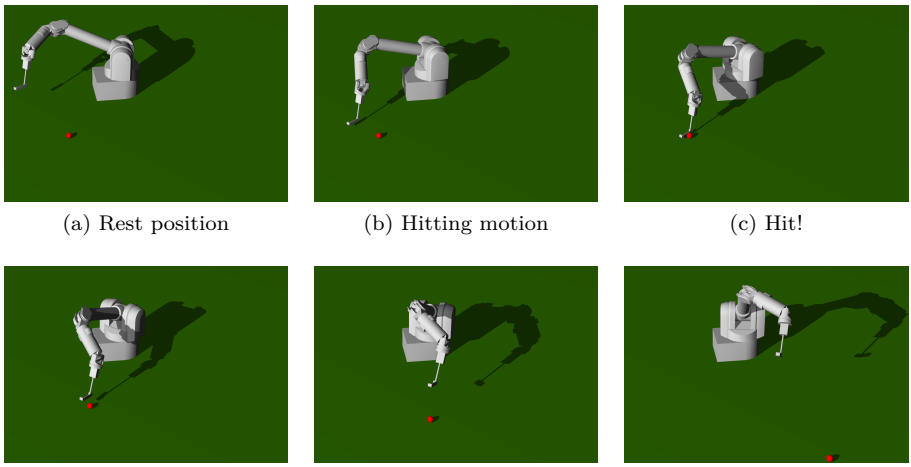
The minigolf module can be run in normal mode or simulator mode. The major difference is that the normal mode has much more strict security policy, and uses a significantly slower motion when moving to rest position. Another difference is the way the module gets information on the ball and hole positions from the world. In simulator mode, the positions of the objects in the simulated environment are readily available as continuously updated variables. In normal mode, the module receives information from a remote stereovision system tracking the ball and hole[1].

4.2.2 Design

Figure 4.3 presents an overview of how the minigolf module behaves and interacts with its environment. The three different updates represent an important separation of the module components. When using the RobotToolKit to control the WAM, a multithread process is run by the underlying robot interface, calling functions in the module in order of priority.

The reason that the module must be partitioned in this manner is that online control of the WAM requires the control signals to be updated continuously at a rate of 500 hertz. Thus, running the entire module in a single thread would have a very limiting effect on what type of computations could be done in the module. In the multithread environment, the control signals are continuously updated from a high priority thread meanwhile more computationally demanding and less critical tasks such as high level motion planning and interfacing with the user are handled by lower priority threads.

[1]It might seem unnecessary to track the hole since on a normal minigolf field that would be stationary. On the fields used in this work however, we move the hole rather than the ball when changing the situation. This is to let the robot use the same posture for hitting the ball in the different situations.



(a) Rest position (b) Hitting motion (c) Hit!

(d) Braking (e) Braking (f) Idle

Figure 4.2: Figure shows the WAM at different stages in the hitting motion. In (a), the robot is in its rest position, awaiting to initialize the hitting motion. In (b), the hitting motion has been started and the robot accelerates the end effector towards the ball. Coincidentally with the ball being hit in (c), the robot goes into the braking state. In (d) and (e), the robot is in the braking state, gently lifting the golf club while braking the joints. When all the joints have stopped moving (f) the robot goes into idle mode.

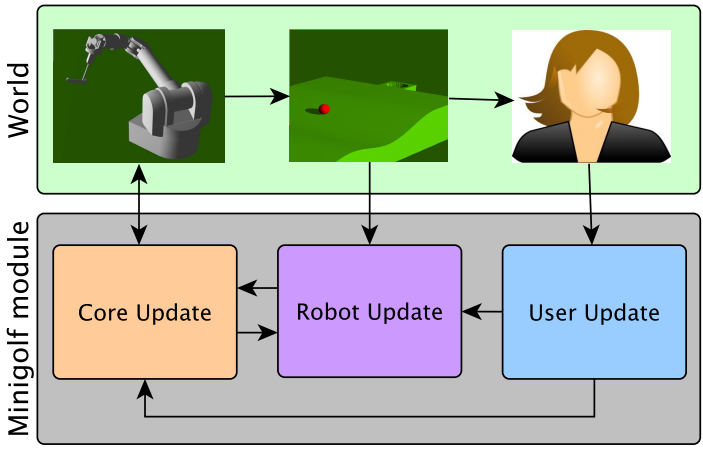


Figure 4.3: Overview of the interaction between the different components of the minigolf module and its environment. Arrows in this figure indicate transfer of information or orders.

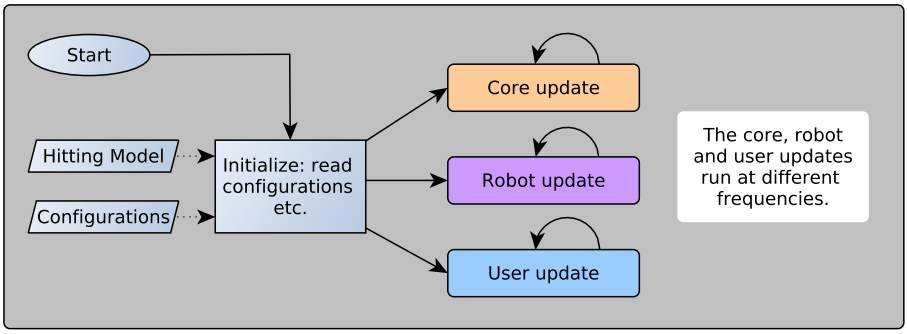


Figure 4.4: Basic flowchart of the minigolf module. Note that the core, robot and user updates are called in three different threads with different priority. Hence, they are updated at different frequencies, depending on priority and computational load.

Overview

When the minigolf module is started, the first thing that happens is that it reads a configuration file and sets up the internal variables as specified. Then, it loads the hitting model, whose parameters are loaded from a file specified in the configuration file. After the initialization, the module starts to loop its three updates until module is closed. Figure 4.4 summarizes what happens when the module is called. Note that the update frequency of the different updates are not constant. The high priority core update runs at a fixed frequency while the two lower priority updates may vary in update frequency, depending on what the current state of the module is. Each update will be treated separately in the three following sections.

Core update

The core update is the highest priority update in the module. When the module is running, the core update is looping with a frequency of 500 hertz. This means that the execution of the update must take at most 2 milliseconds. Consequently, demanding computations and system calls cannot be done in the core update.

Despite this, almost everything in the module is handled by the core update. In fact, the only things not being done in the core update are the high level Cartesian motion control with DS through GMR, position updates from the world and interfacing with the user.

In figure 4.5 the flowchart for one loop of the core update is presented. After updating the internal state of the module by reading the sensors from the robot, different things happen depending on what the current state is. The states hitting motion and rest position both have control schemes in Cartesian task space. These states are also the only ones that control the end effector orientation. As described in section 2.1, the end effector orientation is controlled so as to keep the golf club head perpendicularly aligned to the hitting direction as in figure 4.6. This involves

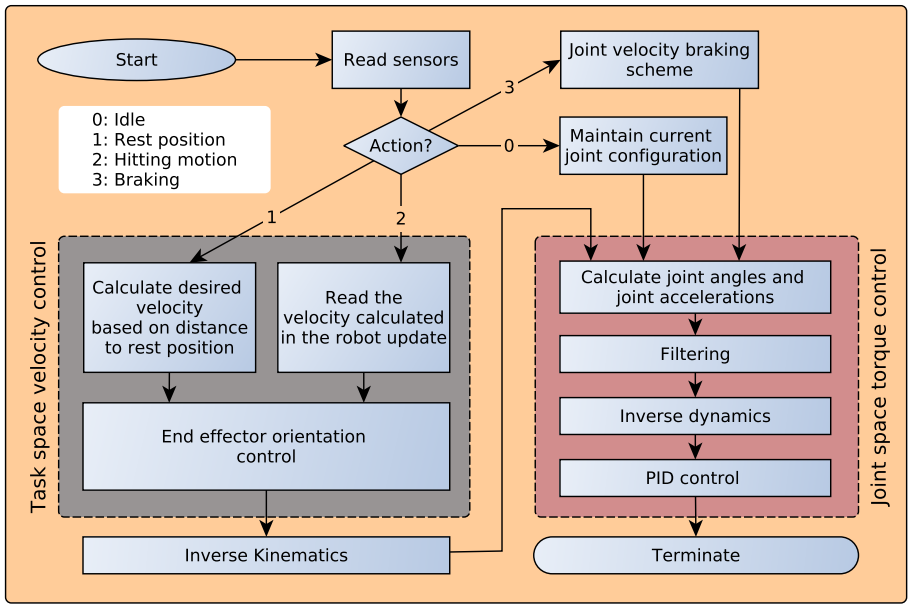


Figure 4.5: Flowchart of one iteration of the core update.

tilting the end effector by a small angle against the global z-axis to compensate for the angle between the golf club stick and head. After a change is computed in task space, it is transferred to joint space with inverse kinematics..

