
Real Time Probabilistic Models for Robot Trajectories

Probabilistische Echtzeitmodelle für Robotertrajektorien

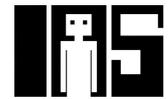
Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)

genehmigte Dissertation von Sebastian Gomez Gonzalez aus Kolumbien

März 2020 — Darmstadt — D 17



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Real Time Probabilistic Models for Robot Trajectories
Probabilistische Echtzeitmodelle für Robotertrajektorien

Genehmigte Dissertation von Sebastian Gomez Gonzalez aus Kolumbien

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prof. Dr. Petar. Kormushev.
3. Gutachten:

Tag der Einreichung: 04.11.2019

Tag der Prüfung: 16.12.2019

Darmstadt – D 17

Please cite this document with:

URN: urn:nbn:de:tuda-tuprints-114926

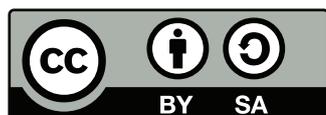
URL: <http://tuprints.ulb.tu-darmstadt.de/id/eprint/11492>

Dieses Dokument wird bereitgestellt von tuprints,

E-Publishing-Service der TU Darmstadt

<http://tuprints.ulb.tu-darmstadt.de>

tuprints@ulb.tu-darmstadt.de



This publication is licensed under the following Creative Commons License:

Attribution-ShareAlike 4.0 International

<http://creativecommons.org/licenses/by-sa/4.0/>

Erklärung zur Dissertation

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Darmstadt, den 6. März 2020

(Sebastian Gomez Gonzalez)

Thesis Statement

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

In the submitted thesis the written copies and the electronic version are identical in content.

Darmstadt, March 6, 2020

(Sebastian Gomez Gonzalez)

Abstract

Robot learning has the potential to give robotic systems the ability to perform multiple tasks and solve difficult tasks in dynamic environments. Probabilistic approaches to robot learning have several properties interesting for robotic applications such as providing uncertainty estimates and likelihood evaluations, useful for decision making and finding atypical environment states where acting might be dangerous for the robot. There are also some typical challenges that robot learning in general and specially probabilistic approaches face for robotics. Real time robot applications such as robot table tennis place strict latency requirements for prediction, likelihood evaluations or other important operators. The amount of data available for learning in robotic applications is also typically not very large, increasing the risks of overfitting specially for probabilistic approaches that usually have more parameters than deterministic methods for the same predictive accuracy. Finally, for certain applications with complex sensors such as computer vision systems it is important to have robot learning methods capable of operating with missing observations and outliers.

In this thesis, we use robot table tennis as an example of a challenging application to propose or extend probabilistic learning approaches for trajectory representations. We place special focus on evaluating the latency of the real time critical operators, trying to ensure safety of the robot to unexpected environment states, operating with missing observations or outliers, and learning with relatively small training sets. Although table tennis is our inspiring application, we propose operators that can be used for other robot applications, trying to keep the table tennis specific heuristics to a minimum.

First we discuss how to learn a robot policy from demonstrations using Probabilistic Movement Primitives. We propose a learning method to learn a movement primitive from a small set of demonstrations performed by a human expert. We compare the proposed learning method with a least squares based method, showing that the least squares method is a special case of the proposed learning algorithm. We also show experimentally that the proposed learning method does not suffer from the overfitting problems of the least squares method and the table tennis hitting and return rate is superior. We also propose adaptation operators in joint and task space for the learned movement primitives, necessary to react to changes in the robot environment such as different incoming ball trajectories or the location of objects like a grinder and brewing chamber for a coffee preparation task. We also present a vision system for real-time object tracking. We focus on reliability of the estimates produced by the vision system, reducing the number of outliers to a minimum, specially as the number of available cameras increases. We use the proposed vision system to track the table tennis ball for robot table tennis with a speed of 180 frames per second. Finally, we introduce a new method for forecasting the future value of a trajectory given its past observations based on variational auto-encoders. We use the proposed model to predict the trajectory of the ball from previous observations of the ball position. The proposed method has a better accuracy for long term predictions than traditional time series forecasting methods such as recurrent neural networks or using differential equations based of physical models, provided that the spin of the ball is not observed by the vision system.



Zusammenfassung

Roboterlernen hat das Potenzial, Robotik-Systemen die Fähigkeit zu verleihen, mehrere Aufgaben unterschiedlicher Schwierigkeit in dynamischen Umgebungen zu lösen. Probabilistische Ansätze für das Roboterlernen haben mehrere interessante Eigenschaften in Robotik-Anwendungen, wie zum Beispiel die Bereitstellung von Unsicherheitsabschätzungen und Wahrscheinlichkeiten, die nützlich für Entscheidungsprozesse sowie das Finden von atypischen Umgebungszuständen sind, in denen es für den Roboter gefährlich sein könnte, zu handeln. Weiterhin gibt es einige typische Herausforderungen im Roboterlernen im Allgemeinen und bei probabilistischen Ansätzen im Speziellen, denen die Robotik begegnen muss. Echtzeit-Robotikanwendungen wie Roboter-Tischtennis bringen strikte Anforderungen an die Latenz von Vorhersagen, der Auswertung von Wahrscheinlichkeiten und anderen wichtigen Operationen mit sich. Die Menge an Daten, die für das Lernen zur Verfügung stehen, ist in Robotik-Anwendungen außerdem für gewöhnlich nicht besonders groß, was die Gefahr von Overfitting erhöht, insbesondere für probabilistische Ansätze, die normalerweise bei gleicher Vorhersagegenauigkeit mehr Parameter aufweisen als deterministische Modelle. Darüber hinaus ist es für bestimmte Anwendungen mit komplexer Sensorik, wie zum Beispiel Computer-Vision-Systemen, wichtig, Lernmethoden einzusetzen, die mit fehlenden Daten sowie Ausreißern umgehen können.

In dieser Arbeit verwenden wir Roboter-Tischtennis als Beispiel einer anspruchsvollen Anwendung, um neue probabilistische Lernansätze zur Repräsentation von Trajektorien vorzuschlagen oder bestehende zu erweitern. Besonderes Augenmerk legen wir dabei auf die Auswertung der Latenz von für den Echtzeit-Ablauf kritischen Operationen, auf die Gewährleistung der Sicherheit des Roboters gegenüber unerwarteten Umgebungszuständen, den Betrieb mit fehlenden Daten oder Ausreißern und auf das Lernen mit verhältnismäßig kleinen Trainingssets. Obwohl Tischtennis die inspirierende Anwendung ist, führen wir Operatoren ein, die auch in anderen Robotik-Projekten angewendet werden können, wobei die für Tischtennis spezifischen Heuristiken auf ein Minimum reduziert werden.

Zunächst besprechen wir, wie eine Policy aus Demonstrationen mit Hilfe von Probabilistic Movement Primitives gelernt werden kann. Wir schlagen eine Lernmethode vor, die es erlaubt, ein Movement Primitive anhand von wenigen Demonstrationen eines menschlichen Experten zu lernen. Diese Lernmethode vergleichen wir mit einem Ansatz, der auf der Methode der kleinsten Quadrate basiert, wobei wir zeigen, dass der Kleinste-Quadrate-Ansatz ein Spezialfall der vorgeschlagenen Methode ist. Auf der Basis von experimentellen Daten wird gezeigt, dass die vorgeschlagene Lernmethode nicht anfällig für die Overfitting-Probleme der Kleinste-Quadrate-Methode ist und dass die Tischtennis-Treffer- und Rückspielraten höher sind. Wir führen außerdem Adaptations-Operatoren im Joint- und Task-Space für die gelernten Movement Primitives ein, welche notwendig für die Reaktion auf Veränderungen in der Umgebung des Roboters wie veränderte Balltrajektorien oder den veränderten Ort von Objekten wie einer Kaffeemühle und -maschine in einer Kaffeezubereitungsaufgabe sind. Darüber hinaus stellen wir ein Vision System für das Echtzeit-Tracking von Objekten vor. Der Schwerpunkt liegt dabei auf der Zuverlässigkeit der resultierenden Schätzwerte für die Objektposition, wobei die Anzahl an Ausreißern auf ein

Minimum reduziert wird, insbesondere bei erhöhter Anzahl von Kameras. Wir verwenden das vorgeschlagene Vision System für das Tracking des Tischtennisballs für das Roboter-Tischtennis mit einer Geschwindigkeit von 180 Bildern pro Sekunde. Schließlich führen wir eine neue Methode für die Vorhersage des zukünftigen Werts einer Trajektorie auf Basis von vergangenen Beobachtungen basierend auf Variational Autoencodern ein. Wir wenden das vorgeschlagene Modell auf die Vorhersage der Balltrajektorie auf Grundlage von aufgezeichneten Ballpositionen an. Die vorgeschlagene Methode hat eine bessere Genauigkeit für Langzeitvorhersagen als gängige Methoden zur Vorhersage von Zeitreihen wie rekurrente neuronale Netze oder Physik-basierte Differenzialgleichungen, vorausgesetzt, dass der Spin des Balls nicht vom Vision System erfasst wird.

Acknowledgments

The work presented on this thesis would not have been possible without the support of a numerous group of people. I want to start thanking my advisor Jan for his guidance and for helping me to keep always on track. I also want to thank all my colleagues and friends at the Max Planck Institute who devoted a long time to discuss ideas with me, discussed ways of testing which ideas were more promising and read my papers providing amazing feedback with which I improved my publications and paper writing skills. Specially, I want to thank Mateo, Diego, Dieter, GB and Paul with whom I had several interesting discussions and from whom I learned a lot. I also want to thank my colleagues at TU Darmstadt for all the useful feedback they provided me, specially after my talks when I was visiting, and for making my visits to Darmstadt more pleasant.

The inspiring and exiting discussions I had during the time I worked on my doctoral studies also helped my to grow professionally and personally, for which I am very grateful. Sports, and specially cycling, was very important for me to keep a healthy body and mind. Without a healthy life style it would not have been possible for me to finish this thesis. Specially, I want to thank Eric, Mateo, Sergey and Diego for introducing me to cycling, train with me and keep me motivated to train hard to reach my goals.

I am most grateful to my parents, who thought me the most important lessons of my life, motivated me to work hard to achieve all my goals, and supported my in every way they could during my studies and my education in general. And last but not least, I thank my loving wife Lucia who complements me in the best possible way and helps me to keep a positive attitude when I face challenging situations.



List of Symbols

The following table denotes the conventions and notation that we used throughout the **thesis**. Where possible, notation is kept consistent with prior work in the area.

Notation

Notation	Description
x	scalar
\mathbf{x}	vector
\mathbf{X}	matrix
\mathbf{X}^\top	transpose of a matrix
\mathbf{X}^{-1}	inverse of a matrix
$p(x)$	probability distribution
$\text{KL}(p(x) q(x))$	Kullback-Leibler divergence
$\mathbb{E}[x]$	Expected value

Symbols

Symbols	Description
\mathbf{y}_t	observation at time t
$\mathbf{y}_{1:t}$	observations from time 1 to time t
\mathbf{q}_t	joint state at time t
τ	Trajectory
T	Duration of a trajectory
D	Dimensionality of the observations \mathbf{y}_t



Figures and Tables

List of Figures

1.1. Two robot setup	4
1.2. Example ball trajectories	5
1.3. Detecting potentially dangerous robot states	7
1.4. Example of a dangerous robot trajectory	8
2.1. Robot executing a forehand ProMP striking movement	13
2.2. ProMP probabilistic graphical model	14
2.3. Hitting movement trajectory samples	16
2.4. Conditioning number of the ProMP covariance matrix	19
2.5. Log-likelihood improvement with each iteration of the EM algorithm	22
2.6. Task space distribution in task space for a hitting movement ProMP	23
2.7. Convergence of the ProMP parameters	29
2.8. Robot preparing coffee	30
2.9. Evaluation pattern for the coffee task	31
2.10. Human demonstrating a forehand hitting movement	33
2.11. Histogram of the success and hitting rates on robot table tennis	36
3.1. Robot table tennis setup	39
3.2. Single shot object detection on table tennis ball	42
3.3. Semantic segmentation on table tennis ball images	44
3.4. Histogram of the error of the vision system	50
4.1. Ball trajectory forecasting example	54
4.2. Encoder and decoder networks	56
4.3. Forecasting error on simulation and the real system	58
4.4. Forecasting error with the number of given observations	61

List of Tables

2.1. Running time of the ProMP conditioning operators	26
2.2. Generalization performance for the coffee task	32
2.3. Performance of the table tennis task	35
3.1. Estimation error and failure probability of the vision system	48
3.2. Latency of the 3D position estimation procedure	49



List of Algorithms

1.	EM Algorithm to train ProMPs	18
2.	EM algorithm for ProMPs with approximated E-step	20
3.	Adapting a ProMP in task space with Laplace approximation	25
4.	Procedure to test the generalization performance of a single ProMP on a pouring coffee experiment	31
5.	Procedure used on the table tennis experiments	34
6.	Finding the set of pixels of an object.	43
7.	Remove outliers by finding the largest consistent subset of 2D observations for stereo vision.	45



Contents

List of Symbols	vii
Figures and Tables	ix
List of Algorithms	xi
1. Introduction	3
1.1. Robot Setup	3
1.2. Probabilistic Trajectory Models	4
1.3. Contributions	7
1.4. Thesis Outline	9
2. Adaptation and Robust Learning of Probabilistic Movement Primitives	13
2.1. Introduction	13
2.2. Robust Learning of Probabilistic Movement Primitives	15
2.3. Adaptation of Probabilistic Movement Primitives	22
2.4. Experiments and Results	28
2.5. Epilogue	36
3. Reliable Real Time Ball Tracking for Robot Table Tennis	39
3.1. Introduction	39
3.2. Reliable Real-Time Ball Tracking	42
3.3. Experiments and Results	47
3.4. Epilogue	50
4. Real Time Trajectory Prediction Using Deep Conditional Generative Models	53
4.1. Introduction	53
4.2. Trajectory Prediction	54
4.3. Deep Conditional Generative Models for Trajectory Forecasting	55
4.4. Experiments	60
4.5. Epilogue	62
5. Conclusions and Future Work	65
5.1. Discussion and Future Work	66
5.2. Publications	67
A. Appendix	77
A.1. Derivation of the ELBO for TVAEs	77
Curriculum Vitae	79



1 Introduction

Autonomous robot systems have been successfully deployed in applications with well-structured environments that do not change drastically, such as manufacturing robots for industry. Most of these robots are designed and programmed to perform one task only, and their environment is controlled to avoid unexpected events. Giving robots the ability to learn is key to achieve autonomous robots capable of performing multiple tasks or operating in more dynamic environments. A robot can learn from a teacher, for example a human expert, using imitation learning techniques. A robot can also learn from its own experience by trial and error using reinforcement learning by trying to optimize a reward function.

Probabilistic approaches to robot learning have the potential to capture the variability present in the robot environment or the demonstrations of the teacher. At prediction time, probabilistic methods can provide uncertainty estimates in addition to mean predictions. The uncertainty estimates are useful for decision making. Finally, probabilistic methods provide methods to evaluate likelihoods. A likelihood evaluation can be used to determine if the current environment of the robot is too different from the training data, in which case the action predicted by the machine learning method might be dangerous to execute.

Examples of probabilistic approaches to robot learning include movement primitive representations based on continuous and smooth basis functions [1] and Hidden Markov models [2], model based reinforcement learning [3], model free reinforcement learning [4], generative models for time series modeling [5, 6] and forecasting [7, 8]. Some of the challenges of applying probabilistic learning methods to robotics include that the training sets are typically not too big, and learning the additional parameters required to represent variability and correlation (or covariance) may result in overfitting or numerical instability [9]. In addition, real time robotic applications require fast predictions, rendering some common probabilistic inference methods like Montecarlo sampling less attractive due to their computational cost.

We use robot table tennis as an example of a challenging robot application with a dynamic environment, suitable to evaluate probabilistic robot learning methods on a real time critical environment. The ball trajectory is changing all the time and it is not easy to predict due to the effect of the spin. The robot needs to react quickly to successfully hit and return the ball. Uncertainty estimates can be used to make decisions that trade off accuracy of the predictions and reaction time. For example, when a few ball observations are available, the uncertainty over the future ball trajectory is higher and it might be a good idea to wait until more observations are available to make better decisions. However, waiting for more observations reduces the reaction time.

1.1 Robot Setup

The main robot tasks we will use on this paper is table tennis. We will also use a coffee preparation task that we describe in detail in Chapter 2, but we used the same robot arm used for table tennis and the same vision system. In this section, we introduce briefly the robot setup used on our robot experiments.



Figure 1.1.: The robot table tennis setup used as an example application. The setup consists of two Barrett WAM arms capable of high speed motion. In the ceiling we have a custom made light system where the intensity can be configured by software. The hardware used for the vision system consists of four RGB cameras connected by Ethernet to a computer with two low cost NVidia GPUs to process about 180 frames per second for each of the four cameras.

Figure 1.1 shows the robot table tennis setup used for the experiments on Chapters 3 and 4. This setup consists of two Barrett WAM robot arms capable of high speed motion. For the vision system we have a custom made light system in which we can configure the light intensity by software. This light system was designed to illuminate the table tennis workspace for both robots as uniformly as possible. We used four RGB Ethernet cameras from Prosillica, capable of providing up to 180 frames per second at VGA resolution. This cameras support using triggering cables, we set one camera as the master to trigger all the other cameras. To ensure that the cameras were taking pictures at the same time, we took pictures of screen displaying time with a resolution up to $\frac{1}{100}$ of a second and ensuring the displayed time in the images of all the cameras was the same.

The four cameras are connected to a computer in charge of running the vision system. In Chapter 3, we explain in detail how the vision system works. The vision computer broadcasts the detected ball positions over the network. Each of the robots is controlled by a separate computer that reads the ball positions sent by the vision computer. To control the robots, the robot control computers receive the joint angles of its robot and provide desired motor torques that we compute using inverse dynamics with a PD controller on top to correct for model errors. The control loop runs at frequency of 500 Hz.

It is important to mention that the described robot setup is the final robot setup obtained by the end of this PhD thesis. The setup used on Chapter 2, for instance, consisted only on one robot and the vision frequency was of 60 ball observations per second instead of 180. But the rest of the details are the same as what is described in this section.

1.2 Probabilistic Trajectory Models

We use the term trajectory is several chapters of this thesis, including the introduction and the thesis title. In this section, we define what we mean by trajectory and several of the operations

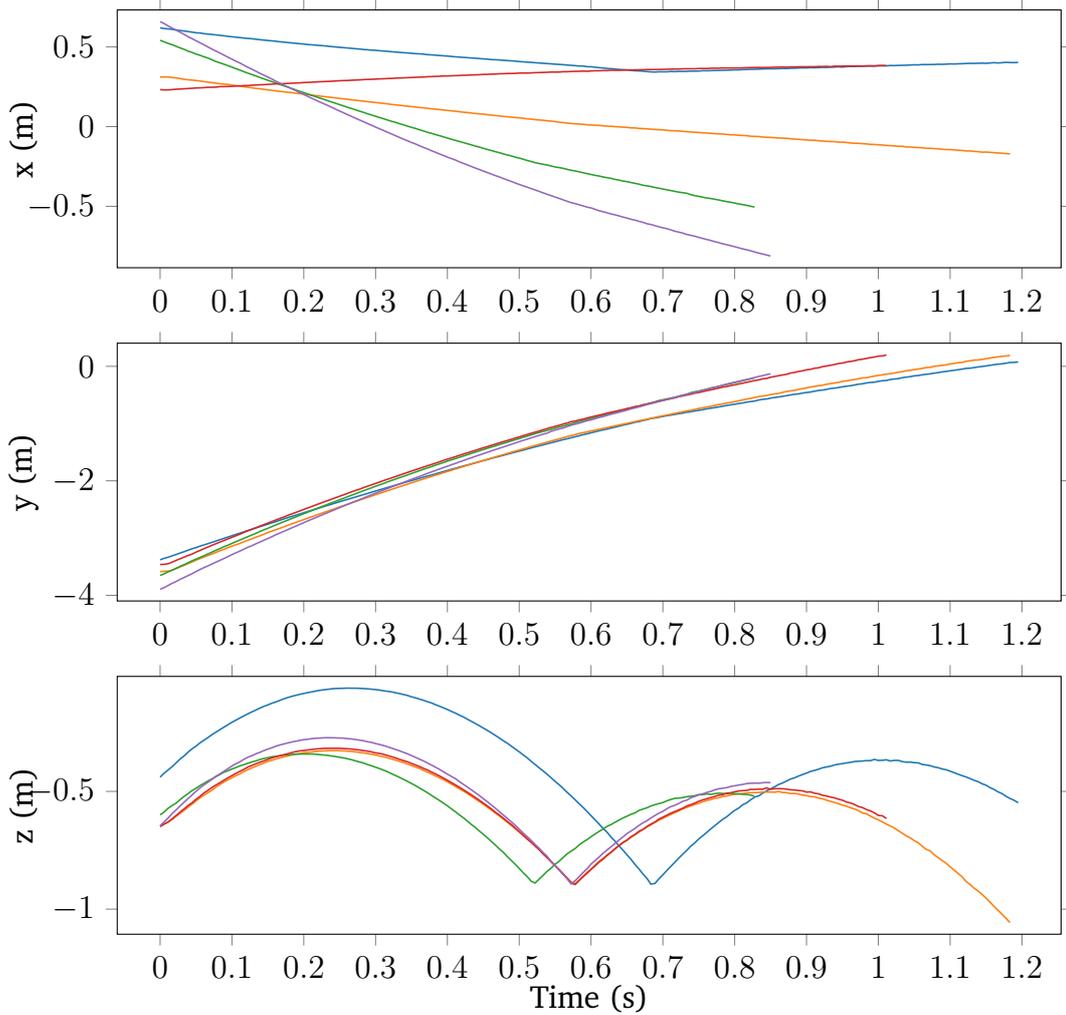


Figure 1.2.: Different samples of table tennis ball trajectories in X, Y and Z (in meters) with respect to time (in seconds). The five ball trajectories depicted in this figure were taken randomly from the ball trajectory dataset collected with the vision system proposed in Chapter 3 to train the ball prediction model proposed in Chapter 4.

we are interested in running with trajectory probability distributions. Each of the upcoming chapters will introduce again the notation we briefly introduce in this section and expand on the operations developed on that particular chapter.

The term trajectory is commonly used in the robotics community to refer to a realization of a time series or Markov decision process. Formally, we define a trajectory $\tau_n = \{\mathbf{y}_{nt}\}_{t=1}^{T_n}$ of total length T_n as a sequence of multiple observations \mathbf{y}_{nt} , where the index t represents time and n indexes the different trajectories in the data set. Informally, a trajectory is a quantity of interest for the robotic application that changes with time, for example, the joint state of a robot or the position of a table tennis ball. Figure 1.2 shows five different ball trajectory samples. In this case, the observation \mathbf{y}_{nt} is a 3 dimensional vector representing the ball position at time index t of the ball trajectory n .

Each trajectory $\tau_n \sim P(\tau)$ is assumed to be independently sampled from the trajectory distribution $P(\tau)$. If $p(\tau)$ represents for example the distribution over ball trajectories, Figure 1.2 is showing five independent samples from the distribution $p(\tau)$.

1.2.1 Learning a Trajectory Distribution

One of the problems we address in this thesis is how to learn the trajectory distribution from a relatively low data set of collected trajectories. Data efficiency is important for robot applications, specially when human supervision is required for the safety of the robot or to collect the data itself. An example of the data collection from human demonstrations we use in this thesis is when we collect hitting movement from a human teacher moving the robot arm, and subsequently learn a distribution over this hitting trajectories as a probabilistic movement primitive.

1.2.2 Computing Conditional Distributions

A very important operator that is also used several times on this thesis is computing and sampling a conditional distribution over trajectories. For example, we might be interested in starting a robot joint trajectory in some particular joint state, or reach a particular task space configuration at some specified point in time like for example the predicted position of the table tennis ball.

Another use for conditional distributions is for trajectory forecasting, when we condition the trajectory distribution in all the previous observations and are interested to predict the future values of the trajectory. For example, predict the future trajectory of the table tennis ball given the past observations reported by the vision system. Assuming that the current time is represented by t , trajectory forecasting consists on evaluating the following conditional distribution

$$p(\mathbf{y}_t, \dots, \mathbf{y}_T \mid \mathbf{y}_1, \dots, \mathbf{y}_{t-1}).$$

Note that an important characteristic of all the use cases previously described for computing conditional distributions is that they must meet real-time latency constraints. The ball prediction is not useful if it takes more time to compute it than the ball flying time minus the robot reaction time.

1.2.3 Evaluating Likelihoods

Evaluating the Likelihood of a particular trajectory τ under the learned trajectory distribution $p(\tau)$ is useful for robot safety. Suppose for instance that the table tennis robot manages to hit a ball, and the ball subsequently hits the net and rolls back towards to robot on the table. Attempting to hit such a ball will result in hitting the table, risking the physical integrity of the robot. Moreover, if a machine learning model is used in general on an input point very different to the inputs used on the training set, the prediction is likely to be wrong. If the prediction include robot actions, these actions are likely to be dangerous to execute. Figure 1.3 illustrates the described scenario with a simple robot with one degree of freedom. The robot state is the joint position and velocity pair (q, \dot{q}) . The test point, depicted in blue, is far from the training data. Using a learned policy or robot controller on that particular robot state may have unexpected or dangerous behavior.

Evaluating likelihoods is useful to detect when an observation is very different to the observations obtained on the training set. We used a simple heuristic for safety in our robot table tennis experiments: If the likelihood of the observed ball trajectory is lower than a certain threshold,

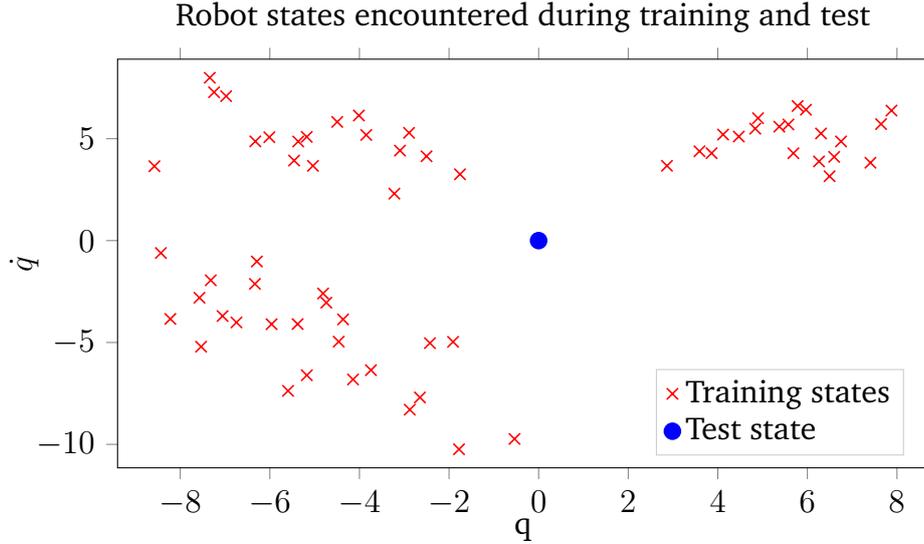


Figure 1.3.: Illustration of a scenario where a probabilistic method can be used to detect a dangerous state in which to execute a robot policy. The figure shows a set of states discovered at training time and a state discovered at test time that is far away from the training data. In such a case, it might be dangerous to execute the learned policy. A probabilistic method could be used to detect such scenarios using the likelihood function. If the likelihood of the test state is lower than a certain threshold, the state is considered too different from the training data and the robot is frozen into its current position instead of using the learned policy.

we do not move the robot to respond such a ball. We found that this simple heuristic was able to filter out the dangerous situations very effectively. Instead of attempting to anticipate all the possible things that could go wrong, we simply avoid reacting to any environment state that is too different from what is expected on a normal game. Figure 1.4 shows an example of a dangerous robot trajectory executed by the table tennis robot using the Mixture of Movement Primitives [10] policy. This policy uses deterministic machine learning methods, and its implementation includes several safety checks. Nevertheless, during one of the trials the robot still executed a trajectory with high velocities and accelerations that resulted on part of the robot cover falling off. During all the experiments performed on this thesis, no single dangerous trajectory was executed by the robot using the simple likelihood threshold safety check.

1.3 Contributions

This thesis addresses probabilistic learning approaches over trajectories using a robot table tennis platform as a guide application. Real time performance and uncertainty or variability quantification are some of the key features recurrent to our approach for robot table tennis. In this section, we summarize the main contributions of this thesis.

1.3.1 Robust Learning of Probabilistic Movement Primitives

A Probabilistic Movement Primitive (ProMP) is a probabilistic representation framework for robot trajectories used to learn motor behaviour from a human teacher performing demonstrations. An advantage of using a probabilistic approach to represent movement primitives, is that it captures the variability of the teacher instead of capturing only the mean behaviour. A ProMP can, in addition, capture the correlation between the different joints of the robot on the demonstrated behaviour.

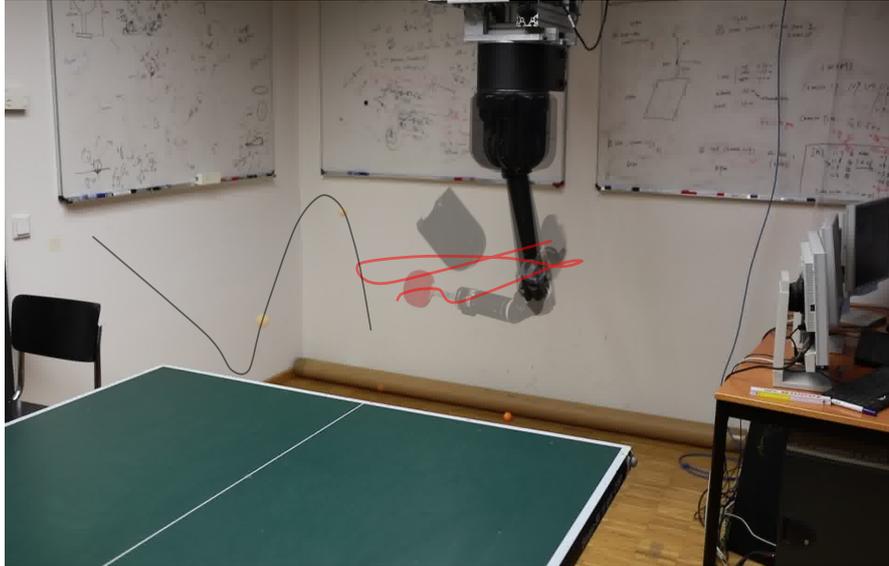


Figure 1.4.: Example of a dangerous robot trajectory executed by the table tennis robot arm produced by a table tennis policy called Mixture of Movement Primitives [10]. During the trial depicted in this figure, the robot reached very high joint velocities and accelerations despite the implemented safety checks. During all the experiments performed on this thesis, no single dangerous trajectory was executed by the robot using the simple likelihood threshold safety check.

However, in order to capture the variability of the teacher and the correlation between the joints of the robot, the ProMP framework uses more parameters than its deterministic counterparts. As a result, traditional learning methods that rely on maximum likelihood estimation result in numerical instability and overfitting, specially when the available amount of training data is low.

We propose a method to learn the parameters of a ProMP using prior distributions over the parameters of the ProMP, and maximizing the posterior distribution instead of maximizing the likelihood. The optimization of the posterior is performed using an Expectation Maximization algorithm, with closed form update equations. We compare our learning algorithm with previous work, showing that our algorithm is more general and the resulting ProMP parameters are numerically stable.

1.3.2 Adaptation Operators for Probabilistic Movement Primitives

Some robot applications will require to adapt the learned ProMPs to the state environment before the execution of the primitive. For example, in table tennis the striking movement needs to be adapted to intercept the trajectory of the ball, that is different every time.

We propose a method to adapt a ProMP to have a desired task space configuration. We also propose methods to adapt the ProMP in joint space that are slightly different to the methods originally proposed in [1], taking advantage of the numerical stability obtained with our learning method. We evaluate the proposed adaptation operators in a robot table tennis and a coffee pouring task.

1.3.3 Reliable Real Time Ball Tracking for Robot Table Tennis

One of the main robot applications discussed in this thesis is robot table tennis. Table tennis is highly dynamic task difficult to play for robots and humans. To successfully play table tennis,

the robot requires a vision system that provides reliable ball observations with low latencies. We proposed a ball tracking system that focuses on reliability, using real time machine learning methods to track the position of the ball on each image, and subsequently producing a single 3D ball position detecting and rejecting errors produced by the object detection method. To detect errors we used an outlier rejection by consensus method. We evaluate the proposed vision system accuracy and latency. We show that our method outperforms the vision system proposed in [11] in both average accuracy and resilience to outliers. We furthermore show in simulation that adding more cameras to the table tennis setup would result in a higher accuracy and outlier detection rates.

1.3.4 Trajectory Variational Auto-Encoders

We propose a novel method for trajectory forecasting based on variational auto-encoders. We show that the proposed method has higher accuracy for long term trajectory prediction than other traditional machine learning methods for time series forecasting such as LSTM neural networks, because our approach does not suffer from the cumulative error problems of recurrent networks that use their own predictions to make predictions farther into the future. The proposed approach provides the real time performance requirements for applications such as robot table tennis. In addition, like other variational auto-encoder approaches, our method provides uncertainty estimates about the predicted trajectory. We evaluate the proposed method by predicting the table tennis ball trajectory based on the 3D positions reported by the vision system. The spin is not observed, but the proposed method can still predict the ball trajectory with the required accuracy for table tennis, suggesting that the method can partially infer spin from the observed ball trajectory itself.

1.4 Thesis Outline

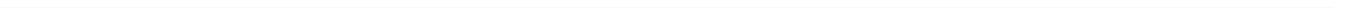
The chapters of this thesis are structured in the chronological order in which each of these research projects was conducted. This ordering should simplify understanding in detail the experiments of the later chapters that use the techniques introduced earlier in the thesis. Nonetheless, each of the chapters is relatively self contained.

Chapter 2 introduces a learning method and adaptation operators for probabilistic movement primitives. The movement primitive is learned from human demonstrations. We use the learned primitives and the adaptation operator for the robot table tennis task and for a coffee preparation task.

We use [11] as the vision system to track table tennis balls at a frequency of 60 Hertz, and we discover that this vision system produces outliers very often, requiring additional heuristics to reject outliers based on physics models. To deal with a higher variability in the ball trajectory, we needed a faster and more reliable vision system.

In Chapter 3, we propose a vision system for real time object tracking using modern machine learning techniques of object detection. Unlike [11], we assume that the object detection techniques will detect the wrong object some times. We use an outlier detection by consensus technique to produce a single 3D position from multiple cameras while being robust to outliers. Chapter 4 introduces a novel method for trajectory forecasting based on variational auto-encoders. The uncertainty about the future trajectory is captured with the latent variable.

We show that the proposed model is more accurate for long term prediction than recursive methods like recurrent neural networks. We evaluate the proposed method in the table tennis setup by predicting the trajectory of the ball without directly observing spin.