The output from the control module is a set of joint torques that should be applied to the WAM. The desired torques are calculated by doing inverse dynamics on the system with the current joint angles, current joint velocities and desired joint accelerations. Then, a PID-controller is used to compensate for the imperfect model of the robot dynamics used by inverse dynamics.

Note that there is also a filtering step in joint torque control loop. This filtering is necessary because new calculations on desired velocity while in hitting motion are only performed in the robot update. This has the effect that the desired velocity in task space is constant for a number of iterations (typically 5) which corresponds to the amount of core updates per robot update. Hence, the acceleration will be zero for all iterations except right after a new velocity calculation has been performed. This results in a high frequency component in the joint acceleration that makes it impossible to use the control loop described here. The problem is solved by smoothing the joint velocities using a simple finite impulse response low-pass filter before proceeding to inverse dynamics.

Robot update

The robot update is of lower priority than the core update and runs approximately once for every five core updates when in hitting motion. Figure 4.7 gives an

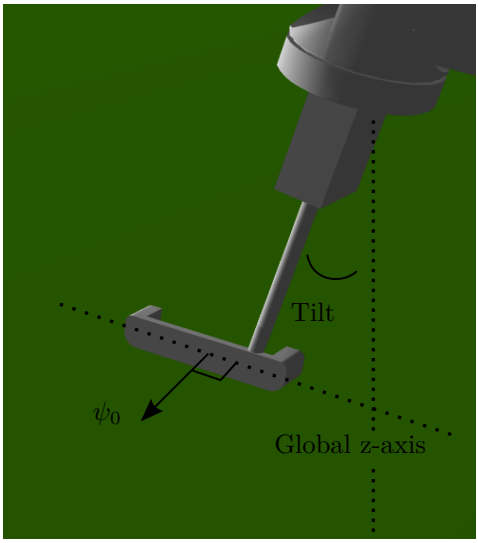


Figure 4.6: End effector orientation.

overview of the robot update.

First, ball position and hole position are perceived either by stereovision or from the simulator depending on the operation mode. Then, if recording is activated, all relevant internal variables and state information are saved together with a time stamp.

The above mentioned steps are carried out at every robot update iteration. If the current state is hitting motion, the desired velocity is calculated in the robot update. This is done through GMR on a GMM representing the DS of the hitting motion, as described in section 2.1.1.

User update

The lowest priority update is the user update. The user can type commands into a console at any time. Then, when the user update runs, it starts by checking if a command has been entered in the console, and if a recognized command has been entered then action corresponding to that command are executed. An example of a such command is *start hitting motion*. When the user update receives this command it will change the internal state of the module to hitting motion. This means that the robot and core updates will change their behavior according to figures 4.7 and 4.5.

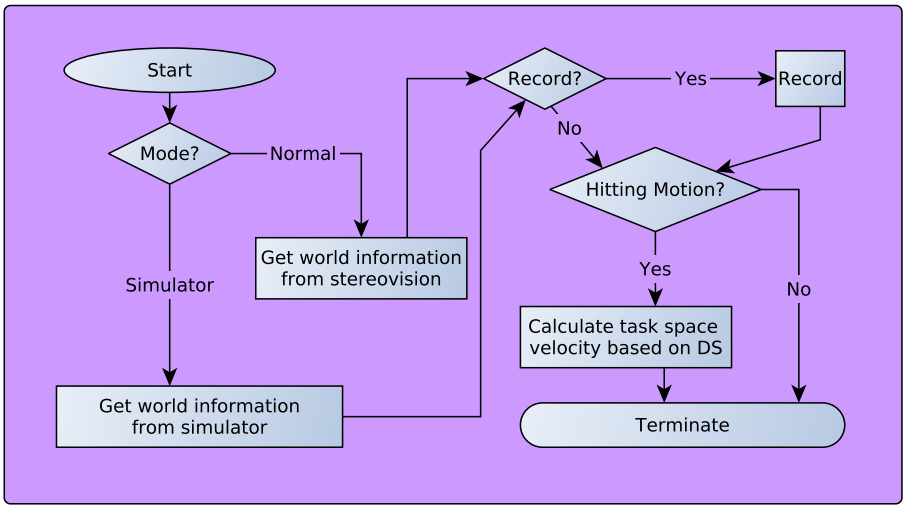


Figure 4.7: Flowchart of one iteration of the robot update.

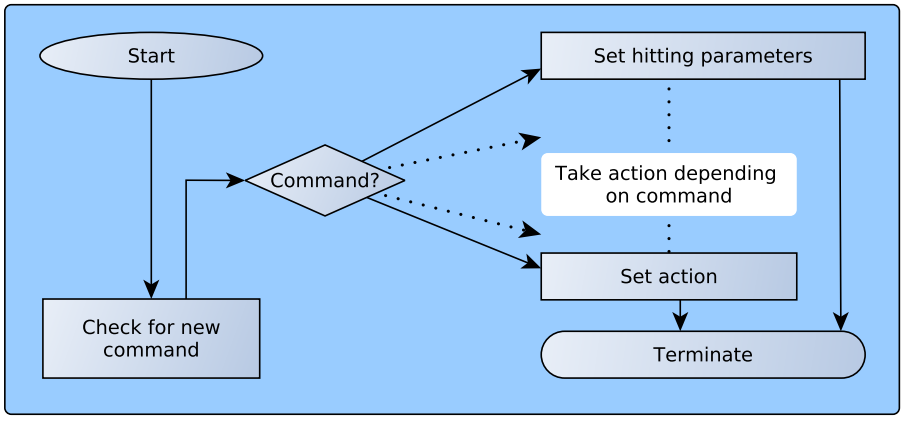


Figure 4.8: Flowchart of one iteration of the user update.

Chapter 5

Experiment setup

Chapter 3 gave a theoretic background of the techniques used to learn the hitting parameters. Chapter 4 introduced the practical framework for this work. In this chapter, we present the different setups used to evaluate the different learning techniques in the context of the minigolf task, as well as measuring the performance of the minigolf module.

5.1 Data Collection

As explained in 3.1, in this work we use only good examples for learning the hitting parameters. To collect a data set, the robot is operated by a user that manually specifies hitting angle and speed. When a successful setting is found, the hitting parameters are saved to the data set along with the input vector. Once the collection is complete, the data set is stored in a file.

As discussed in section 3.2, it is vital to impose continuity on the data being used for training. In practice, this is done by choosing a style or strategy with which to hit the ball. For an advanced field this is especially important. Of course, when collecting data it might not always be so easy to know what strategy one has employed after having found a successful parameter setting. Two strategies which are easy to follow and have therefore been utilized in this work are:

Minimum speed The goal of this strategy is to hit the ball such that it has the lowest possible speed at the moment of sinking.

Minimum angle With this strategy, the ball should be hit as closely as possible along a straight line through the initial ball position and the hole. This often means that the ball must be hit with a relatively high hitting speed.

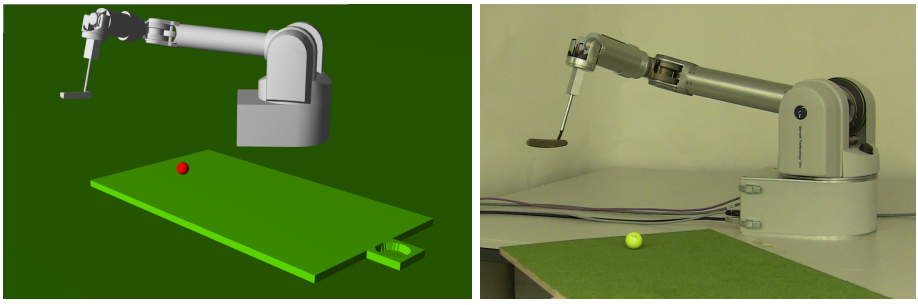
5.2 Fields

The level of difficulty of the minigolf task is entirely dependent on the field. Playing at an easy field, such as a flat field, is not nearly as challenging as facing a field

with a curved surface. To evaluate the system, experiments on several different fields are conducted. The fields used in the simulator are generated as mesh data in Matlab and imported to RobotToolKit which builds shapes in the robot environment based on this data. The fields used for the real robot were built by J.B. Keller at LASA. They are covered with a fake grass material, of the type often found on a minigolf course. Below follows a description of the different fields used in this work.