2 Adaptation and Robust Learning of Probabilistic Movement Primitives

Probabilistic representations of movement primitives open important new possibilities for machine learning in robotics. These representations are able to capture the variability of the demonstrations from a teacher as a probability distribution over trajectories, providing a sensible region of exploration and the ability to adapt to changes in the robot environment. However, to be able to capture variability and correlations between different joints, a probabilistic movement primitive requires the estimation of a larger number of parameters compared to their deterministic counterparts, that focus on modeling only the mean behavior.

In this chapter, we make use of prior distributions over the parameters of a probabilistic movement primitive to make robust estimates of the parameters with few training instances. In addition, we introduce general purpose operators to adapt movement primitives in joint and task space. The proposed training method and adaptation operators are tested in a coffee preparation and in robot table tennis task. In the coffee preparation task we evaluate the generalization performance to changes in the location of the coffee grinder and brewing chamber in a target area, achieving the desired behavior after only two demonstrations. In the table tennis task we evaluate the hit and return rates, outperforming previous approaches while using fewer task specific heuristics.

2.1 Introduction

Techniques that can learn motor behavior from human demonstrations and reproduce the learned behavior in a robotic system have the potential to generalize better to different tasks. Multiple models have been proposed to represent complex behavior as a sequence of simpler movements typically known as movement primitives. A movement primitive framework should provide operators to learn primitives from demonstrations, adapt them to achieve different goals and execute them in a sequence on a robotic system.

Deterministic Movement Primitive frameworks have been used successfully for a variety of robotic tasks including locomotion [12], grasping [13], ball in a cup [14] and pancake flipping [15]. However, deterministic representations capture only the

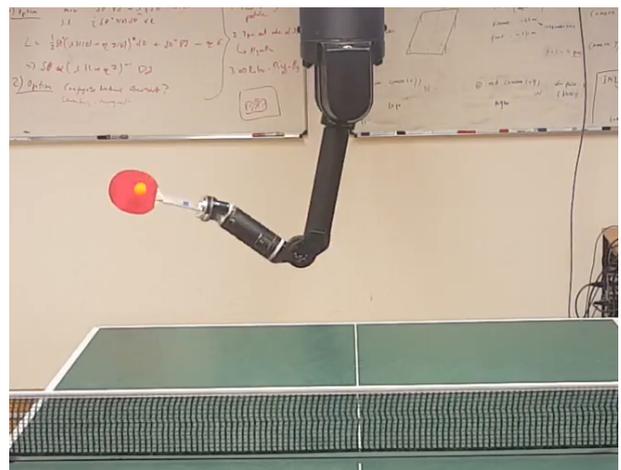


Figure 2.1.: Robot table tennis setup used to evaluate the proposed methods. The ball is tracked using four cameras attached to the ceiling. The robot arm is a Barrett WAM capable of high speed motion, with seven degrees of freedom like a human arm.

mean behavior of the demonstrations of the teacher. The variability in the demonstrations is not captured nor used.

In biological systems, variability seems to be characteristic of all behavior, even in the most skilled and seemingly automated performance [16]. Thus, a movement primitive representation that captures variance in the demonstrated behavior has the potential to model the human teacher better. For a task like table tennis, the variability of the teacher is partially a response to the changes in ball trajectory. Therefore, approaches that capture it have the potential to adapt better to diverse ball trajectories. At the same time, the variability of the teacher can be used to define a region of sensible exploration for a robotic system.

Probabilistic approaches can naturally capture variability using a probability distribution. Some probabilistic representations of movement primitives focus on learning a distribution over demonstrated states using Gaussian Mixture models or Hidden Markov models [2][17]. Subsequently using the log-likelihood as cost function to reproduce the learned movement using an optimal control method [18][19].

Other probabilistic representations focus on learning a distribution over robot trajectories directly. Some approaches represent trajectories as functions of time and the distribution over these trajectories using parametric [1] or non-parametric [5] approaches. The trajectories can also be represented with recursive probability distributions, using latent state space models [6].

In this chapter, we build on top of a probabilistic representation introduced in [1] called Probabilistic Movement Primitives (ProMPs). In this probabilistic formulation of movement primitives, a movement primitive is represented as a probability distribution over robot trajectories. Different realizations of the same movement primitive are assumed to be independent samples from the distribution over trajectories.

ProMPs have typically more parameters than non-probabilistic representations. These extra parameters are used to capture the variability of the movements executed by the teacher and the correlations between different degrees of freedom of the robot. We propose using prior distributions over the ProMP parameters to make robust estimates with few demonstrations. The influence of the prior distribution decreases as more training data becomes available, converging to the maximum likelihood estimates.

This chapter also presents general purpose operators to adapt a ProMP to have a desired joint or task space configuration at a certain time. By joint space we refer to the joint angles and velocities of the robot, and by task space we refer to the world coordinate position and velocity of the end effector of the robot.

The proposed method to learn the movement primitive and the operators to adapt the movement primitives in task and joint space are evaluated with synthetic data, in a robot table tennis task and a robot assisted coffee preparation task. Figure 2.1 shows the robot table tennis setup used in the experiments. The results obtained with the presented method are compared with previous work on robot table tennis. The proposed approach outperforms previous robot table tennis

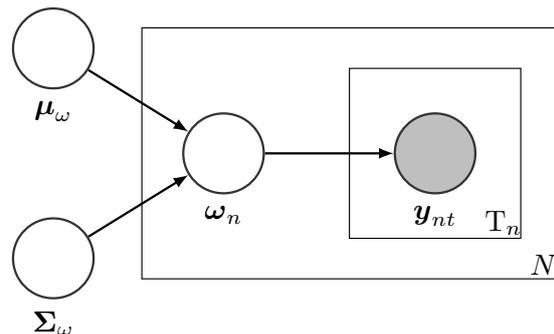


Figure 2.2.: Probabilistic Movement Primitive graphical model for a ProMP. The joint state \mathbf{y}_{nt} is generated from the compact representation of a trajectory ω_n . The mean behavior of the different trajectories is represented by the variable μ_ω , and the variability of the teacher is represented by Σ_ω .

approaches using less task specific heuristics. Examples of task specific heuristics used for robot table tennis in previous approaches include using a Virtual Hitting Planes [20] and computing optimal racket velocity and orientation at hitting time to send balls to the opponent side of the table [10]. The presented approach does not compute racket orientations or velocities to return balls to the opponent’s court. The training data used to learn the movement primitives was built using only successful human demonstrations. The robot was able to learn the behavior required to successfully return balls to the opponent side of the table from the human demonstrations. We use the pouring coffee task to evaluate the generalization performance of the presented method as a function of the number of training instances by changing the position of the coffee grinder and the brewing chamber. The robot manages to pour successfully on the selected testing area after two training demonstrations, suggesting that the presented prior is a sensible choice for this task. Finally, the fact that the presented approach can be used for two robot tasks as different as table tennis and coffee pouring without any changes suggests it has the potential to perform well in several other robot applications.

2.2 Robust Learning of Probabilistic Movement Primitives

Probabilistic movement primitives (ProMPs) are probability distributions used to represent motion trajectories [1]. A trajectory $\tau = \{\mathbf{y}_t\}_{t=1}^T$, can be represented as positions or joint angles at different moments in time. We assume that \mathbf{y}_t is a D dimensional vector that represents the joint measurement at time t of a robotic system with D degrees of freedom.

First, let us introduce a variable $\boldsymbol{\omega} = [\boldsymbol{\omega}_1^\top, \dots, \boldsymbol{\omega}_D^\top]^\top$ that encodes compactly a single robot trajectory, and consists of the concatenation of D weight vectors $\boldsymbol{\omega}_d$ that represent the trajectory of each of the degrees of freedom of the robot, indexed by d .

Given a trajectory realization represented by $\boldsymbol{\omega}$, the joint state at time t is computed as

$$\mathbf{y}_t = [\phi_1(t)^\top \boldsymbol{\omega}_1, \dots, \phi_D(t)^\top \boldsymbol{\omega}_D]^\top + \boldsymbol{\epsilon}_y,$$

where the vector $\phi_d(t)$ is a computed from a set of time dependent basis functions, and $\boldsymbol{\epsilon}_y$ is Gaussian white noise. To obtain smooth trajectories, the basis functions need to be smooth. We use radial basis functions (RBF), polynomial basis functions and a combination of both. The number and type of basis functions to use is a design choice. Each degree of freedom could have a different number of basis functions, but for simplicity we assume every degree of freedom uses K basis functions.

The distribution over the values of the joint state at time t , can be written as

$$p(\mathbf{y}_t | \boldsymbol{\omega}) = \mathcal{N}(\mathbf{y}_t | \boldsymbol{\Phi}_t \boldsymbol{\omega}, \boldsymbol{\Sigma}_y), \quad (2.1)$$

where $\boldsymbol{\Phi}_t$ is a $D \times KD$ matrix used to write the distribution over \mathbf{y}_t in vectorized form, and is defined as

$$\boldsymbol{\Phi}_t = \begin{pmatrix} \phi_1(t) & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \phi_D(t) \end{pmatrix}.$$

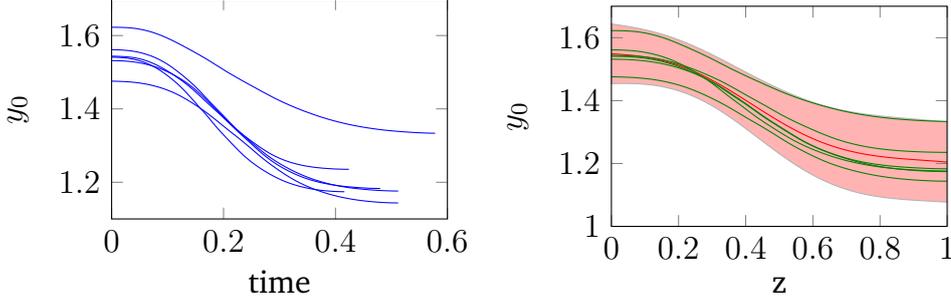


Figure 2.3.: Demonstrated trajectories and learned distribution for the first degree of freedom of Barrett WAM robot arm. The joint value y_0 corresponds to the shoulder yaw recorded in radians as a function of time. Different trajectories of a table tennis forehand motion are demonstrated by the human teacher. These trajectories are depicted in blue and have different durations. The duration of each trajectory is normalized to one to achieve duration invariance, the time invariant trajectories are depicted in green. The learned distribution is depicted in red. The shaded area corresponds to two standard deviations. Note that the model captures the mean behavior and the variability of the teacher at different points in time.

Different realizations of a movement primitive are assumed to have different values for ω . In this model, a particular realization n represented by ω_n is assumed to be sampled from

$$p(\omega_n | \theta_\omega) = \mathcal{N}(\omega_n | \mu_\omega, \Sigma_\omega), \quad (2.2)$$

where $\theta_\omega = \{\mu_\omega, \Sigma_\omega\}$ is a set of parameters that capture the similarities and differences of different realizations of the movement primitive. In the rest of this section, we drop the index n from ω_n for notational simplicity.

Let us write the distribution $p(\omega | \theta_\omega)$ decomposing the $KD \times 1$ vector μ_ω and the $KD \times KD$ matrix Σ_ω in the components corresponding to each degree of freedom,

$$\mathcal{N} \left(\left(\begin{array}{c} \omega_1 \\ \vdots \\ \omega_D \end{array} \right) \middle| \left(\begin{array}{c} \mu_\omega^1 \\ \vdots \\ \mu_\omega^D \end{array} \right), \left(\begin{array}{ccc} \Sigma_\omega^{(1,1)} & \dots & \Sigma_\omega^{(1,D)} \\ \vdots & \ddots & \vdots \\ \Sigma_\omega^{(D,1)} & \dots & \Sigma_\omega^{(D,D)} \end{array} \right) \right).$$

Note that the mean behavior for the degree of freedom d is captured by the $K \times 1$ vector μ_ω^d and the variability by the $K \times K$ matrix $\Sigma_\omega^{(d,d)}$. The correlation between two different joints d_1 and d_2 is captured by $\Sigma_\omega^{(d_1,d_2)}$. The model can be forced to consider all the joints independently by forcing the matrix Σ_ω to be block diagonal.

A probabilistic graphical model is a probabilistic model for which a graph expresses conditional independence assumptions between random variables [21]. Figure 2.2 shows the graphical representation of the probabilistic model used to represent movement primitives. To sample a robot trajectory given the ProMP parameters μ_ω and Σ_ω , a vector ω_n is sampled using (2.2). Subsequently, the new trajectory of length T_n can be sampled using (2.1). If the used basis functions are smooth, the sampled trajectories will also be smooth.

Figure 2.3 show six human demonstrations of a forehand table tennis striking movement and the learned probability distribution. The figure shows the value in radians of the shoulder yaw y_0 with respect to time. The original demonstrations given by the human teacher, depicted in blue, have different durations varying between 0.4 and 0.6 seconds. The time of every demonstration is normalized to be between zero and one to achieve duration invariance using a new variable $z = \frac{t-t_0}{T}$, known as the phase variable [1]. The same demonstrations with respect to the phase variable are depicted in green, and the learned distribution is depicted in

red. The shaded area corresponds to two standard deviations. The ProMP learned from the given demonstrations capture the mean behavior and the teacher variability in different points of time.

2.2.1 Learning from Demonstrations

The parameters θ_ω can be learned from human demonstrations. Let us assume we have N recorded human demonstrations, and extend the notation with an extra sub-index $n \in \{1..N\}$ identifying each demonstration. Thus, the variables \mathbf{y}_{nt} and ω_n represent the joint state of the n th trial at time t and the compact representation of the n th trial respectively. The likelihood of the recorded data is given by

$$p(\mathbf{Y}|\theta_\omega) = \prod_{n=1}^N \int p(\omega_n | \theta_\omega) \prod_{t=1}^{T_n} p(\mathbf{y}_{nt} | \omega_n) d\omega_n,$$

where \mathbf{Y} is the set of values \mathbf{y}_{nt} for all the training instances. Note that evaluating the likelihood requires the computation of an integral over the hidden variables ω_n . Although the integral in this case can be computed in closed form, evaluating the resulting expression would cost cubic time over the trajectory lengths T_n . Instead, we propose to use the expectation maximization algorithm to optimize the likelihood or posterior distribution with linear time costs over the trajectory lengths T_n .

In [22], the ProMP parameters are estimated by first making a point estimate of the hidden variables with least squares, and subsequently finding their empirical mean and covariance matrix as the ProMP parameters. This estimation procedure makes intuitive sense and avoids computing integrals. However, the authors did not provide a mathematical intuition of how their estimation procedure relates to maximizing the marginal likelihood. In the Appendix 2.2.3, we explain in detail the estimation method introduced in [22], and show that it is a special case of an approximation of the proposed EM algorithm to maximize the likelihood. The approximation consists of performing a single EM iteration and approximating the Gaussian distribution computed in the E-step with a Dirac delta distribution, ignoring the uncertainty over the estimates of the hidden variables.

In previous work [23], the parameters were learned maximizing the likelihood. However, maximizing the likelihood results in numerically unstable estimates for the parameters of the ProMP unless a very large number of demonstrations is available. In [23], the matrix Σ_ω is forced to be block diagonal to deal with the numerical problems. As a result, the ProMP parameters could be robustly estimated, but the model becomes incapable of learning the correlation between the different joints of the robot arm.

We use regularization to estimate the ProMP parameters in the form of a prior probability distribution $p(\theta_\omega)$. The posterior distribution over the ProMP parameters is given by

$$p(\theta_\omega|\mathbf{Y}) \propto p(\theta_\omega)p(\mathbf{Y}|\theta_\omega). \quad (2.3)$$

We estimate the parameters θ_ω by maximizing the posterior distribution of (2.3) using the expectation maximization algorithm. This estimator is commonly known as Maximum A Posteriori (MAP) estimate.

The pseudo-code summarizing the training procedure is presented in Algorithm 1. Lines 6 and 7 correspond to the E-step and lines 9 to 13 correspond to the M-step. The values $\epsilon_{nt} = \mathbf{y}_{nt} - \phi_{nt}\bar{\mathbf{w}}_n$ are the residuals used to estimate the sensor noise.

Algorithm 1 Expectation Maximization algorithm to train a ProMP from demonstrations

Input: Demonstration dataset containing the joint states and corresponding normalized time stamps $\mathbf{Y} = \{\mathbf{y}_{nt}, z_{nt}\}$ and the prior parameters $k_0, \mathbf{m}_0, v_0, \mathbf{S}_0$

Output: The ProMP parameters $\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega, \boldsymbol{\Sigma}_y$

- 1: Compute matrices $\boldsymbol{\phi}_{nt} = \boldsymbol{\phi}(z_{nt})$ with the basis functions $\boldsymbol{\phi}$
 - 2: Compute $L = \sum_{n=1}^N \tau_n$
 - 3: Set some initial values for $\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega, \boldsymbol{\Sigma}_y$. We use $\boldsymbol{\mu}_\omega = \mathbf{0}, \boldsymbol{\Sigma}_\omega = \mathbf{I}$ and $\boldsymbol{\Sigma}_y = \mathbf{I}$.
 - 4: **while** Not converged **do**
 - 5: **for** $n \in \{1, \dots, N\}$ **do**
 - 6: $\mathbf{S}_\omega^n \leftarrow (\boldsymbol{\Sigma}_\omega^{-1} + \sum_{t=1}^{\tau_n} \boldsymbol{\phi}_{nt}^\top \boldsymbol{\Sigma}_y^{-1} \boldsymbol{\phi}_{nt})^{-1}$
 - 7: $\bar{\mathbf{w}}_n \leftarrow \mathbf{S}_\omega^n (\boldsymbol{\Sigma}_\omega^{-1} \boldsymbol{\mu}_\omega + \sum_{t=1}^{\tau_n} \boldsymbol{\phi}_{nt}^\top \boldsymbol{\Sigma}_y^{-1} \mathbf{y}_{nt})$
 - 8: **end for**
 - 9: $\boldsymbol{\mu}_\omega^* \leftarrow \frac{1}{N} \left(\sum_{n=1}^N \bar{\mathbf{w}}_n \right)$
 - 10: $\boldsymbol{\mu}_\omega \leftarrow \frac{1}{N+k_0} (k_0 \mathbf{m}_0 + N \boldsymbol{\mu}_\omega^*)$
 - 11: $\boldsymbol{\Sigma}_\omega^* \leftarrow \frac{1}{N} \sum_{n=1}^N \left(\mathbf{S}_\omega^n + (\bar{\mathbf{w}}_n - \boldsymbol{\mu}_\omega)(\bar{\mathbf{w}}_n - \boldsymbol{\mu}_\omega)^\top \right)$
 - 12: $\boldsymbol{\Sigma}_\omega \leftarrow \frac{1}{N+v_0+KD+1} [\mathbf{S}_0 + N \boldsymbol{\Sigma}_\omega^*]$
 - 13: $\boldsymbol{\Sigma}_y \leftarrow \frac{1}{L} \sum_{n=1}^N \sum_{t=1}^{\tau_n} [\boldsymbol{\epsilon}_{nt} \boldsymbol{\epsilon}_{nt}^\top + \boldsymbol{\phi}_{nt} \mathbf{S}_\omega^n \boldsymbol{\phi}_{nt}^\top]$
 - 14: **end while**
 - 15: **return** $\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega$ and $\boldsymbol{\Sigma}_y$.
-

2.2.2 Prior Distribution

We use a Normal-Inverse-Wishart as a prior distribution over the ProMP parameters $\boldsymbol{\mu}_\omega$ and $\boldsymbol{\Sigma}_\omega$, given by

$$\begin{aligned} p(\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega) &= \text{NIW}(\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega \mid k_0, \mathbf{m}_0, v_0, \mathbf{S}_0) \\ &= \mathcal{N}\left(\boldsymbol{\mu}_\omega \mid \mathbf{m}_0, \frac{1}{k_0} \boldsymbol{\Sigma}_\omega\right) \mathcal{W}^{-1}(\boldsymbol{\Sigma}_\omega \mid v_0, \mathbf{S}_0), \end{aligned}$$

where $\mathcal{W}^{-1}(\boldsymbol{\Sigma}_\omega \mid v_0, \mathbf{S}_0)$ is an inverse Wishart distribution, used frequently as a prior for covariance matrices. The main reason why we decided to use a Normal-Inverse-Wishart prior for the ProMP model is because it is a conjugate prior, resulting in closed form updates for the parameters in the EM algorithm and simplifying the inference process. Furthermore, the parameters of this prior distribution have a simple interpretation. Lines 9 and 11 compute the Maximum Likelihood estimates (MLE) $\boldsymbol{\mu}_\omega^*$ and $\boldsymbol{\Sigma}_\omega^*$. Lines 10 and 12 compute the MAP estimates $\boldsymbol{\mu}_\omega$ and $\boldsymbol{\Sigma}_\omega$. Note that the MAP estimates are a weighted average of the MLE estimates $\boldsymbol{\mu}_\omega^*$ and $\boldsymbol{\Sigma}_\omega^*$ and the assumed prior parameters for the mean \mathbf{m}_0 and covariance \mathbf{S}_0 respectively. In the limit of infinite data, the MAP estimates converge to the MLE estimates.

We use a non informative prior for $\boldsymbol{\mu}_\omega$ in our experiments by setting $k_0 = 0$. Note that by setting $k_0 = 0$, the MAP estimate $\boldsymbol{\mu}_\omega$ becomes the MLE estimate $\boldsymbol{\mu}_\omega^*$. If a large number of basis functions is used, a sensible choice for the prior parameters is to use $\mathbf{m}_0 = \mathbf{0}$ and $k_0 > 0$. Such a prior will prevent large values on the estimated vector $\boldsymbol{\mu}_\omega$, similar to the regularization used in Ridge Regression.

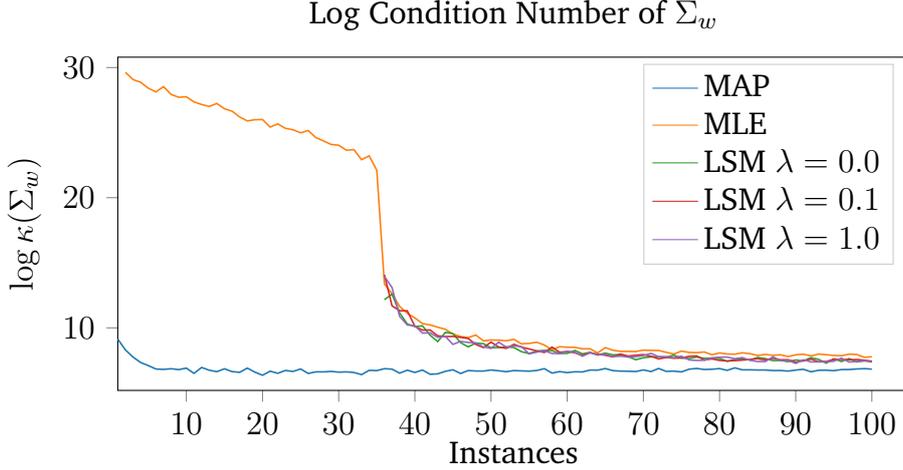


Figure 2.4.: Conditioning number of the covariance matrix Σ_w obtained with multiple learning algorithms. Intuitively, a lower matrix condition number for Σ_w translates into more robustness and numerical stability. The conditioning number is presented in logarithmic scale. Note that the condition number stabilizes around 6 demonstrations for Maximum A Posteriori (MAP), whereas Maximum Likelihood (MLE) and the Least Squares method (LSM) with different regularization values λ requires around 50 demonstrations. In consequence, to avoid numerical problems the Prior distributions should be used unless a very large amount of data is available.

For Σ_ω we use an informative prior. Intuitively, the parameter v_0 of the inverse Wishart prior represents how confident we are about our initial guess of the value of Σ_ω before looking at the data. We use $v_0 = \dim(\omega) + 1$, that is the minimum value for v_0 that results in a proper prior distribution [21]. We set the prior parameter S_0 as

$$S_0 = (v_0 + KD + 1) \text{blockdiag}(\Sigma_\omega^*), \quad (2.4)$$

where Σ_ω^* is the maximum likelihood estimate of Σ_ω computed in line 11 of Algorithm 1. Intuitively, the prior distribution favors considering joints independent when few data is available and gradually learn the correlation of different joints as more data is obtained. Using (2.4), the update equation for Σ_ω on line 12 of Algorithm 1 can be written as

$$\Sigma_\omega = \frac{1}{N + N_0} [N_0 \text{blockdiag}(\Sigma_\omega^*) + N \Sigma_\omega^*],$$

with $N_0 = v_0 + KD + 1$. Note that the MAP estimate of Σ_ω is a linear combination of the full MLE estimate and the MLE estimate under the assumption that all joints are independent. With a large number of trials N , the MAP estimate will converge to the MLE estimate as expected.

One of the reasons why we recommend using an informative prior for Σ_ω , is because its MLE estimate is typically numerically unstable. We used the matrix condition number $\kappa(\Sigma_\omega)$ to measure numerical stability of the Maximum A Posteriori (MAP) and Maximum Likelihood Estimator (MLE) estimates of Σ_ω . Intuitively, the condition number provides a measure of the sensitivity of an estimated value to small changes in the input data [24]. Therefore, a smaller the condition number means a more numerically stable estimate. Figure 2.4 shows the change in the condition number for Σ_ω in logarithmic scale with respect to the number of training instances for both the MAP and MLE estimates. The condition number for the MAP estimate is depicted in blue, and stabilizes around 6 training instances. On the other hand, the MLE estimate depicted in red requires around 50 training instances to stabilize.

Algorithm 2 EM training algorithm with a Dirac delta approximation for the E-step

Input: Demonstration dataset containing the joint states and corresponding normalized time stamps $\mathbf{Y} = \{\mathbf{y}_{nt}, z_{nt}\}$ and the prior parameters $k_0, \mathbf{m}_0, v_0, \mathbf{S}_0$

Output: The ProMP parameters $\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega, \boldsymbol{\Sigma}_y$

- 1: Compute matrices $\phi_{nt} = \phi(z_{nt})$ with the basis functions ϕ
 - 2: Compute $L = \sum_{n=1}^N \tau_n$
 - 3: Set some initial values for $\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega, \boldsymbol{\Sigma}_y$. We use $\boldsymbol{\mu}_\omega = \mathbf{0}, \boldsymbol{\Sigma}_\omega = \mathbf{I}$ and $\boldsymbol{\Sigma}_y = \mathbf{I}$.
 - 4: **while** Not converged **do**
 - 5: **for** $n \in \{1, \dots, N\}$ **do**
 - 6: Compute $\hat{\boldsymbol{\omega}}_n$ with (2.8)
 - 7: **end for**
 - 8: $\boldsymbol{\mu}_\omega^* \leftarrow \frac{1}{N} \left(\sum_{n=1}^N \hat{\boldsymbol{\omega}}_n \right)$
 - 9: $\boldsymbol{\mu}_\omega \leftarrow \frac{1}{N+k_0} (k_0 \mathbf{m}_0 + N \boldsymbol{\mu}_\omega^*)$
 - 10: $\boldsymbol{\Sigma}_\omega^* \leftarrow \frac{1}{N} \sum_{n=1}^N \left((\hat{\boldsymbol{\omega}}_n - \boldsymbol{\mu}_\omega)(\hat{\boldsymbol{\omega}}_n - \boldsymbol{\mu}_\omega)^\top \right)$
 - 11: $\boldsymbol{\Sigma}_\omega \leftarrow \frac{1}{N+v_0+KD+1} [\mathbf{S}_0 + N \boldsymbol{\Sigma}_\omega^*]$
 - 12: $\boldsymbol{\Sigma}_y \leftarrow \frac{1}{L} \sum_{n=1}^N \sum_{t=1}^{\tau_n} [\boldsymbol{\epsilon}_{nt} \boldsymbol{\epsilon}_{nt}^\top]$
 - 13: **end while**
 - 14: **return** $\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega$ and $\boldsymbol{\Sigma}_y$.
-

2.2.3 Relation to the Least Squares method to train ProMPs

An alternative method of training ProMPs was proposed by [22]. We show that the method proposed in [22] is a special case of the EM algorithm presented in this chapter for the MLE case, with a single iteration and approximating the Gaussian distributions over the hidden variables $\boldsymbol{\omega}_n$ with a Dirac delta distribution on the mean.

The method presented in [22] consists of making point estimates of the hidden variables $\boldsymbol{\omega}_n$ with least squares. Subsequently, the mean and covariance of the point estimates are used to estimate the ProMP parameters. The point estimates of $\boldsymbol{\omega}_n$ are computed for every trajectory using

$$\boldsymbol{\omega}_n = (\boldsymbol{\Phi}_n^\top \boldsymbol{\Phi}_n + \lambda \mathbf{I})^{-1} \boldsymbol{\Phi}_n^\top \mathbf{y}_n, \quad (2.5)$$

where $\boldsymbol{\Phi}_n$ and \mathbf{y}_n are the vertical concatenation of the matrices $\boldsymbol{\Phi}_{nt}$ and vectors \mathbf{y}_{nt} respectively, and λ is a ridge regression parameter that can be set to zero unless numerical problems arise. Subsequently, the ProMP parameters can be estimated using the MLE estimates for Gaussian distributions

$$\boldsymbol{\mu}_\omega^* = \frac{1}{N} \sum_{n=0}^N \boldsymbol{\omega}_n, \quad (2.6)$$

$$\boldsymbol{\Sigma}_\omega^* = \frac{1}{N} \sum_{n=0}^N (\boldsymbol{\omega}_n - \boldsymbol{\mu}_\omega)(\boldsymbol{\omega}_n - \boldsymbol{\mu}_\omega)^\top. \quad (2.7)$$

Figure 2.4 shows the numerical stability of the matrix $\boldsymbol{\Sigma}_\omega$ of equations (2.5) to (2.7) using multiple values of λ . Note that this training procedure has the same numerical stability issues that the MLE estimates independently of the value of λ . The reason is that the numerical problems do not come from the estimation of $\boldsymbol{\omega}$, where λ is being used, but from the estimation

of the covariance matrix itself on (2.7). Note also that Figure 2.4 displays the conditioning number of Σ_ω for this method using a minimum of 36 demonstrations. The reason is that using (2.7) with $N < KD$ would result in a rank deficient matrix Σ_ω whose condition number would be $+\infty$.

The discussed numerical issues of training a ProMP using (2.5) to (2.7) make the learned model dangerous to use directly for robotic applications. In [22], a small diagonal matrix is added to Σ_ω and in addition a artificial noise matrix Σ_y^* needs to be used during conditioning. A very large number of training instances $N \gg KD$ would be required to be able to use (2.5) to (2.7) without any additional tricks. Using the proposed prior distribution solves the numerical problems in a theoretically sound way, and in the limit of infinite amount of data it converges to the expected estimation procedure using maximum likelihood.

However, the estimation method proposed in Algorithm 1 differs from using (2.5) to (2.7) in more than just using a prior distribution. To show the differences and similarities, let us now analyze the EM algorithm presented in this chapter if we approximate the E-step with a Dirac delta distribution. Note that using a Dirac delta distribution $\delta(\omega - \hat{\omega}_n)$ means making a point estimate $\hat{\omega}_n$ of the hidden variables ω_n without any uncertainty. The value of the point estimates $\hat{\omega}_n$ is given by

$$\hat{\omega}_n = (\Sigma_\omega^{-1} + \Phi_n^\top \Sigma_y^{-1} \Phi_n)^{-1} (\Sigma_\omega^{-1} \mu_\omega + \Phi_n^\top \Sigma_y^{-1} \mathbf{y}_n). \quad (2.8)$$

Algorithm 2 shows the resulting EM algorithm with the discussed approximation for the E-step. The quality of the approximation depends on how much uncertainty is there in the computation of the hidden variables. Let us further assume that we execute one single iteration of Algorithm 2 with initial values $\Sigma_\omega = \lambda^{-1} \mathbf{I}$, $\mu_\omega = 0$ and $\Sigma_y = \mathbf{I}$. It is easy to see that the estimates $\hat{\omega}_n$ would be exactly equivalent to the estimates (2.5) used by [22]. Note also that Lines 8 and 10 of Algorithm 2 compute also exactly the same estimates of [22] on (2.6) and (2.7) for the ProMP parameters in the MLE case.

We can conclude that the training procedure from [22] is equivalent to a single iteration of the approximated training procedure presented in Algorithm 2 on the MLE case with a particular initialization of the ProMP parameters. We have already extensively discussed the advantages of using MAP estimates using the proposed prior distribution. The remaining questions we want to discuss are whether using uncertainty estimates and more than one EM iteration is helpful. Estimating the uncertainty helps in applications where there is actually high uncertainty in the estimation of the hidden variables ω_n due for example to missing observations or high sensor noise. The answer of how much multiple iterations help depends entirely on the parameter initialization (see Line 3 of Algorithm 2). Note that the only difference between the first iteration and the rest is that in the first iteration we are working entirely on our initial guess of the values of the ProMP parameters. Whereas in subsequent iterations we are using optimized estimates of the ProMP parameters.

For our robot experiments, the sensor noise is on the order of 10^{-3} radians and the number of samples per trajectory is between 200 and 500 per degree of freedom. Furthermore, there are no missing observations as we can always read the joint sensor values. With such a low signal to noise ratio and without missing observations the values of the hidden variables ω_n can be estimated very precisely and with low uncertainty. As we expected, we did not observe any difference in the performance in any of our robot experiments using the estimates produced by Algorithms 1 and 2. In fact, the estimated parameters μ_ω and Σ_ω produced by both algorithms

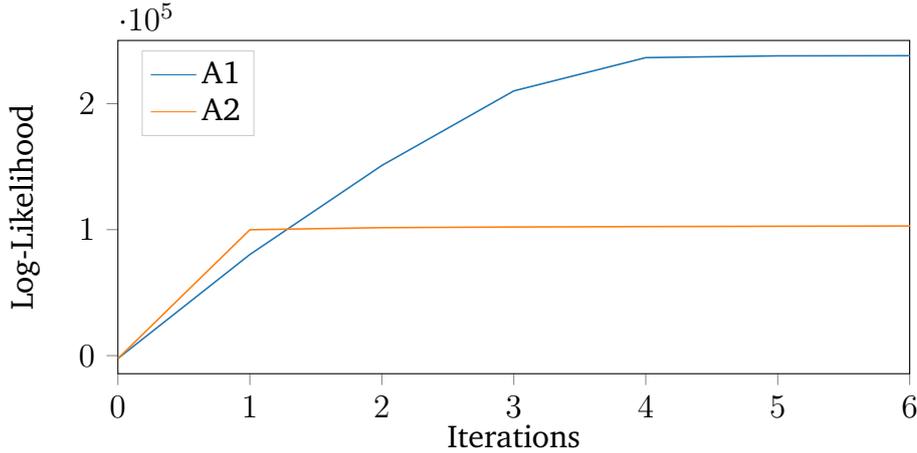


Figure 2.5.: Log-Likelihood improvement with every iteration of the proposed EM algorithm presented in Algorithm 1 (A1) and the approximated version presented in Algorithm 2 (A5). Both algorithms make initially poor estimates of the hidden variables ω_n . However, the proposed algorithm also captures the uncertainty over the estimates of the hidden variables, opposed to the approximated algorithm. As a result, the proposed algorithm continues to successfully improve the likelihood after the first iteration, whereas the approximated algorithm improves only marginally after the first iteration.

are virtually the same. We can conclude that for modeling robot trajectories on a robot setup like ours, using the approximation of the E-step with a Dirac delta distribution does not impact the performance compared to the complete EM estimation.