5.2.1 The flat field

The simplest possible minigolf field, the flat field, is the first field used for experiments. For flat fields, it is possible to hit the ball using a wide range of speeds and still sink the ball provided that the angle is correct. Thus, hitting speeds has not been learned for data sets collected on the flat field. Thus, for the flat field experiment, ξ was only changed along a straight line, effectively removing one of its dimensions from the learning problem. Experiments on the flat field were conducted in the simulator as well as on the real robot.



(a) (b)

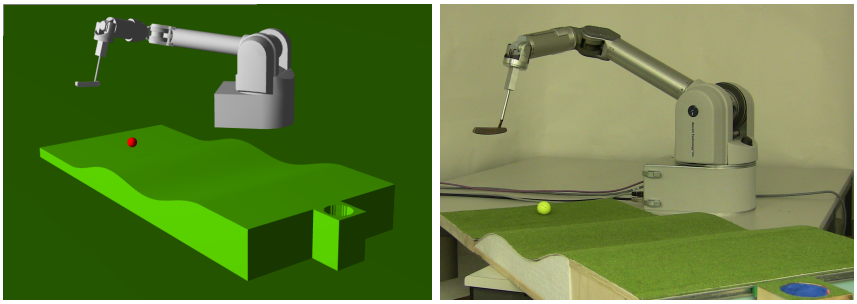
Figure 5.1: The flat field in simulator and with real robot.

5.2.2 The multiple hills field

The multiple hills field consists of an upward slope with two bumps. This field is significantly harder to play than the flat field. The added difficulty is partly due to the fact that it is harder to guess the hitting parameters. Another reason is that this field tends to amplify errors in the launch angle, thus forcing the player to be more precise when executing the hitting motion. The multiple hills field was used for experiments both in the simulator and with the real robot, see figure 5.2

5.2.3 The regular hill field

This field is often found on real minigolf fields. Those who have tried playing minigolf on this field know how hard it is. While being very hard to play, interestingly this field has a relatively simple hitting parameter mapping. The only



(a) (b)

Figure 5.2: The multiple hills field in the simulator and with the real robot.

valid hitting angle for any situation is to hit the ball in a straight line towards the hole, i.e. along ξ. Thus, the problem is reduced to finding an appropriate hitting speed. Despite the simple hitting parameter mapping, the regular hill field is indeed a very hard field to play. This is because the field has a tendency to "punish" even a small error in the hitting angle. Thus, to play this field, the user needs a very precise execution of the hitting motion, which is probably why human players experience such difficulty with this type of field. The regular hill field was only implemented in the robot simulator, see figure 5.3.

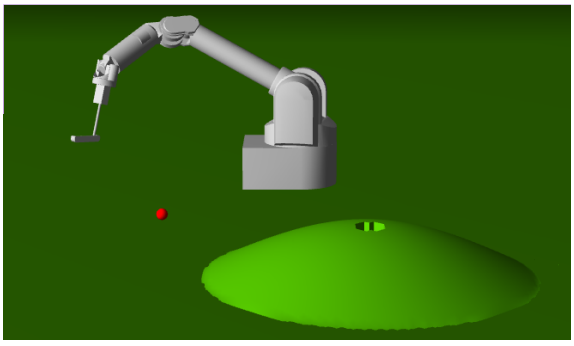
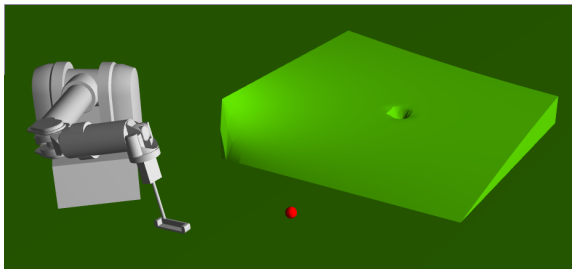


Figure 5.3: The regular hill field in the simulator.

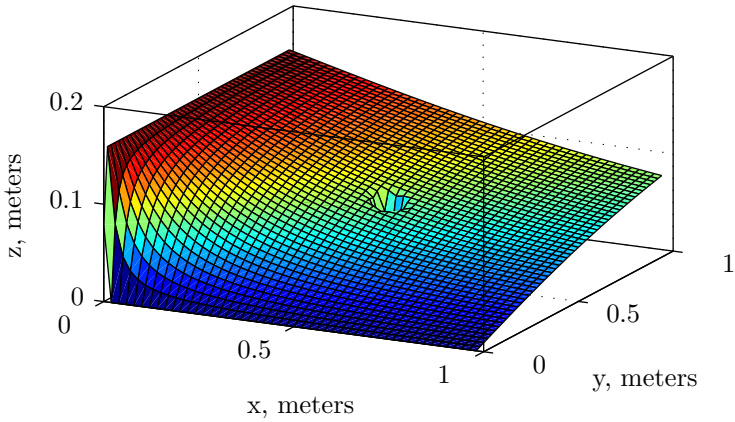
5.2.4 The arctan field

The arctan field was named by its origins: the shape of the field is a scaled evaluation of the arctan function over a two-dimensional grid. The arctan field is by far the most advanced minigolf field considered in this work. Not only is it very hard to predict hitting parameters, but errors in both launch speed and angle often lead to severe misses (i.e. the ball does not even come close to the hole). The arctan field was used for experiments in the simulator, see figure 5.4a. Figure

5.4b is the same field but with a coded color scheme, so that the height variations can more easily be seen.



(a)



(b)

Figure 5.4: The arctan field in the robot simulator (a) and with a colormap to facilitate perception the field structure (b)

Chapter 6

Results

In this chapter, after analyzing the performance of the minigolf module, we use the fields presented in section 5.2 to evaluated the modeling performance of GPR and GMR (refer to chapter 3) in the context of predicting hitting parameters. We measure the learned skill of the WAM controlled by the minigolf module endowed with the different learning techniques presented.

6.1 Performance of the minigolf module

Before evaluating the statistical methods used to choose hitting parameters based on input data, it makes sense to investigate the accuracy with which the system can actually reproduce a hitting motion with given hitting parameters. The 7 DoF WAM used in this work has a joint angle resolution of 0.008° [12]. However, it is likely that the change of speed, which can be seen as a perturbation to the original hitting model DS (refer to section 2.1.2), introduces an error in hitting angle. Other components in the minigolf module, e.g. the inverse kinematics, is also likely to introduce an error. As all recordings indicate that there is no significant error in the hitting speed at the hitting point, we do not analyze the error in the hitting speed.

6.1.1 Launch angle error

Let the launch error ϵ_l be the error introduced by the system when trying to launch a ball at the desired angle θ, with the resulting actual launch angle θ_l:

$$\theta_l = \theta + \epsilon_l \tag{6.1}$$

The launch error is likely to be caused by:

- Approach errors, i.e. the angular offset between the approach trajectory of the end effector and the hitting direction. In the ideal, i.e. when the approach error is zero, the trajectory of the end effector is aligned with the desired hitting direction, as explained in section 2.1. The approach error

can be caused by the perturbation introduced in the hitting model DS by changing the speed (refer to section 2.1) or the inverse kinematics.

- Error in the control of the end effector orientation, refer to section 4.2.2.
- Nonlinearities on the ball. The dimples on the minigolf surface can affect the launch angle.
- Nonlinearities on the field surface.

On all data sets from simulation, the launch error is easily acquired since the tracking of the ball is perfect. One data set was collected on the simulator flat field specifically for this purpose. 10 trajectories were recorded when trying to hit with angle $-10°$, $0°$ and $10°$. As can be seen in figure 6.1, the launch error for each hitting angle is significantly biased. This is not as serious an issue as one might think, since the learning technique will automatically compensate for this bias[1]. What would be more alarming would have been if there was a large spread in the error distribution. Of the errors shown in figure 6.1, hitting angle $-10°$ has the biggest spread with a standard deviation of $0.042°$. This is a small error which will only bear significance on highly challenging fields such as the regular hill field.

Unfortunately, the tracking of the ball from the real robot is not accurate enough to allow any useful estimation of the launch angle. However, the experiment in the simulator shows strong correlation between approach errors and launch errors. It is a fair assumption that the correlation between launch error and approach error is strong also for the real robot. The approach error from a data set from the real robot can be seen in figure 6.2. All data sets from the real robot give similar results, indicating that the approach error is indeed of the same character as in the simulator: heavily biased with a small variance. We are unable to quantify this variance though. One would have to assume that it is at least as big as the one obtained from the simulator.