To show an example problem where uncertainty estimates are more important, we decided to run a small additional experiment where we use a ProMPs to model a table tennis ball trajectory. We collected 80 ball trajectories using the robot vision system, there are a few missing observations due to occlusion or errors in the image processing algorithms as well as a higher signal to noise ratio. Subsequently, we trained two models using the exact and approximated training algorithms. Finally, we tested the trained models predicting the ball position at time $t = 1.2s$ given the first 160 milliseconds of ball observations. On this experiment, we used $K = 8$ basis functions and $D = 3$ dimensions. The initial parameters for both algorithms were $\Sigma_\omega = \mathbf{I}$, $\mu_\omega = 0$ and $\Sigma_y = \mathbf{I}$

Figure 2.5 shows the evaluation of the log likelihood for each iteration of the EM algorithm for both the proposed and the approximated versions. Both algorithms are provided the exact same data and use the exact same parameter initialization. Note that the proposed algorithm outperforms the approximated algorithm in this particular problem. The distance between the ground truth ball position measured by the vision system and the position predicted by the ProMP models trained with Algorithm 1 is around **10 cm**, whereas the error of the ProMP trained with Algorithm 2 is around **50 cm**.

As a final argument in favour of using Algorithm 1 instead of Algorithm 2, note that we are not gaining anything out of the approximation. The algorithmic complexity is exactly the same in both cases, and estimating the uncertainty does not hurt the learning algorithm even on the cases where it is very low and the approximation seems to be accurate.

2.3 Adaptation of Probabilistic Movement Primitives

Adapting a movement primitive by setting initial positions, desired via points or final positions is a necessary property to generalize to different situations. These desired via points could be specified in joint space or in task space. For example, a table tennis striking movement

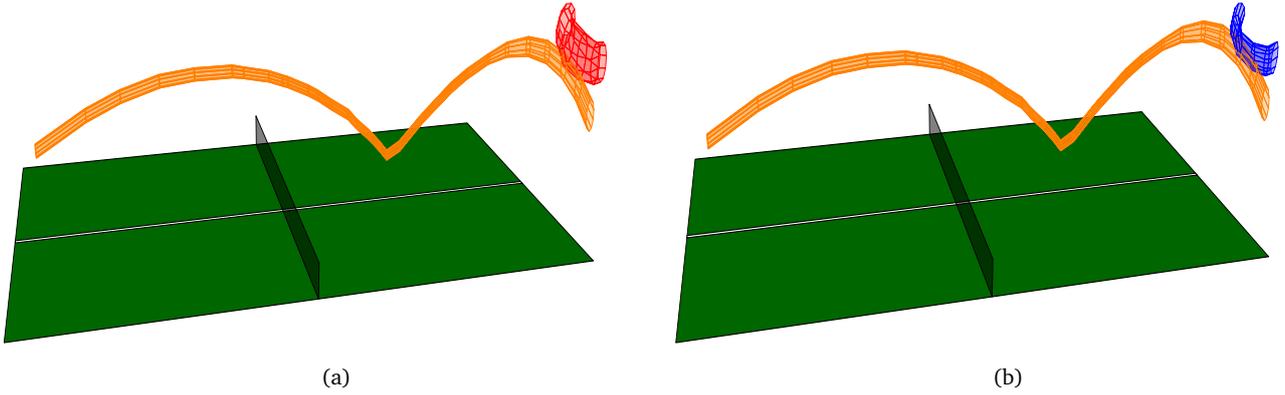


Figure 2.6.: Task space distributions of the ball and the racket center before and after adapting the ProMP in task space. The distribution of the ball is presented in orange. Figure 2.6a depicts in red the distribution of the center of the racket computed from the ProMP learned from human demonstrations. Subsequently, the ProMP is adapted to hit the ball using Algorithm 3, and the resulting ProMP is depicted in blue in Figure 2.6b. Note that the adapted ProMP is similar to the original ProMP learned from human demonstrations, but the probability mass is concentrated in the area that overlaps with the ball trajectory distribution.

needs to start in the current joint configuration of the robot and later reach the predicted task space position of the table tennis ball. In this section, we present operators to adapt movement primitives in joint and in task space. In addition, we evaluate the execution time of these operators showing that they can all run in less than one millisecond on a standard computer, satisfying the real time requirements of the applications presented in this chapter.

2.3.1 Adapting a ProMP in Joint Space

In the original formulation of ProMPs [1], it was proposed to adapt a ProMP in joint space by conditioning on a desired observation \mathbf{y}_t^* with some noise matrix Σ_y^* that was referred to as the desired accuracy. However, the authors do not provide any intuition on how Σ_y^* should be computed or estimated.

We introduce an approach to adapt in joint space by conditioning in the distribution over joint trajectories to reach a particular value $\mathbf{y}_t = \mathbf{y}_t^*$ without any artificial accuracy matrix Σ_y^* . The reason why we do not need the artificial noise matrix Σ_y^* opposed to [1] is that we do not suffer from numerical problems inverting Σ_ω due to the different training procedure. The conditioned distribution $p(\omega | \mathbf{y}_t = \mathbf{y}_t^*) \propto p(\mathbf{y}_t = \mathbf{y}_t^* | \omega) p(\omega)$ can be computed in closed form and is given by

$$\begin{aligned}
 p(\omega | \mathbf{y}_t = \mathbf{y}_t^*) &= \mathcal{N}(\omega | \mathbf{m}_\omega, \mathbf{S}_\omega), \\
 \mathbf{m}_\omega &= \mathbf{S}_\omega (\phi_t^\top \Sigma_y^{-1} \mathbf{y}_t^* + \Sigma_\omega^{-1} \boldsymbol{\mu}_\omega), \\
 \mathbf{S}_\omega &= (\Sigma_\omega^{-1} + \phi_t^\top \Sigma_y^{-1} \phi_t)^{-1}.
 \end{aligned}$$

There are cases where we do not know the exact value of the desired joint configuration \mathbf{y}_t^* , but instead we have a probability distribution $\mathbf{y}_t^* \sim \mathcal{N}(\mathbf{y}_t^* | \boldsymbol{\mu}_q, \Sigma_q)$. For example, in the table tennis task, we need to condition the striking movement on the future position of the ball predicted with the ball model. The distribution of the ball is, however, in task space. In Section 2.3.3, we explain how to transform a target task space distribution to a joint space distribution. For the moment, we assume we have a target distribution in joint space, that we can marginalize using

$$p(\omega | \boldsymbol{\mu}_q, \Sigma_q) = \int p(\omega | \mathbf{y}_t = \mathbf{y}_t^*) \mathcal{N}(\mathbf{y}_t^* | \boldsymbol{\mu}_q, \Sigma_q) d\mathbf{y}_t^*,$$

which can be computed in closed form obtaining

$$p(\boldsymbol{\omega} | \boldsymbol{\mu}_q, \boldsymbol{\Sigma}_q) = \mathcal{N}(\boldsymbol{\omega} | \mathbf{m}_\omega, \mathbf{S}_\omega),$$

$$\mathbf{m}_\omega = \mathbf{S}_\omega(\boldsymbol{\phi}_t^\top \boldsymbol{\Sigma}_y^{-1} \boldsymbol{\mu}_q + \boldsymbol{\Sigma}_\omega^{-1} \boldsymbol{\mu}_\omega), \quad (2.9)$$

$$\mathbf{S}_\omega = \mathbf{T}_\omega + \mathbf{T}_\omega \boldsymbol{\phi}_t^\top \boldsymbol{\Sigma}_y^{-1} \boldsymbol{\Sigma}_q \boldsymbol{\Sigma}_y^{-1} \boldsymbol{\phi}_t \mathbf{T}_\omega, \quad (2.10)$$

with $\mathbf{T}_\omega = (\boldsymbol{\Sigma}_\omega^{-1} + \boldsymbol{\phi}_t^\top \boldsymbol{\Sigma}_y^{-1} \boldsymbol{\phi}_t)^{-1}$. The ProMPs can also be adapted with desired velocities or accelerations using the same method, replacing the basis function matrices $\boldsymbol{\phi}_t$ by their respective time derivatives $\dot{\boldsymbol{\phi}}_t$ and $\ddot{\boldsymbol{\phi}}_t$.

The run time complexity for both adaptation operators is bounded by $O(K^3 D^3)$. In our robot experiments we used a model with $KD = 35$, obtaining an average execution time of 0.044 ms. In the experimental section, we provide running times for different model sizes.

2.3.2 Probability Distribution of a ProMP in Task Space

We compute a probability distribution in task space from a ProMP learned in joint space making use of the geometry of the robot. We assume that we have access to a deterministic function $\mathbf{x}_t = \mathbf{f}(\mathbf{y}_t)$ called the forward kinematics function that returns the position in task space \mathbf{x}_t of a point of interest like the end effector of the robot given the joint state configuration \mathbf{y}_t . The deterministic forward kinematics function \mathbf{f} , can be expressed in our probabilistic framework using

$$p(\mathbf{x}_t | \mathbf{y}_t) = \delta(\mathbf{x}_t - \mathbf{f}(\mathbf{y}_t)),$$

where δ is the Dirac delta function. The task space distribution can be computed from the ProMP parameters learned in joint space using

$$p(\mathbf{x}_t | \boldsymbol{\theta}_\omega) = \int p(\mathbf{y}_t | \boldsymbol{\theta}_\omega) p(\mathbf{x}_t | \mathbf{y}_t) d\mathbf{y}_t.$$

The distribution $p(\mathbf{x}_t | \boldsymbol{\theta}_\omega)$ can not be computed in closed form for a non-linear forward kinematics function. We compute an approximated distribution $p(\mathbf{x}_t | \boldsymbol{\theta}_\omega)$ making a linear Taylor expansion of the forward kinematics function around the ProMP mean, obtaining

$$p(\mathbf{x}_t | \boldsymbol{\theta}_\omega) = \mathcal{N}(\mathbf{x}_t | \mathbf{f}(\boldsymbol{\Phi}_t \boldsymbol{\mu}_\omega), \mathbf{J}_t \boldsymbol{\Sigma}_\omega \mathbf{J}_t^\top), \quad (2.11)$$

where $\mathbf{J}_t = \mathbf{J}(\boldsymbol{\Phi}_t \boldsymbol{\mu}_\omega)$ is the Jacobian of the forward kinematics function [25] evaluated at $\mathbf{y}_t = \boldsymbol{\Phi}_t \boldsymbol{\mu}_\omega$. Figure 2.6a shows the task space distribution of a ProMP learned from demonstrations to strike a table tennis ball as well as some particular ball trajectory distribution. The distribution of the center racket is depicted in red and the distribution of the predicted ball trajectory is depicted in orange.

2.3.3 Adapting ProMPs in Task Space

For many applications, it is more natural to define goals in task space. For instance, in robot table tennis the movement primitive should be adapted such that the position of the racket

matches the predicted position of the ball. In this section, we present our approach to condition a ProMP learned in joint space to have a desired task space distribution.

We denote the desired task space state at time t by the random variable \mathbf{x}_t , with probability distribution given by

$$p(\mathbf{x}_t|\boldsymbol{\theta}_x) = \mathcal{N}(\mathbf{x}_t | \boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x),$$

where the parameters $\boldsymbol{\theta}_x = \{\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x\}$ are user inputs that represent the desired task space configuration and its uncertainty respectively.

Given a desired end effector position \mathbf{x}_t and a ProMP with parameters $\boldsymbol{\theta}_\omega = \{\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega\}$, a probability distribution for the joint configuration can be computed by

$$p(\mathbf{y}_t|\mathbf{x}_t, \boldsymbol{\theta}_\omega) \propto p(\mathbf{x}_t|\mathbf{y}_t)p(\mathbf{y}_t|\boldsymbol{\theta}_\omega), \quad (2.12)$$

where $p(\mathbf{x}_t|\mathbf{y}_t)$ is given by (2.3.2) and $p(\mathbf{y}_t|\boldsymbol{\theta}_\omega)$ is the joint space distribution given by the ProMP.

The distribution $p(\mathbf{y}_t|\mathbf{x}_t, \boldsymbol{\theta}_\omega)$ represents a compromise between staying close to the demonstrated trajectories and achieving the desired racket configuration. Thus, for a robot arm with redundant degrees of freedom, where multiple joint space configurations can achieve the desired racket configuration, the presented approach will prefer joint solutions that are closer to the demonstrated behavior.

To achieve the desired task space distribution $p(\mathbf{x}_t|\boldsymbol{\theta}_x)$ instead of a particular value \mathbf{x}_t , we marginalize out \mathbf{x}_t from (2.12) obtaining

$$\begin{aligned} p(\mathbf{y}_t|\boldsymbol{\theta}_x, \boldsymbol{\theta}_\omega) &= \int p(\mathbf{y}_t|\mathbf{x}_t, \boldsymbol{\theta}_\omega)p(\mathbf{x}_t|\boldsymbol{\theta}_x)d\mathbf{x}_t \\ &\propto p(\mathbf{y}_t|\boldsymbol{\theta}_\omega) \int p(\mathbf{x}_t|\boldsymbol{\theta}_x)p(\mathbf{x}_t|\mathbf{y}_t)d\mathbf{x}_t. \end{aligned} \quad (2.13)$$

Note that $p(\mathbf{y}_t|\boldsymbol{\theta}_x, \boldsymbol{\theta}_\omega)$ is a distribution in joint space that again compromises between staying close to the demonstrated behavior and achieving the desired task space distribution. The integral in (2.13), required to compute the normalization constant of $p(\mathbf{y}_t|\boldsymbol{\theta}_x, \boldsymbol{\theta}_\omega)$, is intractable. We used Laplace Approximation [26] to compute a Gaussian approximation for $p(\mathbf{y}_t|\boldsymbol{\theta}_x, \boldsymbol{\theta}_\omega)$. With a Gaussian distribution for $p(\mathbf{y}_t|\boldsymbol{\theta}_x, \boldsymbol{\theta}_\omega)$, the operator to adapt ProMPs in joint space discussed in Section 2.3 can be used to obtain a new adapted ProMP.

Algorithm 3 Algorithm to adapt a ProMP in task space using Laplace Approximation

Input: Parameters of desired task space distribution $\boldsymbol{\theta}_x = [\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x]$ and ProMP to adapt $\boldsymbol{\theta}_\omega = [\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega]$.

Output: A new ProMP modulated to strike the ball

- 1: $\boldsymbol{\mu}_q \leftarrow \arg \max_{\mathbf{y}_t} (\log p(\mathbf{y}_t|\boldsymbol{\theta}_x, \boldsymbol{\theta}_\omega))$
 - 2: Compute $\boldsymbol{\Lambda}_q$ as the second derivative of $\log p(\mathbf{y}_t|\boldsymbol{\theta}_x, \boldsymbol{\theta}_\omega)$ with respect to \mathbf{y}_t evaluated at $\mathbf{y}_t = \boldsymbol{\mu}_q$
 - 3: $\boldsymbol{\Sigma}_q \leftarrow \boldsymbol{\Lambda}_q^{-1}$
 - 4: Compute \mathbf{m}_ω and \mathbf{S}_ω with (2.9) and (2.10)
 - 5: **return** new ProMP with $\boldsymbol{\mu}_\omega = \mathbf{m}_\omega$ and $\boldsymbol{\Sigma}_\omega = \mathbf{S}_\omega$
-

Algorithm 3 describes the procedure to adapt a ProMP in task space using Laplace Approximation. The mean μ_q and covariance Σ_q of the approximated joint space distribution are computed in Lines 1 and 3 respectively. The presented operator for task space conditioning consists of a non linear optimization to compute $p(\mathbf{y}_t|\boldsymbol{\theta}_x, \boldsymbol{\theta}_\omega)$ followed by a use of the joint space conditioning operator. As a result, the task space conditioning operator is necessarily slower than the joint space conditioning operator. In Section 2.3.4, we show that the execution time of the presented operator is nonetheless reliably below 3 milliseconds for ProMP sizes up to $KD = 350$, satisfying the real time requirements of our robot applications by a large margin.

Figure 2.6 depicts the task space distribution of a ProMP learned from forehand strike demonstrations before 2.6a and after 2.6b adapting it to hit a ball trajectory seen at test time. Note that the adapted ProMP has the probability mass concentrated in the region that overlaps with the ball trajectory distribution.

2.3.4 Execution Time of the Presented Operators

Many use cases for the operators presented in this chapter to adapt the movement primitives will have real time execution requirements. If we want to adapt a movement primitive with respect to sensor values measured at time t_1 and subsequently execute the movement primitive in the robot at time t_2 , the total execution time for the operator cannot exceed $t_2 - t_1$.

For example, to make sure that the executed movement primitive starts on the current robot joint state, we use the joint conditioning operator on the measured joint state just before starting the execution of the movement primitive. For our robot experiments, we used a control loop of 500 Hertz. Therefore, we have a real time constrain of 2 ms to read the sensor value for the joint state, condition the ProMP to start at the measured value and send the required motor commands.

Table 2.1 show the average execution time and standard deviation in milliseconds for the operators presented in this chapter. Each operator is executed 1000 times for each of the different sizes of ProMPs in a Lenovo Thinkpad X2 Carbon laptop with a processor Intel Core i7-6500U 2.50GHz and 8 GB of RAM. We report the size as the product of the degrees of freedom D and the number of basis functions per degree of freedom K .