Since models that work well in practice (results shown in section 6.3) have been built for the multiple hills and flat fields on the real robot, we can at least conclude that for fields of this complexity, the accuracy is good enough.

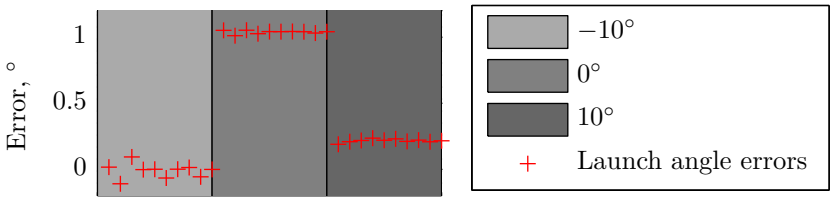


Figure 6.1: Launch angle errors for a set of 10 attempts for three different angles on the flat field in the simulator.

[1] In the training phase, the error is compensated for by the user who simply selects parameters that renders successful attempts. It makes no difference if the reason the parameters must be chosen in a certain way is the field features or to compensate for an error introduced by the robot when hitting.

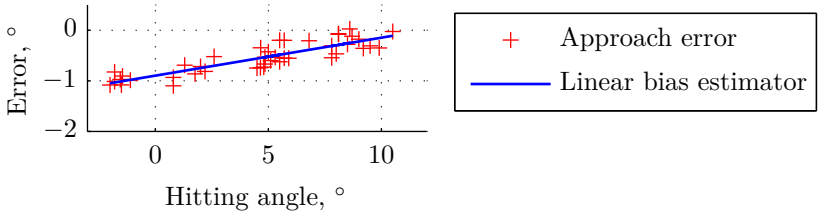


Figure 6.2: Approach errors for a range of hitting angles. A linear estimator is plotted to along with the errors to illustrate the linear structure of the bias.

6.2 Learning techniques

In this section, the results from applying GPR, GMR and joint GMR as described in chapter 3 are presented.

6.2.1 The flat field

The flat field experiment was conducted in the simulator and on the real robot with as similar settings as possible. The flat field in the simulator was correctly scaled and placed at the same position with the same orientation as the flat field for the real robot.

The inputs in the training set collected for the flat field consists of 11 equally spaced points along a line. As explained in section 5.2.1, learning the hitting speed for the flat surface is not interesting. For obvious reasons, this excludes the joint GMR method described in section 3.4.3 as a viable option. In effect, we are interested in finding a one-to-one mapping from position along the line in input space to the hitting angle. To do this, we use GPR and optimize the parameters of the ARD covariance function for maximum likelihood on the training set and GMR with EM on the training set. A first glance at the training set in figure 6.3 reveals the linear structure that one would expect from a flat field. Clearly, a relatively simple model will adequately model the joint angle mapping for this field.

Playing on the flat field allows for little fantasy. In fact, there is only one possible strategy (refer to section 3.2), and that is to hit the ball in a straight line towards the hole[2]. Thus, assuming high model complexity for the hitting parameter mapping for this field is likely to result in pointless overfitting. Thus, only two Gaussians were used for the GMR, and the noise prior for the GPR is set to a relatively high value before optimizing (refer to section 3.3.1).

We validated the performance of each method based on the mean square error (MSE) over a validation data set:

Definition 6.1 (MSE) *The mean square error of the function estimate* $\hat{h}_\theta(\xi)$ *on*

[2]We do not consider strategies that involve hitting the ball with spin, in which case the ball could be hit with a wide range of different angles.

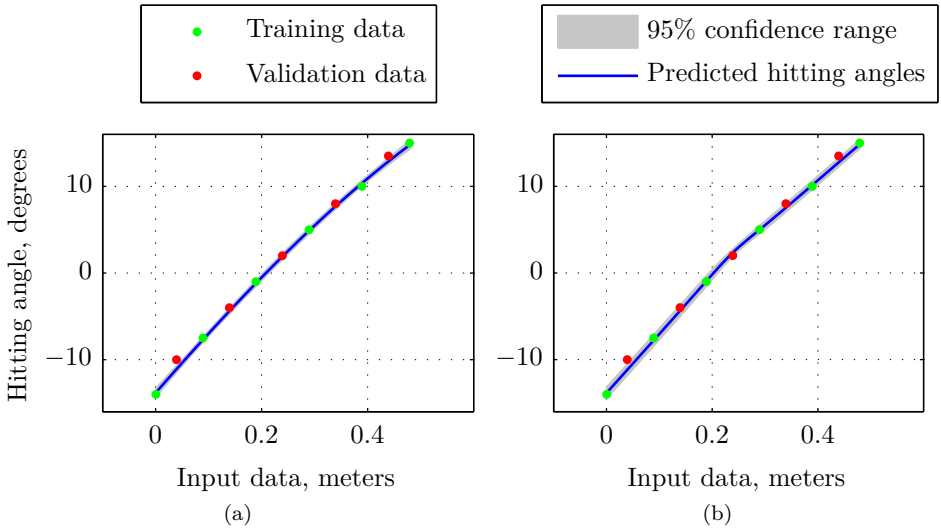


Figure 6.3: Resulting regression curves from models (GPR left, GMR right) based the training set collected on the flat surface with the real robot.

validation set $\{\theta^i, \xi^i\}_{i=1}^{N_v}$ *is defined as:*

$$MSE = \frac{1}{N_v}\sum_{i=1}^{N_v}(\theta^i - \hat{h}_\theta(\xi^i))^2 \tag{6.2}$$

The MSE from the models in figure 6.3 are presented in table 6.2. Although this measure suggests that GPR performs better than GMR in this situation, it should be noted that for the flat field, both of the models are satisfactory in that they both produce values that sink the ball for new inputs.

	GPR	GMR
MSE	0.1643	0.2226

Table 6.1: The mean square error resulted from GPR and GMR predicting the hitting angle on the validation data set for the flat field.

6.2.2 The multiple hills field

The experiments on the multiple hills field were conducted both on the robot and in the robot simulator. As in the experiments with the flat field, the input is one-dimensional and we consider only the hitting angle for learning. Although we considering the same one-to-one learning problem here as on the flat surface, there is an important difference. For the flat field, we only have to learn the hitting angle and then we can use this angle with any speed that is fast enough to bring the ball to the hole, and slow enough to not make the ball skip over the hole when it reaches it. For the multiple hills field, however we are still only learning the hitting angle, if a hitting speed different from the one used for training is used in combination with a predicted hitting angle, the attempt will almost certainly fail. The reason for this is that by choosing a fixed hitting speed we chose a specific strategy (refer to section 3.2). This means that any hitting angle predicted by a model built based on this training data belongs to the strategy used for training.

The shape of the field naturally indicates that a more complex hitting angle mapping than the one for the flat field should be considered. This intuition has been experimentally verified by restricting the complexity as with the flat field, which resulted in a very poor model in terms of success rate for attempts at unseen inputs. Hence, it appears as though the nonlinearities represented by the training data must be respected to a higher extent. Recall that higher model complexity can be modeled by increasing the number of Gaussians in the GMM and decreasing the noise prior and lenghtscales in GPR.

In figure 6.5 the resulting models are shown. Note that even though the setup was the same in the simulator as on the real experiment, the data collected from the real robot seem to have a more complex structure than the data set collected from the simulator. This can be seen very well when comparing figures 6.5a and 6.5c, paying special attention to the shaded area representing the predictive confidence. Both models have been optimized with similar noise priors, but have resulted in very different length scales (refer to definition 3.2). This results in the higher predictive variance in between the training data.

The difference is illustrated in figure 6.4, where the regression curves for the simulator and the real robot are plotted together. One tempting conclusion from figure 6.4 would be that the models are overfitting the training data, and that the sublinear structure shared by models from the simulator and the real robot are the true functions. This is not the case though, as experiments show that both the robot simulator and the real robot need the nonlinear characteristics present in each model to predict well the hitting angles for new inputs. Using a model based on training data from the simulator to predict hitting angles on the real robot fails, and vice verse. Some possible reasons for this discrepancy are:

- Subideal modeling of environment. Great effort was made at modeling the field and ball correctly, but small details such as the golf ball dimples[3] and filt structure of the field material where not included in the simulator models.

[3]The small depressions on the surface of the golf ball. These have the effect of improving the aerodynamics of the ball in flight [26], but have no useful function in minigolf.

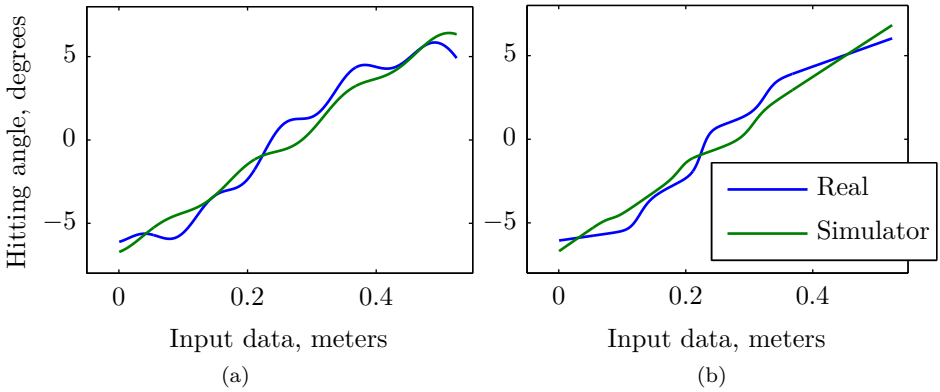
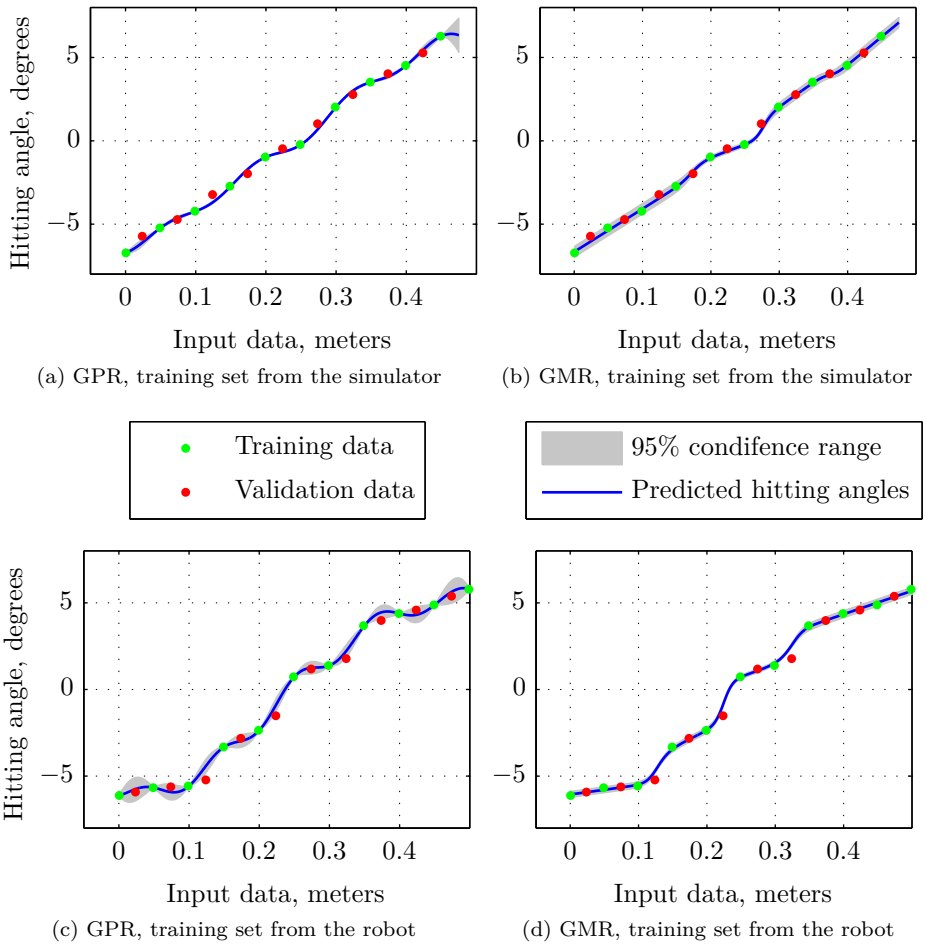


Figure 6.4: Comparison of the results obtained with training data from the simulator and the real robot for GPR (left) and GMR (right).

- Robot condition. The simulated robot enjoys ideal specified characteristics, whereas the real robot has been used for previous experiments and may have changed characteristics slightly due to wire tension differences etc.
- Subideal physical modeling of the golf club tool. The inertia properties of the golf club are critical to the control of the real robot (see chapter 4).

	GPR	GMR
Simulator MSE	0.0751	0.0322
Real MSE	0.1999	0.1655

Table 6.2: The MSE of GPR and GMR predicting hitting angles for the validation data set for the multiple hills field.



(a) GPR, training set from the simulator

(b) GMR, training set from the simulator

(c) GPR, training set from the robot

(d) GMR, training set from the robot

Figure 6.5: Resulting models with GPR and GMR for training sets collected from the real robot and from the robot simulator. Note the higher complexity present in the data from the real robot.

6.2.3 The regular hill field

The regular hill field is an interesting field in that choosing the hitting angle is trivial (refer to section 5.2.3), yet it is one of the most difficult type of fields one encounters at a minigolf course. Since there is only a single possible strategy for finding the hitting angle for each input[4], the player needs to learn what hitting speed should be used for each angle. Since at each point in input space we can calculate exactly what hitting angle should be used, learning the hitting angle for this field might seem unnecessary. However, in order to compensate for the biased error in launch direction (refer to section 6.1.1), the hitting angle must be learned anyway. Initial experiments using the theoretical hitting angle were performed, and all attempts failed.

Indeed, as can be seen in figures 6.6b and 6.6d, the structure of the theoretical hitting angle is clearly present in the learned models. In 6.6a, two Gaussians were used. This gave only a slightly better result than using only one Gaussian, which is equivalent to putting all the predictions in a plane. Using three for this data set invariably results in models with very poor predictions at the validation points. As can be seen in figure 6.6c, the optimization of the hyperparameters in GPR have resulted in a fairly simple model when compared to GMR. In this case, according to table 6.3, GMR gave a slightly better prediction of the validation data set.

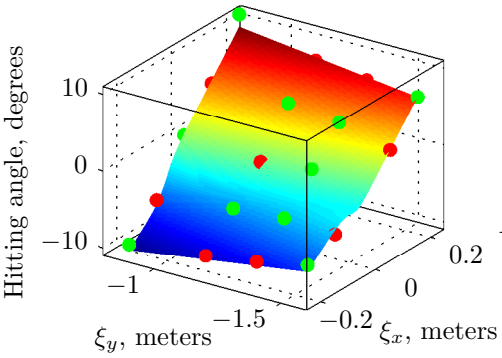
While the estimating the hitting angle for the regular hill is a fairly easy task, choosing the correct hitting speed less obvious. The ball must be hit fast enough so that it manages to climb the face of the field, yet slow enough not to skip over the hole. Experiments show that the regular hill field is more sensitive to high hitting speeds than the flat and multiple hills fields. Looking at the field layout in figure 5.3, this makes sense, as the ball will have a vertical velocity component once it reaches the top of the hill in immediate vicinity to the hole. The tendency of the regular hill field to punish launch angle errors is stronger for low hitting speeds. Thus, in search of a hitting speed model that minimizes the sensitivity to errors in the hitting angle, a fairly hitting speed was used. However, this is a hard strategy to follow, resulting in a data set that may contain samples of neighboring strategies. This might explain the comparatively poor results[5] of GPR versus GMR in the hitting speed prediction as can be seen in figure 6.7. Also, the poor ability to generalize far from the training data of GPR as discussed in section 3.4 is very clear. The two leftmost validation points in figure 6.7b are too far from any training data to get support from the training points, and the prediction consequently drops as explained in example 3.1. GMR however, generalizes much better and provides good quality predictions at all validation points.

6.2.4 The arctan field

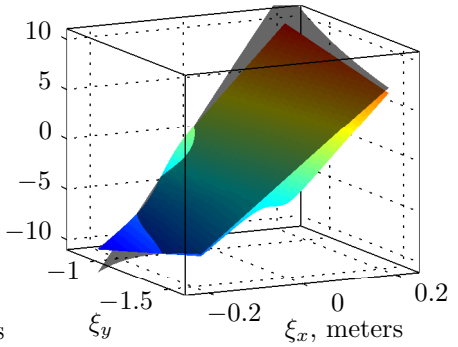
Compared to the other fields considered in this work, the arctan field is significantly harder to play. Within a given strategy, the range of possible angles and speeds

[4] In fact, there is generally a small interval of valid hitting angles, but all of those are samples of the same strategy. Refer to section 3.2.