On our robot experiment we used a ProMP with size $K = 5$ and $D = 7$, that corresponds to the smallest entry in Table 2.1. However, note that even on a ProMP with $KD = 350$ we can meet the real time requirements to play robot table tennis. The operator to condition in joint space can be reliably run under the 2 milliseconds required for our control loop of 500 Hertz. The vision system used on the experiments presented in this chapter [27], produces 60 ball observations per second. Therefore, we can potentially correct the ProMP trajectory to changes in the ball trajectory after every ball observation with a running time below 16 milliseconds.

KD	Joint Space [ms]	Task Space [ms]
35	0.0448 ± 0.0164	0.7212 ± 0.2920
70	0.0642 ± 0.0104	0.8328 ± 0.5484
140	0.1880 ± 0.0245	1.0764 ± 0.3179
210	0.5294 ± 0.5879	1.4291 ± 0.2423
280	0.8686 ± 0.7944	1.9267 ± 0.3822
350	1.2095 ± 0.4829	2.3173 ± 0.3135

Table 2.1.: Average execution time of joint and task space conditioning operators in milliseconds for ProMPs of different sizes. The table presents the mean and standard deviation of the running times for each operator in milliseconds.

Note that our task space conditioning operator runs reliably under 3 milliseconds, satisfying the real time requirements by a large margin.

2.3.5 Additional Operators for Hitting Movements

The operators introduced in Sections 2.3 and 2.3.2 are general purpose operators that can be used on any application to adapt ProMPs in joint space or task space. Application specific knowledge can also be incorporated into probabilistic frameworks in the form of likelihoods or prior distributions. In this section we introduce a task specific operator to determine the starting time of the hitting movement primitive that maximizes the likelihood of hitting the ball.

Learning an appropriate initial time of a movement to strike a ball just from human demonstrations is not easy. The moment to start the hitting movement depends highly on the current ball trajectory, and its correct estimation is crucial to hit the ball.

In [20], the Virtual Hitting Plane (VHP) method is used to estimate the hitting time. The VHP method consists on computing a hitting point as the intersection between the predicted ball trajectory mean and a predefined plane. Subsequently, the start time is computed based on the predicted hitting time. The VHP approach assumes that the optimal hitting points are always located on a predefined plane [28]. One disadvantage of this approach is that finding a particular hitting plane that works for a diverse set of ball trajectories is hard.

In this section we define the likelihood of hitting the ball. The hitting point and hitting time are marginalized out in the formulation of this likelihood. As a result, the proposed method makes no assumption on the location of the hitting points unlike the VHP method.

Likelihood of Hitting the Ball

Let $\mathbf{x}_b(t)$ and $\mathbf{x}_r(t)$ be random variables that represent the position of the ball and the racket at time t respectively. We say that the racket hits the ball at time t if $\mathbf{x}_b(t) = \mathbf{x}_r(t)$, ignoring for simplicity the geometry of the ball and the racket. For brevity, we will drop the time t from the notation unless it is necessary.

Let us define a new continuous random variable $\mathbf{z}(t) = \mathbf{x}_b(t) - \mathbf{x}_r(t)$ representing the difference between the position of the ball and the racket. The ProMP is learned from recorded robot trajectories independently of the ball trajectories. As a result, the probability distributions of the racket $p(\mathbf{x}_r(t))$ and the ball $p(\mathbf{x}_b(t))$ are independent, and the probability distribution of \mathbf{z} is given by

$$p(\mathbf{z}) = \int p(\mathbf{x}_b = \mathbf{x})p(\mathbf{x}_r = \mathbf{x} - \mathbf{z})d\mathbf{x}. \quad (2.14)$$

Note that $p(\mathbf{z})$ is a probability density function. The event of hitting the ball, formalized as $\tilde{x}_r = \tilde{x}_b$, corresponds to $\mathbf{z} = \mathbf{0}$ on the probabilistic formulation. If we evaluate the density on $\mathbf{z} = \mathbf{0}$, we obtain

$$p(\mathbf{z} = \mathbf{0}) = \int p(\mathbf{x}_b = \mathbf{x})p(\mathbf{x}_r = \mathbf{x})d\mathbf{x}. \quad (2.15)$$

The probability distribution of the trajectory of the ball can be computed using a Kalman Filter or a ProMP over ball positions, and the distribution of the racket is computed from the ProMP

in joint space using the operator described in Section 2.3.2. A solution for (2.15) can be efficiently found in closed form [29] because $\mathbf{x}_b(t)$ and $\mathbf{x}_r(t)$ are Gaussian distributed. Given a ball trajectory, the initial time of the movement primitive is computed by maximizing the likelihood of hitting the ball.

To increase the robustness of the algorithm, we marginalize the hitting time with a prior distribution $p_h(t)$, computing the likelihood of hitting the ball $H(\boldsymbol{\theta}_h)$ as

$$H(\boldsymbol{\theta}_h) = \int_{t_0}^T p(\mathbf{z}(t) = \mathbf{0})p_h(t)dt, \quad (2.16)$$

where $\boldsymbol{\theta}_h = (t_0, T)$ is the initial time and duration of the movement primitive.

Using a uniform prior distribution over time $p_h(t)$ to find the optimal starting time, results on hitting the ball mostly at the beginning or the end of the striking movement, where the speed of the racket is low and hitting the ball is easier. However, to send the ball to the opponent's side of the table a higher racket velocity is desirable. We use a Gaussian distribution $p_h(z) = \mathcal{N}(z | \mu_z = 0.5, \sigma_z = 0.1)$, where $z = (t - t_0)/T$ is the time variable normalized to be between zero and one. Note that this prior assigns higher probability mass to hit the ball in the middle of the movement, where the velocity of the movement is higher. As a result, the rate of balls returned to the opponent side of the table increased from 3% to 47%.

Safety in the Execution of the Movement

Previous approaches to robot table tennis use complex heuristics to ensure that the planned trajectories are safe to execute. Common heuristics include detection of collisions between the robot and the table [30], verification that the planned trajectory do not exceed the joint limits [31], etc. Having good heuristics to ensure safe movements is important in table tennis. It is easy to find unexpected events that could result in dangerous movements for the robot. For instance, if the ball hits the net, then rolls on the table, and the robot still attempts to hit the ball, it will also hit the table.

We use a threshold on the likelihood of hitting the ball as the unique safety measure. If the likelihood of hitting the ball before conditioning the ProMP is very low, it is because the movement required to hit the current ball trajectory is very different from the human demonstrated behavior. Our intuition for using the threshold as a security measure is that movements that are similar to the human demonstrations are safe to execute. Using only this simple heuristic, no dangerous movement has been executed by the robot in more than 2000 robot trials used for the experiments on this chapter and robot demonstrations to the public.

2.4 Experiments and Results

We evaluate the presented methods with synthetic data and with a real robot experiments for table tennis and assisting coffee brewing. For the robot experiments we used Barrett WAM arm with seven degrees of freedom capable of high speed motion. The robot control computer uses a 500 Hz control loop, receiving joint angle measures and output motor commands every 2 ms. To track the position of objects of interest like the table tennis ball and the coffee machine, we used four Prosilica Gigabit cameras and the vision system described in [27]. This vision system

tracks the position of a table tennis ball with an approximate frequency of 60 Hz, we attached a table tennis ball to the coffee machine for the coffee brewing experiments.

On all our robot experiments we used five basis functions per degree of freedom. Fifth [20] and third [32] order polynomials have been previously used successfully for robot table tennis approaches. Note that the same results should be achievable with a ProMP with six or four polynomial basis functions respectively taking into account the constant term. On the other hand, radial basis functions (RBFs) have been typically used with ProMPs [1] for other robot applications. We tried different combinations of RBFs and polynomial basis functions, obtaining the best results using three RBFs and a first order polynomial, for a total of five basis functions.

2.4.1 Parameter Convergence on Synthetic Data

The purpose of the experiment with synthetic data, is to evaluate how accurate are the estimates of the ProMP parameters as a function of the number of training instances n when the assumptions we made for the prior distribution are incorrect. We generate synthetic data from a reference ProMP that displays a strong correlation between different degrees of freedom, opposing the proposed prior assumptions. Subsequently, we test if the proposed learning procedure converges to the expected parameters and how many training examples are necessary for convergence.

On this synthetic data experiment there is no notion of training or test sets. We simply generate n sample trajectories from a reference ProMP with known parameters μ_ω and Σ_ω .

Subsequently, we train a new ProMP with the sampled trajectories obtaining a new set of parameters $\hat{\mu}_\omega^n$ and $\hat{\Sigma}_\omega^n$ and compare how close they are to the reference parameters μ_ω and Σ_ω using the Frobenius norm. In this experiment we used five basis functions $K = 5$ and four degrees of freedom $D = 4$. To ensure a high correlation, we set the parameters of the base ProMP such that the last two degrees of freedom are the addition and subtraction of the first two degrees of freedom respectively.

Figure 2.7 show the average parameter estimation error with respect to the number of training instances n for different set of parameters. The error over the parameters μ_ω that represent the mean behavior is depicted in red. The error over the parameters Σ_ω are divided in the block diagonal terms that represent the captured variability of the movement (depicted in blue) and the rest of the parameters that represent the captured joint correlations (depicted in green). The error of the different set of parameters is normalized between zero and one to facilitate comparison, and the error curves are smoothed out using splines to facilitate visualization of convergence.

Note that the learning algorithm converges to the true value as expected. However, more training examples are required to converge to the correlation parameters because the prior is favour-

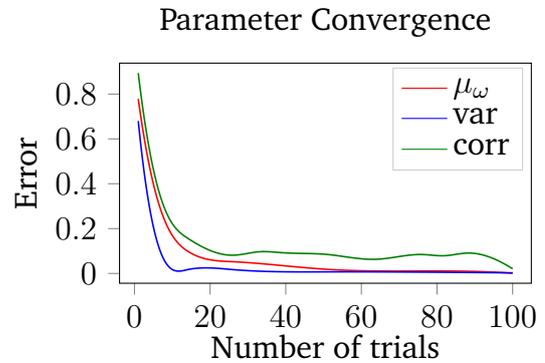


Figure 2.7.: Convergence of the ProMP parameters as a function of the number of training instances in an adversarial scenario. The convergence of different sets of parameters is depicted with different colors. The set of parameters corresponding to the mean behavior, variability of the movement, and correlation between joints are depicted in red, blue and green respectively.

ing joint independence in a high joint correlation scenario. The effect of the proposed prior is to prefer independence between the joints in absence of strong evidence of correlation.

The results from this experiment may suggest that the presented probabilistic framework require large amounts of data samples to learn a movement primitive. In contrast, we show we can learn a coffee-pouring and a table-tennis experiment that the proposed approach, using only two and eight training examples respectively. There are two main explanations why we can converge with fewer training instances to the target performance on different tasks. First, the prior distribution assumptions may be more accurate in some real world tasks than in the adversarial example chosen in this section. Second, we can not compare convergence in parameter space to convergence in the performance of a particular task. The reason is that there might be multiple different parameter values with a similar task performance.

2.4.2 Assisting Coffee Brewing

A coffee preparation task was one of the tasks used to evaluate the proposed methods. We use an inexpensive coffee grinder and an Aeropress as a brewing method. Figure 2.8 depicts the robot executing the steps required to prepare a cup of coffee. First, the robot needs to move to the top of the grinder and pour fresh coffee beans. Subsequently, the robot moves to place the spoon under the grinder funnel to pick the coffee grounds. Finally, the robot pours the coffee grounds into the brewing chamber.

The coffee task requires sequencing movement primitives to pour coffee beans of grounds in different locations and picking the grounds from the grinder. At the same time,

the robot should avoid hitting the grinder, coffee machine or the table to prevent damaging the robot, the coffee machines or spilling the coffee. Therefore, this task allows us to test the ability of the proposed framework to divide a complex task into multiple simpler primitives as well as learning from the teacher the right set of movements that avoid hitting external objects.

Additionally, the movement primitives to pour or pick coffee should be adapted to the position of the coffee machine or the grinder in order to succeed. The position of these objects is obtained from the vision system in task space, providing an opportunity to test the operator to adapt movement primitives in task space. The operator to condition movement primitives in joint space is also used to start the executed movement primitive at the robot current joint position. For the coffee task we want to test how well the proposed approach adapts to changes in the position of the grinder or the coffee machine, whereas for the table tennis task the goal is to determine how well it adapts to changes in the ball trajectory. Note that the position of the grinder and coffee machine in different experiment trials is easy to control with relatively good precision, while controlling the table tennis ball trajectory between different experiment trials is virtually impossible. As a result, we decided to invest more effort in the experiment design to test the generalization ability of a single movement primitive in the coffee task.

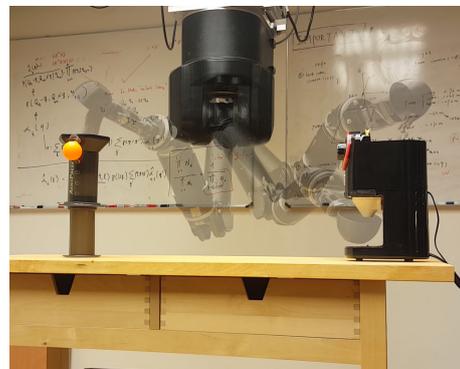


Figure 2.8.: The robot executing the coffee task. First, the robot moves towards the top of the coffee grinder to pour fresh beans into it. Subsequently, the robot moves towards the bottom of the grinder to pick the grounds. Finally, the robot deposits the coffee grounds in the brewing chamber of the coffee machine.

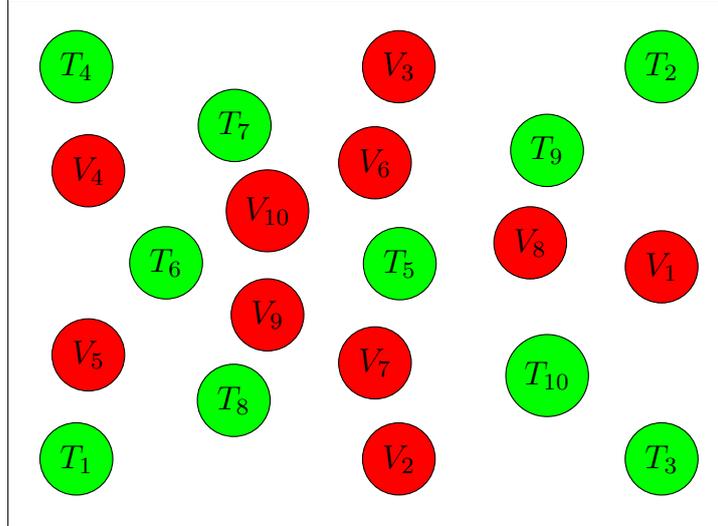


Figure 2.9.: Training and validation set pattern for the position of the coffee machine, designed to evaluate the generalization on a target area. The training pattern was selected with Lloyd’s algorithm to cover the target area evenly, and is depicted with green circles. The evaluation pattern is depicted in red, and was selected to be far from the training points while covering evenly the target area. The numbers in the green and red circles represent the order used for training and validation positions for the coffee machine respectively.

To test the generalization ability of a single movement primitive we focused on the movement that pours the coffee grounds in the coffee machine. We generated a pattern with training and evaluation positions for the coffee machine with a rectangular shape of 42cm x 59.4cm. This size corresponds exactly to an A2 format paper size that was printed for the experiments. To select a set of positions that covers evenly the training area we used Lloyd’s algorithm [33]. For the evaluation set we used an algorithm that selected a set of points in the rectangle that maximized the distance to the training set. Figure 2.9 shows a resized version of the resulting format for the training and the validation positions for the coffee machine as green and red circles respectively. The numbers on the circles represent the order of the events that should be used in the experiment, and we used them to test the performance on the training and validation sets as a function of the training data. In this experiment we evaluated the success rate of pouring coffee in the machine with a number of training instances varying from 1 to 10 training samples.

Algorithm 4 Procedure to test the generalization performance of a single ProMP on a pouring coffee experiment

Input: Training set positions $\{T_1, \dots, T_{10}\}$ and validation set positions $\{V_1, \dots, V_{10}\}$ from Figure 2.9.

Output: Training set performance P_n^t and evaluation set performance P_n^v with n training samples for $n \in \{1, \dots, 10\}$.

- 1: **for** $n \in \{1, \dots, 10\}$ **do**
 - 2: $\text{train}(\{T_1, \dots, T_n\})$
 - 3: $P_n^t \leftarrow \text{evaluate}(\{T_1, \dots, T_n\})$
 - 4: $P_n^v \leftarrow \text{evaluate}(\{V_1, \dots, V_{10}\})$
 - 5: **end for**
-

The procedure to train and evaluate the performance is explained with detail as a pseudo-code in Algorithm 4. In this pseudo-code the $\text{train}(\cdot)$ function consists on the human training the robot to pour coffee grounds on the coffee machine on the positions passed as argument, and the $\text{evaluate}(\cdot)$ function consists on the robot attempting to pour coffee on the specified positions and evaluating the success rate. For example, to evaluate the validation set performance

with two training examples, we would train the robot to pour coffee on positions $\{T_1, T_2\}$ and subsequently evaluate the pouring performance in positions $\{V_1, \dots, V_{10}\}$.

We used coffee beans instead of coffee grounds on the pouring experiments to simplify the definition of success in a pouring attempt. A trial is considered successful if and only if all the beans end up in the brewing chamber after pouring. No spilling is allowed, a trial is considered failed if one or more beans fall out of the brewing chamber after the pouring movement is executed.

Table 2.2 summarize the results of the pouring performance measured on this experiment. We expected a curve of generalization performance increasing slowly as a function of the number of training data, but the results obtained showed that after demonstrating the pouring movement only in T_1 and T_2 the robot could successfully generalize to all the validation points. With the same two training instances we tried to validate generalization in the points $\{T_3, \dots, T_{10}\}$ and the robot successfully poured coffee in those positions as well. Note that the results presented in Table 2.2 do not mean that the presented approach can generalize to any pouring point given only two demonstrations. If for example, we provide T_1 and T_8 as training examples and attempted to validate in the rest of the pouring area, not only the pouring is likely to fail but the resulting planned movement might be dangerous to execute. The ProMPs, as most machine learning methods that assume independently identically distributed data (IID), does not handle extrapolation well.

Training Samples	Training	Validation
1	1/1	1/10
$\{2, \dots, 10\}$	10/10	10/10

Table 2.2.: Summary of the results of the generalization performance experiment pouring coffee with a single ProMP. Using only two training samples was enough to generalize to all the target area. The obtained results suggest that at least for this task the selected prior is a sensible choice.

With only one training instance of pouring the robot could not generalize well. However, note that the robot managed to pour successfully at the given training position and one of the validation positions. The validation position where the robot poured successfully was V_5 , that is the closest validation point to the given training point T_1 , as can be seen in Figure 2.9. The distance between T_1 and V_5 in the printed pattern is 10.4 cm. We also tried to validate the single training instance example on the points $\{T_2, \dots, T_{10}\}$, but it failed spilling the coffee every time.

An alternative method to solve the coffee task without learning from human demonstrations would require trajectory planning with collision avoidance in order to succeed. Additionally, common sense knowledge like keeping the spoon pointing up all the time except when the robot is pouring would have to be explicitly programmed. Instead, our approach learns these common sense knowledge and strategies to succeed avoiding collisions with the grinder and brewing chamber from the human demonstrations. In the next section we evaluate our method in a table tennis task. We believe that robot table tennis is significantly harder than the coffee task presented in this section for a number of reasons that we discuss with more detail in the following section. Unfortunately, it is very hard to control precisely the ball trajectory and as a result, we cannot provide detailed generalization performance as with the coffee task. Instead, we will focus on evaluating the hit and return rate performance compared to previous work.

2.4.3 Robot Table Tennis

Robot table tennis is a highly dynamic task difficult to play for robots and humans. Unlike the coffee task it has strong real time requirements. The timing of the movement is as important as the movement itself to succeed hitting and returning the ball. Furthermore, it is not trivial or obvious which kind of movements would result in success for a given ball trajectory, making this problem especially interesting for learning approaches that can uncover these patterns given a set of successful trial examples.

In this section we evaluate the proposed approach in a robot table tennis setup. In this task we use a table tennis ball gun to throw balls to the robot. Subsequently, we measure whether or not the robot hits the ball and if the ball landed successfully in the opponent's court according to the table tennis rules.

For all the experiments presented in this section, we collected eight human demonstrations of a particular striking movement to train a ProMP. Unlike the coffee task, the high variability in the results makes it hard to determine the optimal number of training samples to increase the success rate. Informally, we did not notice any significant performance improvements using more than eight demonstrations.

To segment the striking movement from the rest of the demonstrated behavior we used the zero crossing velocity heuristic method. First, we found the point where the racket hit the ball t_h by detecting the change in direction of the ball. Subsequently, we found a time interval (t_a, t_b) such that $t_h \in (t_a, t_b)$ and both t_a and t_b were zero crossing velocity points. We found that this heuristic reliably segments table tennis striking movements if the hitting time t_h can be detected accurately. Some times we could not detect the hitting time t_h accurately because of vision problems. In such case we simply discarded that trajectory from the training set. We decided to use six as the minimum number of segmented demonstrations in the training set to proceed with the experiments. That is, if more than two demonstrations were discarded by the segmentation heuristic we collected the training data again.



Figure 2.10.: A human subject moving the robot in gravity compensation mode. Gravity compensation mode was used to obtain the human demonstrations necessary to train the robot.

Let us explain in detail how we apply the proposed method to table tennis. A high level pseudocode of the table tennis strategy is presented in Algorithm 5. This algorithm receives as input a ProMP already trained to play table tennis using human demonstrations, moves the robot to an initial position and blocks its execution until the vision system produces new ball observations. Subsequently, the obtained ball observations are used to predict the rest of the ball trajectory, the optimal initial time is computed from the ball trajectory by maximizing the likelihood of hitting the ball, and the trained ProMP is conditioned in task space using the operator presented in Section 2.3.2. Before executing the ProMP conditioned to hit the ball, it is conditioned in joint space to start in the current robot joint state.

Algorithm 5 Procedure used on the table tennis experiments

Input: A ProMP `promp0` trained for table tennis using human demonstrations.

```
1: while running do
2:   move_to_init_state(promp0)
3:   wait_ball_obs()
4:   repeat
5:     ball_obs  $\leftarrow$  get_ball_obs()
6:     ball_traj  $\leftarrow$  predict_ball_traj(ball_obs)
7:      $t_0 \leftarrow$  comp_optimal_t0(ball_traj)
8:     new_promp  $\leftarrow$  cond_hit(promp0, ball_traj)
9:   until  $t_0 \geq$  current_time()
10:  new_promp.cond_joint_space(get_joint_state())
11:  execute(new_promp)
12: end while
```

Note that the lines 5 and 8 in Algorithm 5 are in a loop to allow for re-planning. This feature is an important improvement over the previous work presented in [23], because it allows for corrections over the predictions of the ball trajectory produced by the ball model in line 6. In [23], a set of ball observations of a certain size was obtained and the ball model was used only once to predict the rest of the ball trajectory. Subsequently, the robot would “close its eyes” and attempt to hit the predicted ball trajectory. In consequence, it was hard to fix a sensible size for the initial set of observations. A small set would not provide enough information to predict accurately the ball trajectory, and a large set could potentially leave a small reaction time to the robot effectively losing the opportunity to hit the ball. We took advantage of the short execution time of the presented operators using re-planning. We simply take any amount of available ball observations to predict the ball trajectory and adapt the ProMP, but we keep doing so while there is still time for corrections.

The starting time of the movement primitive is computed in Line 7 using the operator presented in Section 2.3.5. We compare a uniform prior and a Gaussian prior over the hitting time. The Gaussian prior was discussed in Section 2.3.5, and it is based on the observation that humans typically hit the ball on the middle of the movement. We obtained a substantial improvement on the number of times that the robot manages to successfully return the ball to the opponent’s court, that we will call in the rest of this chapter the *success rate*.

The replanning feature is possible only because of the fast execution time of the proposed adaptation operators. Therefore, replanning is an example of how the computational efficiency of the proposed methods can have an impact on the success of a task where accurate prediction models are not available. Table 2.3 presents the results of an experiment to measure the improvement of performance due to re-planning and the hitting time prior both independently and combined. We placed the ball gun in a position that the human teacher found comfortable and collected a set of demonstrations, the ball gun parameters were kept fixed during the rest of the experiment. We trained a ProMP with Algorithm 1 using the collected demonstrations. We use the exact same trained ProMP during this experiment to make sure that the measured improvements are only due to the re-planning and hitting prior features. Note that the change in the prior over the hitting time had a very significant impact on the success rate, increasing it from 5.2% to 40.9% without re-planning and from 9.1% to 67.7% with re-planning. On the other

Re-planning	Hit time prior	Hit rate	Success rate
No	Uniform	73.7%	5.2%
No	Gaussian	79.5%	40.9%
Yes	Uniform	93.2%	9.1%
Yes	Gaussian	96.7%	67.7%

Table 2.3.: Performance improvement for hit and success rate due to re-planning and the prior over the hitting time. The ball gun was fixed to the same settings on these four experiments, and the same ProMP was used in every case trained with Algorithm 1. The goal of these experiments was to test the effect of re-planning and the hitting time prior both independently and combined. Note that re-planning has a significant positive impact mostly over the hitting rate, whereas the prior over the hitting time affects mostly the success rate. The best performance is obtained as expected with a combination of both.

hand, the re-planning feature improved in general about 20% on the hit rate, and the success rate improvement was only substantial in combination with the hitting time prior, improving from 40.9% to 67.7%.

In Section 3.2, we discussed how maximum likelihood estimation (MLE) produced unstable estimates of the ProMP parameters opposed to the Maximum A-Posteriori estimates (MAP). To prevent stability problems, the MLE estimates computed in [23] force the matrix Σ_ω to be block diagonal. As a result, the computed ProMP considers all the joints independent.

To measure the effect of using the proposed training method opposed to considering all the joints independent with MLE, we tested ProMPs trained with both methods with several ball gun configurations using always the procedure for execution on Algorithm 5 with both re-planning and the prior over the hitting time. We obtained an average success rate of 66.3% and a hit rate of 95.4% for the MAP trained ProMP. For MLE we obtained an average of 47.7% and 79.8% for the hit and success rates respectively.

We also compare the performance of our method with a different robot table tennis method based on heuristics [20] called the MoMP method [10]. Figure 2.10 shows a human subject moving the robot in gravity compensation mode.

Figure 2.11 shows an histogram of the success and hit rates obtained in this experiment for both MAP and MLE training, the MoMP method and the human subjects. The histogram was generated with the bootstrap method, generating 5000 random samples of 50 trials from the collected data. The success and hit rates were computed for each of the 5000 samples and recorded in the histogram. We decided to present an histogram of these results instead of just a number to account for the variability of the results natural to the table tennis experiments.

An interval containing 90% of the probability mass of the success rate histogram for the MLE and MAP trained ProMPs would locate the success rate between 34.0% and 58.0% for MLE and between 60.0% and 80.0% for MAP. From these confidence intervals we can conclude that the difference in success rate of learning the joint correlations with the MAP algorithm compared to the MLE algorithm that assumes the joints as independent is significant.

Furthermore, the table tennis procedure presented in Algorithm 5 used for the MAP and MLE trained ProMPs does not include any heuristic or method to successfully return the ball to the opponent’s court. In both cases this behavior has to be learned from the demonstrated data. The fact that the success rate of the MAP trained ProMP is significantly better than the success rate of the MLE trained ProMP that forces Σ_ω to be block diagonal, suggests that the joint correlations encode information important to successfully return table tennis balls.

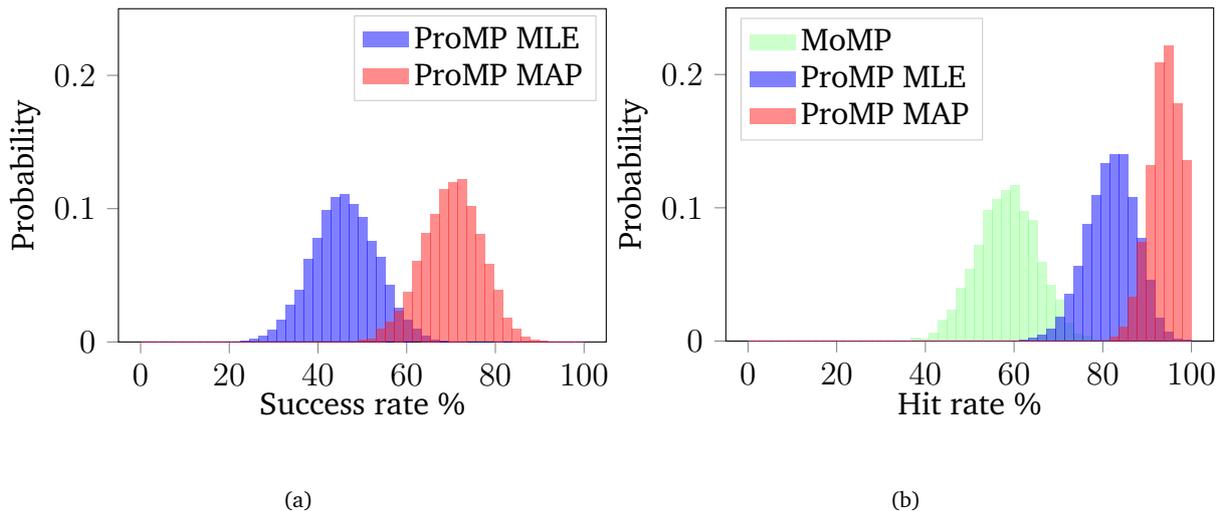


Figure 2.11.: Histogram of the success and hit rates on table tennis for ProMPs trained with MAP and MLE. We also compare against humans moving the robot in gravity compensation mode and the MoMP method. For the table tennis experiments, repeating the same experiment multiple times will likely produce different hit and success rate performance. In consequence, we decided to present a histogram of the results computed with the bootstrap method representing how likely is it to obtain a particular hit or success rate for different methods.

The performance of the presented approach was significantly better than the MoMP method for both hit and success rates in our experiments. The MoMP method is based on several heuristics that would require a great amount of hand tuning to achieve a good success rate for a particular ball gun configuration. As a result, it is very hard to tune this method to generalize well to different ball gun locations and orientations. On the other hand, our method generalizes well to changes on the ball trajectory and can be easily retrained if the ball gun configuration is significantly changed.

2.5 Epilogue

This chapter introduces new operators to learn and adapt probabilistic movement primitives in joint and in task space. The presented learning algorithm uses a prior distribution to increase the robustness of the estimated parameters. Using the proposed prior distribution over the ProMP parameters is an effective way to improve robustness and learn with few training instances while conserving enough flexibility in the model to learn the dependencies between the joints as more data becomes available.

This chapter also presents simple and fast operators to adapt movement primitives in joint and task space, making use of standard methods of probability theory. These operators were evaluated in the coffee task and table tennis task to adapt the learned movement primitive to the coffee machine position and the ball trajectory respectively. The presented operators to adapt movement primitives can be applied to any other robotic applications.

We tested the performance improvements due to table tennis specific advantages like re-planning and the prior over the hitting time. More importantly, we tested the improvements due to the presented learning algorithm and its ability to learn the joint correlations independently of the table tennis specific improvements. We show that the difference on the learning algorithm alone is enough to obtain a statistically significant improvement.

Unlike previous approaches to robot table tennis, our approach does not model the interaction between the racket and the ball. The reason why the presented method can successfully send balls to the opponent's side of the table is because the training data used to learn the movement primitive contains mostly successful examples. Thus, the behavior of successfully returning balls is completely learned from data.

A limitation of the presented training method is that it requires manual segmentation of the robot trajectories. Someone needs to specify where every movement primitive starts and ends in the demonstrated behavior. A better approach would be to consider the segmentation as another hidden variable and add it to the proposed EM inference algorithm. The problems of automatic segmentation and clustering should be considered in future work.



3 Reliable Real Time Ball Tracking for Robot Table Tennis

Robot table tennis systems require a vision system that can track the ball position with low latency and high sampling rate. Altering the ball to simplify the tracking, using for instance infrared coating, changes the physics of the ball trajectory. As a result, table tennis systems use custom tracking systems to track the ball. These custom tracking systems are based on heuristic algorithms applied to RGB images captured with a set of cameras respecting the real time constraints. However, these heuristic algorithms often report erroneous ball positions, and the table tennis policies typically need to incorporate additional heuristics to detect and possibly correct outliers. In this chapter, we propose a vision system for object detection and tracking that focus on reliability while providing real time performance. Our assumption is that by using multiple cameras, we can find and discard the errors obtained in the object detection phase by checking for consistency with the positions reported by other cameras. We provide an open source implementation of the proposed tracking system to simplify future research in robot table tennis or related tracking applications with strong real time requirements. We evaluate the proposed system thoroughly in simulation and in the real system, outperforming previous work. Furthermore, we show that the accuracy and robustness of the proposed system increases as more cameras are added. Finally, we evaluate the table tennis playing performance of the method presented in Chapter 2 using the proposed vision system. We measure a slight increase in performance compared to a previous vision system even after removing all the heuristics previously present to filter out erroneous ball observations.

3.1 Introduction

Game playing has been a popular technique to compare the performance of different artificial intelligence methods between themselves and against humans. Examples include board games like Chess [34] and Go [35] as well as sports like robot-soccer [36]. Table tennis has been



Figure 3.1.: Robot table tennis setup used to evaluate the proposed methods. We use four cameras attached to the ceiling to track the position of the ball. The robots used are two Barrett WAM robot arms capable of high speed motion, with seven degrees of freedom like a human arm.

used regularly as a robot task to evaluate the performance of ad-hoc techniques [20], imitation learning [9], reinforcement learning [10] and other techniques in a complex real time environment.

In order to play table tennis, a robotic systems needs reliable information about the ball trajectory with low latency and high sampling frequency. Commercial tracking system like VICON can provide reliable 3D positions with high sampling frequencies, but it requires attaching IR reflective markers to the objects to track. Table tennis balls are very light, and it is not possible to attach a IR marker or even coat the ball surface with IR reflective paint without changing the physics of the ball trajectory. For this reason, robot table tennis approaches typically use software based solutions that take images from a set of video cameras and estimate the 3D position of the ball.

Tracking systems for table tennis balls use fast heuristics to detect the ball respecting the real time constrains required by table tennis systems. These heuristics typically look for round objects and use color information of table tennis balls. Although these heuristics work well most of the time, assuming that the reported ball positions are always correct before the 3D triangulation will result in a number of outliers that increases as more cameras are used in the tracking system.

As a result, robot table tennis systems need to incorporate outlier detection [37] techniques on the reported 3D positions using for example physical models of the ball trajectory [23]. This is unfortunate, since it results in effort duplication and reduces the interest of the machine learning community to work on real robot table tennis platforms.

We propose a simple and efficient framework for object tracking. The proposed framework is tested on a robot table tennis setup and compared with previous work [11]. Unlike previous work, we focus on the reliability of the system without the use of any strong assumptions about the object shape or the physics of the flying ball. To evaluate the performance of the algorithm in setups with different amount of cameras, we use a simulation environment. We show that adding more cameras helps to increase the robustness and the accuracy of the proposed system. In the real system, we evaluate the error distribution of the proposed system and compare it with previous work [11]. We show that the proposed framework is clearly superior in accuracy and robustness to outliers. Finally, we evaluate the system by using the robot table tennis method introduced in Chapter 2, that was designed to be used with the RTBlob vision system [11]. We remove all the heuristics to detect and remove outliers and still obtain a slight improvement of performance using the proposed vision system. Figure 3.1, shows the real robot setup used on the experiments, executing the method proposed in Chapter 2 with the vision system proposed on this chapter.