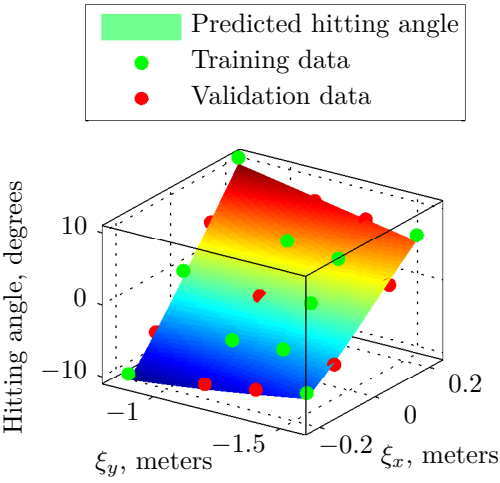
[5] Recall that GPR generally has better local precision than GMR. This is a disadvantage if some of the training data points are bad.



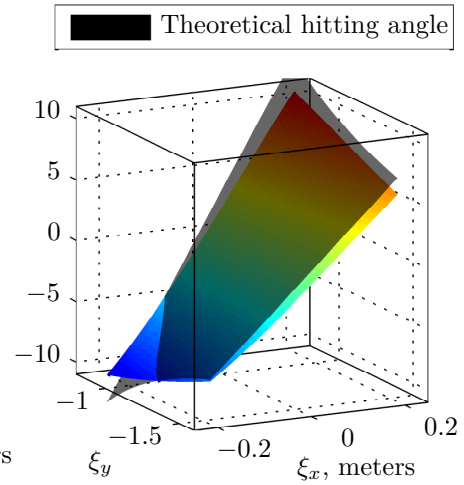
(a) GMR hitting angle



(b) GMR hitting angle with theoretical hitting angle overlaid

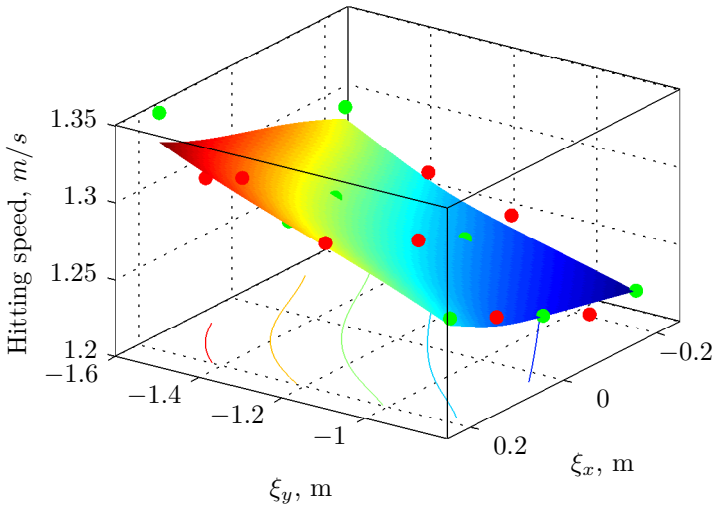


(c) GPR hitting angle

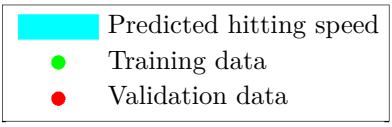


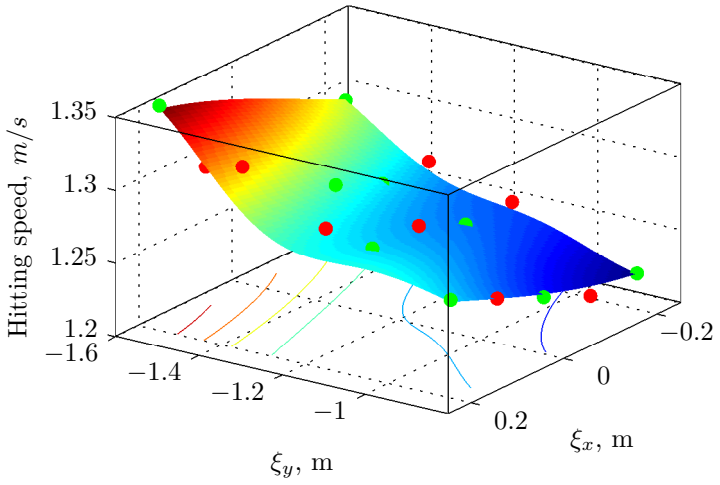
(d) GPR hitting angle with theoretical hitting angle overlaid

Figure 6.6: Figures show hitting angle prediction over input space for the regular hill field. In (a) and (c), the hitting angle predictions from GMR and GPR respectively is plotted along with the training and validation data. In (b) and (d), the hitting angle predictions are plotted with the theoretical hitting angle, represented by the transparent dark surface.



(a) GMR hitting speed





(b) GPR hitting speed

Figure 6.7: Figures show hitting speed prediction over input space for the regular hill field. Note the poor prediction of GPR at the two leftmost validation points. GPR does a very bad job at predicting those values, as they are to far from any training data. GMR handles this better in this case.

Regular hill	GPR	GMR
Simulator MSE, angle	0.1946	0.1884
Simulator MSE, speed	0.4460×10^{-3}	0.3562×10^{-3}

Table 6.3: MSE for GPR and GMR over the validation set for the regular hill field.

Arctan field	GPR	GMR	joint GMR
Simulator MSE, angle	0.22	0.38	0.34
Simulator MSE, speed	0.12×10^{-3}	0.059×10^{-3}	0.063×10^{-3}

Table 6.4: MSE for GPR and GMR over the validation set for the regular hill field.

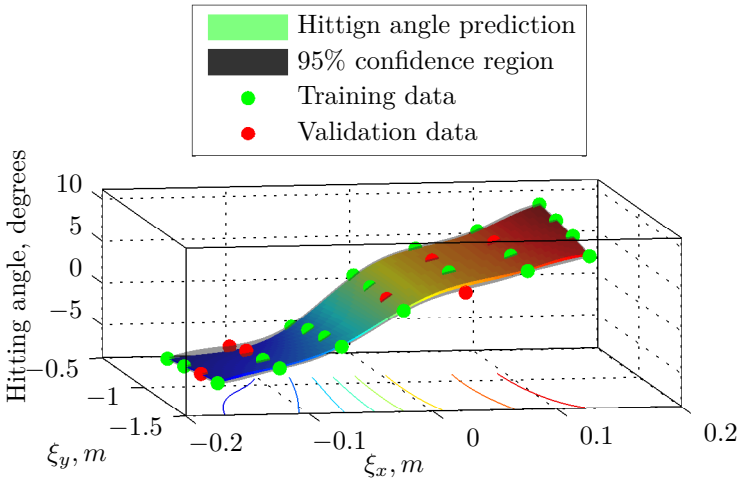
are very narrow. Furthermore, the arctan field offers a huge number of different strategies, which as discussed in section 3.2 makes the training phase harder, as the training data all come from one same strategy. In this experiment, the minimum speed strategy is employed. To make sure that this strategy was employed for all points in the data set, the field was given a very shallow hole. Using this hole, a ball with high speed will instantly roll out again if it is sunk, effectively limiting the choice of hitting parameters to the minimum-speed strategy.

Contrary to the fields in sections 6.2.1, 6.2.2 and 6.2.3 where the learning was focused on either hitting angle or hitting speed, for the arctan field it is absolutely critical to learn good models of both the hitting angle and speed. Thus, it may also be interesting to look at the dependency between these parameters. Therefore, in addition to GPR and separate GMMs, we will in this section use a joint GMM (refer to section 3.3.2) encoding both hitting parameters in one GMM.

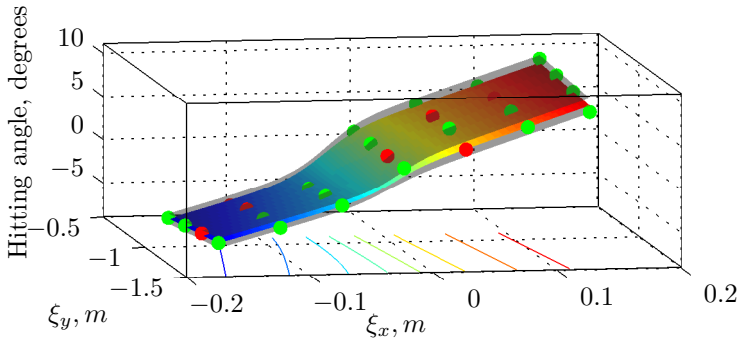
The data set consists of 28 evenly spaced points. Since this is a significantly harder field than the others, it was assumed that a larger training set would be needed to get an acceptable model. Of the data set, 9 points were chosen at random[6] resulting in a training and validation set as in figure 6.10. As established in section 3.4, GPR is generally more sensitive than GMR to the structure of the data in input space. In an attempt to reduce the impact of the training set selection, models were fitted to 1000 different training sets selected randomly from the data set. The MSE for the hitting angle and speed were collected for these 1000 models, with means given in table 6.4.

Examining the MSE for the models optimized for the randomly selected training sets, it is clear that no matter how the training data is selected, GPR generally have a few points with very poor prediction. In fact, the sum in (6.2) is usually dominated by only a small amount of validation points with very poor predictions while the majority of the validation points have very good predictions. For GMR however, the MSE is more evenly distributed over the validation set.

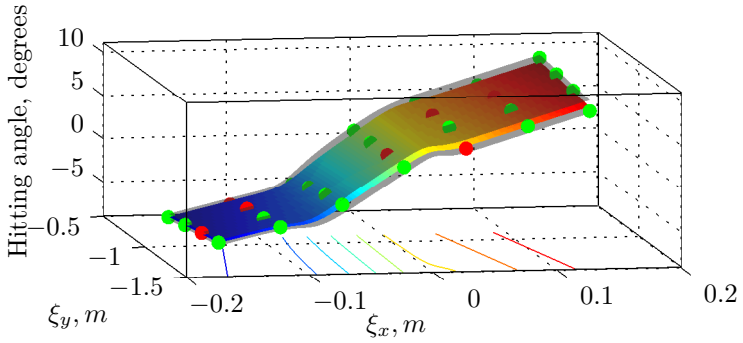
[6] 9 indices from the data set were generated from a uniform integer distribution over the set of data set indices, excluding the corners of the data set which were always included in the training set.



(a) GPR, hitting angle, arctan field

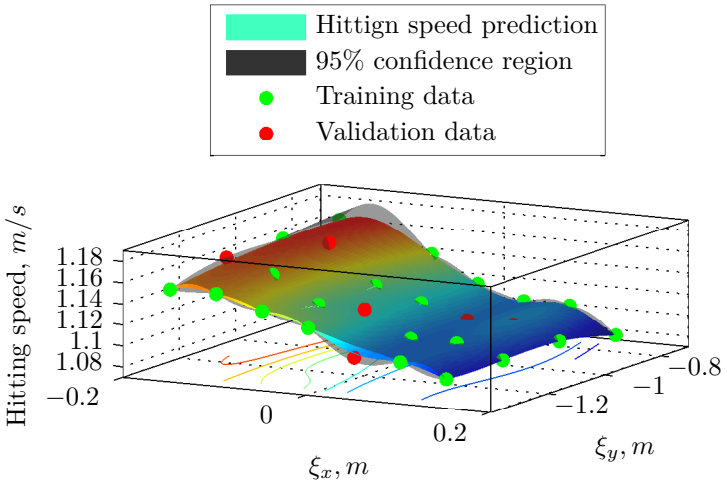


(b) Separate GMR, hitting angle, arctan field

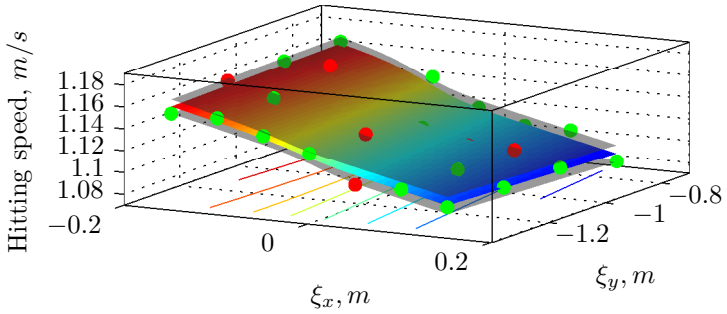


(c) joint GMR, hitting angle, arctan field

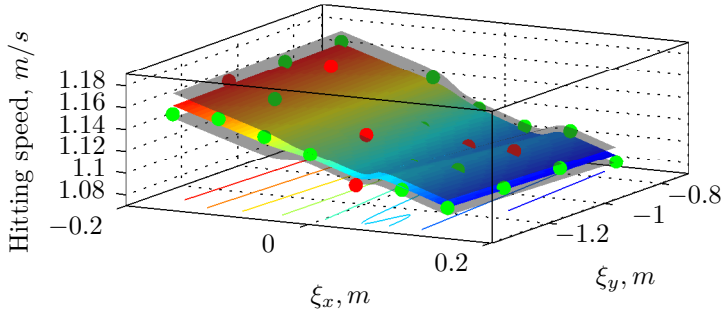
Figure 6.8: Hitting angle for the arctan field. The predictions over the input grid are given by the colored surface. A confidence envelop corresponding to two standard deviations of the predictive distributions are plotted as gray transparent surfaces.



(a) GPR, hitting speed, arctan field



(b) separate GMR, hitting speed, arctan field



(c) joint GMR, hitting speed, arctan field

Figure 6.9: Hitting speed for the arctan field. The predictions over the input grid are given by the colored surface. A confidence envelop corresponding to two standard deviations of the predictive distributions are plotted as gray transparent surfaces.

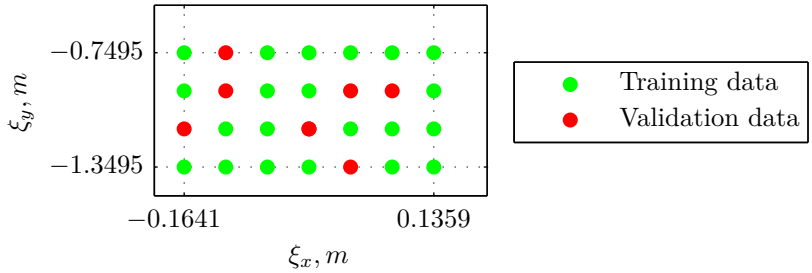


Figure 6.10: The Data set used for the arctan field experiment. Green points indicate points used for training, and red points indicate 9 randomly selected points used for validation.

6.3 System performance

In sections 6.1 we studied briefly the performance of the robot and in section 6.2 we studied how well the learning algorithms predicts the hitting parameters. In this section, we will present how good a minigolf player the robot is when used in combination with the different learning techniques. This is done by letting the learning algorithms predict hitting parameters for a set of randomly selected test points in input space. The world is then set up so as to match one of these test points, and the robot hits with the predicted parameters. The same procedure is done for all the points in the test input set, and the result of each attempt is recorded. Table 6.5 summarizes the results from these experiments.

For the arctan field, the results presented here are from models trained with the training set in figure 6.10, as well as models trained with the full data set, i.e. all the points in figure 6.10.

The first comment one can make about the results are that the WAM endowed with the minigolf module and either of the proposed learning algorithms is a very good minigolf player, likely superior to most humans[7]. Even though all learning algorithms give a high succes rate for all fields, the overall pattern is that GPR is slightly better. This is interesting, as for most fields the opposite was indicated by comparing MSE in the previous section. One possible reason for this may that the high MSE for GPR tends to derive only from a small subset of the validation data, while the majority of the validation points have very good predictions.

A less surprising piece of information is that attempts on the real robot tend to have a lower succes rate attempts for the same field in the simulator. As discussed in section 6.2.2, the training data from from the real robot has a higher inherent complexity. This is one reason that the results might be worse for the real robot versus the simulator. Another reason is the error in the execution of a hitting motion with desired hitting parameters. As discussed in section 6.1, while such an error has not been quantified for the real robot, the default assumption would have to be that the error is more significant on the real robot than on the simulator.

[7]It has been experimentally established that it is far superior to the author.

Table 6.5: Results from letting the robot play minigolf with the different learning algorithms and fields.

Situation	Method	Attempts	Successful	Ratio
Flat field				
Simulator	GPR	30	30	100%
Real Robot	GPR	10	10	100%
Simulator	GMR	30	30	100%
Real Robot	GMR	10	9	90%
Multihill field				
Simulator	GPR	30	26	87%
Real Robot	GPR	10	8	80%
Simulator	GMR	30	28	83%
Real Robot	GMR	10	8	80%
Regular hill field				
Simulator	GPR	30	26	87%
Simulator	GMR	30	24	80%
Arctan field				
Simulator	GPR	30	28	93%
Simulator	GMR	30	24	80%
Simulator	joint GMR	30	25	83%
Arctan field, full data set				
Simulator	GPR	30	30	100%
Simulator	GMR	30	24	80%
Simulator	joint GMR	30	27	90%

Chapter 7

Conclusion and Future Works

In this thesis we addressed the problem of adapting robot motions, specificly we looked at hitting motions in the context of a minigolf task. Although this is a very narrow topic, the results are likely transferable to similar problems, i.e. situations were a basic governing model should be parametically altered so as to fulfill a task goal under differnt circumstances.

The basic assumption that our proposed method depends on are that to solve the minigolf task, it is only necessary to find one combination of valid hitting parameters for each situation. While this does not allow the robot to choose the strategy, it was assumed that this would be enough to solve the task and that learning of the task using this approach would require only a small set of training examples.

The results indicate that this assumption was indeed reasonable. Using between 10 and 30 data points, the two evaluated statistical learning techniques both resulted in robust models of the hitting parameter. The comparison of theese methods was a large part of this work, but deciding on a winner is not clear cut. In terms of usability, there is no difference since in practice, both models must be provided with prior information (number of Gaussians for GMR and reasonable inital guesses of the hyperparameters for GPR) depending on the field and the structure of the training data in input space. In terms of the performance of the robot when used in combination with the developed minigolf module and the different methods, a small advantage can be assigned to GPR. It should also be noted that the increase in computational cost when increasing the size of training data is much bigger for GPR[1] than for GMR.

The shortcoming of our approach is the necessity of a human assistant delivering good examples to the robot. Although evidently a high performance learning system has been implemented, the system is not independent in that it can teach itself. Also, the system proposed here does not improve when new good examples

[1] Doing GPR involves inverting a matrix the size of the training set, refer to (3.7b)

are found. The system could be incrementally improved by continuously adding newfound succesful data to the training set.

An interesting area of future research would be to combine the system proposed here with the reinforcement learning approach presented in [19] so as to quickly have a functioning model that has the role of guiding the parameter selection to the correct strategy as well as providing a reasonable starting point for exploration. Furthermore, extending such a system to also exploit information from bad examples is an interesting and hard problem that if solved could lead to a more human-like learning scheme in robotics.

Bibliography

[1] S.Schaal, "Is imitation learning the route to humanoid robots?," *Cognitive Sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[2] S.Schaal, A.Ijspeert, and A.Billard, "Computational approaches to motor learning by imitation," *Philosophical Transactions: Biological Sciences (The Royal Society)*, vol. 1431, pp. 537–547, 2003.

[3] D.Kulic, W.Takano, and Y.Nakamura, "Incremental learning, clustering and hierarchy formation of whole body motion patterns using adaptive hidden markov chains," *The International Journal of Robotics Research*, vol. 27, no. 7, pp. 761–784, 2008.

[4] J. Hwang, R.Arkin, and D. Kwon, "Mobile robots at your fingertip: Bezier curve on-line trajectory generation for supervisory control," in *Proceedings of the IEEE/RSJ IROS*, 2003.

[5] R.Andersson, "Agressive trajectory generator for a robot ping-pong player," *IEEE Control Systems Magazine*, vol. 9, no. 2, pp. 15–21, 1989.

[6] S. Calinon, F. Guenter, and A. Billard, "On learning, representing and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 37, no. 2, pp. 286–298, 2007.

[7] A. Ijspeert, J.Nakanishi, and S.Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2002.

[8] S. M. Khansari-zadeh and A. Billard, "Imitation learning of Globally Stable Non-Linear Point-to-Point Robot Motions using Nonlinear Programming," in *Proceeding of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[9] S. M. Khansari Zadeh and A. Billard, "Bm: An iterative method to learn stable non-linear dynamical systems with gaussian mixture models." Proceeding of the International Conference on Robotics and Automation (ICRA 2010), 2010, p. 2381-2388, 2010.

[10] J.Kober, K.Mulling, O.Kromer, C. Lampert, B.Scholkopf, and J.Peters, "Movement templates for learning of hitting and batting," in *Proceeding of the International Conference on Robotics and Automation (ICRA)*, 2010.

[11] S. M. Khansari-zadeh and A. Billard, "Autonomous dynamical system approach to generate human-like robot motions with non-zero velocity at a target." Submitted to International Conference of Robotics and Automation, ICRA, 2011.

[12] B. T. Inc., "Whole arm manipulator." http://www.barrett.com.

[13] W. M. S. Federation. http://www.minigolfsport.com.

[14] M. Ramanantsoa and A. Durey, "Towards a stroke construction model," *International Journal of Table Tennis Science*, vol. 2, pp. 97–114, 1994.

[15] S. Calinon, *Robot Programming by Demonstration: A Probabilistic Approach*. EPFL/CRC Press, 2009. EPFL Press ISBN 978-2-940222-31-5, CRC Press ISBN 978-1-4398-0867-2.

[16] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, "learning movement primitives," in *international symposium on robotics research (isrr2003)*, springer, 2004.

[17] N. M. L. A. P. Dempster and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[18] L.Ljung and T.Glad, *Reglerteori, Flervariabla och olinjara metoder*. Studentlitteratur AB, 1997.

[19] J.Kober, E.Oztop, and J.Peters, "Reinforcement learning to adjust robot movements to new situations," in *Proceesings of Robotics: Science and Systems (RSS)*, 2010.

[20] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[21] B. Schölkopf and A. Smola, *Learning with Kernels*. MIT Press, 1999.

[22] C. E. Rasmussen, "Gpml." Matlab package available at http://www.gaussianprocess.org.

[23] B. Noris, "Mldemos." Educational software for illustrating Machine Learning algorithms, accesible at http://mldemos.b4silio.com/.

[24] R. Yates and D. Goodman, *Probabilty and stochastic processes*. John Wiley and Sons Inc., 2005.

[25] X. community. www.xenomai.org.

[26] C. Smith, N. Beratlis, K. Squires, E. Balaras, and M. Tsunoda, "Direct numerical simulations of the flow around a golf ball: Effect of rotation," in *61st Annual Meeting of the APS Division of Fluid Dynamics*, American Physical Society, 2008.

Appendix A

Original project description

A.1 Learning to Control Planar Hitting Motions of a Robotic Arm in a Mini-Golf-like Task

This project will acquaint the student with the complexity of learning a control law for a multi-degrees of freedom robot from human demonstrations. We consider a task that mimics some of the difficulties one encounters when playing mini-golf. In such a task, hitting the golf ball with the right orientation and speed is crucial and requires years of training. In this project, the student will further improve and implement tools developed in the laboratory for estimating such control laws. Control laws are expressed as non-linear autonomous dynamical systems. Estimation is done through non-linear optimization of Gaussian Mixture Models under stability constraints. The learned model will be implemented both in a dynamic simulator and on a seven degrees of freedom robot arm in a realistic mock-up of a mini-golf terrain. Work will proceed as follow:

1. Teaching a simple point-to-point control law (15%): The student will first get acquainted with the theory, i.e. the statistical tools used for modeling the data and the dynamical systems approach for modeling robot motion. Then, the student will collect a set of demonstrations on the real robot to learn various control strategies to hit a ball with the right orientation and right speed so that the ball will reach the hole. We will first consider flat terrains.

2. Implementation in the robot's simulator (15%): Using an existing dynamic simulator of the robot, a new C++ module should be written such that it enables a user to easily communicate with the simulator to perform different task such as: loading a DS model, executing motions, recording motions, and controlling hitting directions.

3. Learning a model to control the hitting direction (50%): To reach this goal, the student should first determine the minimum necessary input and output

variables that can be used to control the hitting direction in flat and hilly terrains. Then these variables are collected for a set of successful demonstrated putting motions generated in the simulator. Finally, a probabilistic model of the collected data set is learned using two statistical methods: Gaussian Mixture Model and Gaussian Process Regression. The performance of each method is evaluated and compared against each other in terms of generalization of the task to areas not seen during the demonstration and also its robustness to perturbations.

4. Implementation on the robot (20%): In this step, the student extends the C++ code written for the robot's simulator. The new code should be able to communicate with both the stereo vision system and the robot in real-time. Besides, the learned model in step 3 should be slightly modified to match well the real experiment set-up.