Although we focus on robot table tennis due to its particular real time requirements, we use machine learning techniques for the object detection part that can be trained to track different kind of objects. A user only needs to label a few images by placing a bounding box around the object of interest and train the system with the labeled images.

Contributions

We provide a open source implementation [38] of a simple table tennis ball tracking system that focuses on reliability and real time performance. The implementation can be used to track different objects simply by retraining the model. The provided open source implementation will enable researchers working on robot table tennis or related real time object tracking applications

to focus their efforts into better strategies or models, instead of devising strategies to determine which observations can be trusted and which can not.

We evaluate the proposed system in simulation and in a real robot table tennis platform. In simulation, we show that increasing the number of cameras results in higher reliability. On the real system, we evaluate an existing robot table tennis strategy using the proposed vision system with four cameras attached to the ceiling. The heuristics used to discard outliers on the ball observations were removed, while obtaining a slightly increase on playing performance. In addition, we provide latency times for the different experiments to show the proposed system can deliver real time performance even with a large number of cameras.

Related Work

Ball tracking systems take an important role in almost all popular ball based sports to aid coaches, referees and sport commentators. Examples include soccer [39, 40], basketball [41], tennis [42], etc. There are multiple systems designed for tracking table tennis balls, some of which include real time considerations or were designed for robot table tennis. Table tennis is a fast game, and that makes it a hard robot problem to tackle. A smashed ball takes about 0.1 seconds to reach the other end of the table, and even at beginner level, it takes about 1 second for the ball to reach the opponent. Considering that robot arms like the Barrett WAM are much slower than a human arm, the amount of time available to make a decision of how and where to move before it is too late to reach the ball is low even to play at a beginner level. As a result, a vision system for robot table tennis needs to provide a high sampling rate with a low latency to provide as much information as possible as early as possible.

RTblob [11] was one of the first vision systems used for robot table tennis applications. It uses four color cameras to track the position of the ball. To find the position of the ball on an image, this system uses a reference orange color and convolves the resulting image with a circular pattern using the fast Fourier Transform for efficiency. Instead of using the four cameras to output one single 3D ball position, this system uses two pairs of two cameras. As a result, if all the cameras are seeing the ball, two 3D position are estimated. In this system, it is not clear how to use more cameras or how to determine which observations are reliable or not. Each table tennis policy that used RTBlob had to implement its own outlier rejection heuristics to determine which produced ball observations were reliable.

There are several other vision systems for robot table tennis, but none of them addresses the problem of how to deal with mistakes from the object detection algorithm in the images. Quick MAG 3 [43] uses a motion blur and a ball trajectory model to estimate and predict ball trajectories. In [44], a background model is used to extract the position of the ball. The detected blobs are filtered out according to their area, circularity and other factors. Finally, a ball model is used to predict the ball trajectory. In [45], the authors focus on the physical models useful to predict the ball trajectory, and use these models for humanoid robot table tennis.

A common design pattern for all the discussed table tennis vision systems, is that the object detection part consists on multiple heuristics based on background subtraction, color templates and basic shape matching on blobs. These approaches tend to work well in practice and satisfy the real time requirements of robot table tennis. Machine learning based methods, on the other hand, have been typically not fast enough for real time robotics, but have the potential to be easily adapted to track different objects by simply labeling new images. Heuristic based methods are hard to adapt to track different objects. We discuss two different machine learning



Figure 3.2.: Ball detection with a Mobilenet deep network architecture using the Single Shot Detection (SSD) method. Note that the SSD method finds the location of the ball in all the images with relatively good accuracy. However, we obtain an average of 15 ball observations per second on a four camera setup, not efficient enough for a highly dynamic task like robot table tennis.

approaches that can be used to find the position of the ball in the image, showing that it is possible to obtain the real time performance required for robot table tennis.

3.2 Reliable Real-Time Ball Tracking

End-to-end systems are an appealing strategy for system design in machine learning research, because it makes less assumptions about how the system works internally. For our table tennis vision setup, an end-to-end system should receive the input images from all the cameras and output the corresponding ball location in 3D cartesian coordinates. However, such an end-to-end solution would have a number of disadvantages for our table tennis setup. For example, adding new cameras or moving around the existing cameras would require to re-train the entire system from scratch.

We divide our vision system into two subsystems. The object detection subsystem that outputs the ball positions in pixel space for each image, and the position estimation subsystem that outputs a single 3D position of the ball based on a camera calibration procedure. To add new cameras we only need to run the calibration procedure, and moving existing cameras requires only the re-calibration of the moved cameras.

First, we discuss about different methods used in the machine learning community to detect general objects in images. In particular, we discuss about object detection and semantic segmentation methods, and their advantages and disadvantages for the ball detection problem. We show that although both methods can successfully find table tennis balls in an image, the semantic segmentation method can be used with smaller models, achieving the required real time execution requirements we need for robot table tennis.

Subsequently, we discuss how to estimate a single 3D ball position from multiple camera observations. We focus particularly on how to deal with erroneous estimates of the ball position in pixel space, for example, when the object detection method fails and reports the location of some other object. We analyze the algorithmic complexity of the proposed methods and we also provide execution times in a particular computer for setups with different number of cameras.

3.2.1 Finding the Position of the Ball in an Image

The problem of detecting the location of desired objects in images has been well studied in the computer vision community [46]. Finding bounding boxes for objects in images is known as object detection. In [47], a method called Single Shot Detection (SSD) was proposed to turn a convolutional neural network for image classification into an object detection network. An important design goal of the SSD method is computational efficiency. In combination with a

Algorithm 6 Finding the set of pixels of an object.

Input: A probability image \mathbf{B} , and a high and low thresholds T_h and T_l .**Output:** A set of object pixels O

```
1:  $(a, b) = \arg \max_{(a,b)} B_{ab}$ 
2: if  $B_{ab} < T_h$  then
3:   return  $\emptyset$ 
4: end if
5:  $O \leftarrow \{(a, b)\}$ 
6:  $q \leftarrow \text{Queue}(\{(a, b)\})$ 
7: while  $q$  is not empty do
8:    $x \leftarrow \text{pop}(q)$ 
9:   for each neighbors  $y$  of  $x$  do
10:    if not  $y \in O$  and  $B_y > T_l$  then
11:       $\text{push}(q, y)$ 
12:       $O \leftarrow O \cup \{y\}$ 
13:    end if
14:  end for
15: end while
16: return  $O$ 
```

relatively small deep network architecture like Mobilnet [48], designed for mobile devices, it can perform real time object detection for some applications.

Figure 3.2, shows example predictions of a Mobilnet architecture trained with the SSD method in a ball detection data set. Each picture shows a section of the image with the corresponding bounding box prediction. The resulting average processing speed using a GPU NVidia GTX 1080 was **60.2 frames per second** on 200 x 200 pixel resolution images. For a 4 camera robot table tennis setup, this would result in about **15 ball observations per second**. Unfortunately, for a high speed game like table tennis, a significantly higher number of ball observations is necessary. However, we consider important to mention the results we obtained with fast deep learning object detection techniques like the SSD method, because it can be used with our method for a different application where the objects to track are more complex and the required processing speeds are lower.

An alternative approach to find objects in images is to use a semantic segmentation method, where the output of the network is a pixelwise classification of the objects of interest or background. For example, [49] uses deep convolutional neural networks to classify every pixel in a street scene as one of 20 categories like car, person and road. To track the ball we only need two categories: Ball and Background. We consider background anything that is not a table tennis ball. Let us denote the resulting probability image as a matrix \mathbf{B} , where B_{ij} is a scalar denoting the probability that the pixel (i, j) of the original image corresponds to a ball pixel or not.

In order to find the actual set of pixels corresponding to the ball, we need some kind of threshold based algorithm that makes a hard zero/one decision of which pixels belong to the object of interest based on the obtained probabilities. We used a simple algorithm that consists of finding the pixel position (a, b) with maximum probability and a region of neighboring pixels with a probability higher than a given threshold.

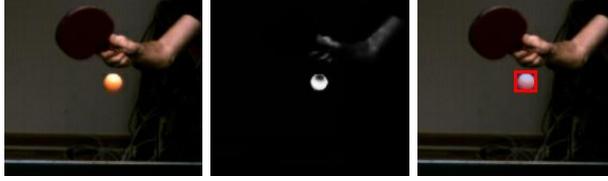


Figure 3.3.: Ball detection using a single convolutional unit in a semantic segmentation setting. The image on the left shows a section of a table tennis scene. The image on the center shows the probability image B representing the probability assigned to each pixel of being the ball. Dark means low probability and bright means high probability. The image on the right shows the detected ball position. This simple model can successfully find the ball in the image, and it is around 50 times faster than the SSD method.

Algorithm 6 shows the procedure to obtain the set of pixels corresponding to the ball from the probability image B . The procedure receives two threshold values T_h and T_l , that we call high and low threshold respectively. In Line 1, we find the pixel position (a, b) with maximum probability on the probability image B . If the maximum probability is lower than the high threshold value T_h we consider there is no ball in the image and return an empty set of pixels. Otherwise, Lines 5 to 15 find a region of neighboring pixels O around the maximum (a, b) with a probability larger than the low threshold T_l using a Breadth First Search algorithm. The center of the ball is computed by averaging the pixel positions in O .

The computational complexity of Algorithm 6 is linear on the number of pixels. If N_t represents the total number of pixels in the image and N_o the number of pixels of the object to track, the computational complexity of Line 1 alone is $O(N_t)$ and the complexity of the rest of the algorithm is $O(N_o)$. However, Line 1 can be efficiently implemented in a GPU, whereas the rest of the algorithm is harder to implement on a GPU due to its sequential nature. Given that $N_t \gg N_o$, we decided to use the GPU to execute Line 1 and implemented the rest of the algorithm in the CPU. In combination with the semantic segmentation approach using a single convolutional unit, we obtained a throughput about **50 times faster than the SSD method** for our ball tracking problem.

Figure 3.3 shows the semantic segmentation results for the table tennis problem using a single convolutional unit with a 5x5 pixels filter size. The picture on the left shows a section of the image captured with our cameras. The picture on the center shows the probability image B assigned by the model to each pixel as being the ball, where white means high probability and black low probability. The picture on the right shows a bounding box that contains all pixels in O returned by Algorithm 6. Note that all the objects in the scene that are not the ball are assigned by the model a very low probability of being the ball, and most of the pixels of the ball are assigned a high probability of being the ball. Actually, the only object that can still be seen not completely dark in the probability image is the human arm, because it has a similar color to the ball in comparison with the rest of the scene.

The throughput of the single 5x5 convolutional unit is about 50 times higher than the throughput of the SSD method on the same hardware with our implementations. As a result, we decided to use the single convolutional unit as the ball detection method, achieving the necessary ball observation frequency and accuracy for robot table tennis. In Section 3.3, we analyze in detail the performance and accuracy of the single convolutional unit. In addition, we compare the accuracy of our entire proposed system with the RTBlob vision system [11] and evaluate the playing performance of an existing robot table tennis method [9] using the proposed system.

Algorithm 7 Remove outliers by finding the largest consistent subset of 2D observations for stereo vision.

Input: A set of 2D observations and camera matrix pairs $S = \{\{x_1, P_1\}, \dots, \{x_k, P_k\}\}$, and pixel error threshold ϵ .

Output: A subset $\hat{S} \subset S$ of maximal size without outliers.

```

1:  $\hat{S} \leftarrow \emptyset$ 
2: for  $i \in \{1, \dots, k-1\}$  do
3:   for  $j \in \{i+1, \dots, k\}$  do
4:      $candidate \leftarrow \text{stereo}(\{P_i, P_j\}, \{x_i, x_j\})$ 
5:      $S_{ij} \leftarrow \emptyset$ 
6:     for  $k \in \{0, \dots, k\}$  do
7:        $\hat{x}_k \leftarrow \text{project}(candidate, P_k)$ 
8:        $p\_err \leftarrow \|x_k - \hat{x}_k\|_2$ 
9:       if  $p\_err < \epsilon$  then
10:         $S_{ij} \leftarrow S_{ij} \cup \{x_k, P_k\}$ 
11:       end if
12:     end for
13:     if  $|S_{ij}| > |\hat{S}|$  then
14:        $\hat{S} \leftarrow S_{ij}$ 
15:     end if
16:   end for
17: end for
18: return  $\hat{S}$ 

```

3.2.2 Robust Estimation of the Ball Position

Once we have the position of the ball in pixel space in multiple calibrated cameras, we proceed to estimate a single reliable 3D ball position. The process to obtain an estimation of the 3D position of an object given its pixel space position in two or more cameras is called stereo vision. For an overview in stereo vision refer to [50].

Previous work on vision systems for robot table tennis focused on providing real time 3D ball positions without considering the possibility of errors in the ball detection algorithm. As a result, robot table tennis systems like [23] had to include outlier detection techniques on the 3D observations using for example physics models. When an observation deviated significantly from the position predicted with the physics model, the observation was discarded. The main drawback of this approach is that we have to discard a complete 3D observation even if only one camera made a mistake localizing the ball in pixel space and the rest of the cameras provided a correct observation. Adding more cameras to such a system would end up in discarding more ball observations instead of improving the entire system reliability. Instead, our approach consists on detecting outliers on the 2D pixel space ball positions individually for each camera. If a small set of cameras detect the ball position incorrectly, but a different set of cameras detect the ball correctly, we can still provide a 3D observation using the correct set of cameras only.

Outlier detection algorithms have received a great amount of attention by the scientific community. The more common approaches consist on detection atypical values in a unsupervised

or supervised fashion [51], modeling a distribution of the typical values in the former case or detecting outliers as a classification problem on the latter. These kind of approaches are not useful in our case, since the distribution of the ball position in the image space is not expected to be different from the position of other objects that could be mistaken by the ball. Instead, we focus on outlier detection by consensus [52]. Examples of algorithms for outlier detection by consensus include RANSAC [53], MLESAC [54], NAPSAC [55] and USAC [56]. All these algorithms consist on taking a random subsample of the observation set, fitting the parameters of the model of our interest only on the subsample set, and keeping best model parameters according to some optimality criteria. The differences between all these algorithms lies on the criteria to select the subsets of observations and to decide what is the best model found so far. The RANSAC (Random Sample Consensus) algorithm [53] is probably the most popular method of outlier detection by consensus used in the computer vision community [52]. The random subsamples are typically taken uniformly at random. A model hypothesis is generated by fitting the model with the random subsample. An observation is considered consistent with the hypothesis if the error is smaller than a parameter ϵ , and the hypothesis with the largest number of consistent observations is selected as the best one. The MLESAC [54] algorithm, maximizes the likelihood of the inlier and outlier sets under a probabilistic model instead of maximizing the size of the consistent set. The NAPSAC [55] algorithm uses the observation that inliers often are closer to each other than outliers. Instead of selecting the subsample set uniformly at random, the NAPSAC algorithm generates a subsample set taking into consideration the distance between the observations.

We propose a consensus-based algorithm that maximizes the size of the support set like RANSAC does, but instead of selecting the subset of observations at random, we try all possible camera pairs, guaranteeing to find a solution if it exists. In the rest of this section, we explain in detail our consensus-based algorithm to find a single the 3D position from several 2D pixel space observations containing outliers. We assume we have access to two functions `project` and `stereo` available from an stereo vision library, as well as the projection matrices P_i for each camera i . Given a 3D point X , the function $x_i = \text{project}(X, P_i)$ returns the pixel space coordinates x_i of projection of X in the image plane of camera i . For the stereo vision method, we are given a set of pixel space points $\{x_1, \dots, x_k\}$ from k different cameras and their corresponding projection matrices $\{P_1, \dots, P_k\}$, and obtain an estimate of the 3D point X by

$$X = \text{stereo}(\{x_1, \dots, x_k\}, \{P_1, \dots, P_k\}).$$

Intuitively, the function `stereo` finds the point X that minimize the pixel re-projection error given by

$$L(X) = \sum_k \text{dist}(x_k, \text{project}(X, P_k)),$$

where `dist` is some distance metric like euclidean distance.

We assume that from a set S of pixel space ball observations reported by the vision system, some of the observations $\hat{S} \in S$ are correctly reported ball positions and the rest of the reported observations $\bar{S} = S - \hat{S}$ are erroneously reported ball positions. We call \hat{S} the inlier or support set and \bar{S} the outlier set. We would like to find the 3D ball position X that minimizes $L(X)$ using only the support set \hat{S} .

We define a set of pixel space observations as consistent if there is a 3D point X such that $L(X) < \epsilon$, where ϵ is a pixel space error tolerance. We estimate \hat{S} by computing the

largest subset of S that is consistent. The underlying assumption is that it should be hard to find a single 3D position that explains a set of pixel observations containing outliers. On the other hand, if the set of observations contains only inliers, we know it should be possible to find a single 3D position X , the cartesian position of the ball, that explains all the pixel space observations.

Algorithm 7 shows the procedure we use to obtain the largest consistent set of observations. Note that we need at least two cameras to estimate a 3D position. Our procedure consists in trying all pairs of cameras (i, j) , estimating a candidate 3D position only with those two observations, and subsequently counting how many cameras are consistent with the estimated candidate position. If c represents the number of cameras reporting a ball observation, the computational complexity of this algorithm is $O(c^3)$.

For a vision system of less than 30 cameras, we obtained real time performance even using a sequential implementation of Algorithm 7. Nevertheless, it is easy to parallelize Algorithm 7. Note that the outermost two for loops can be run independently in parallel. In Section 3.3, we evaluate the real time performance and the accuracy of the 3D estimation simulating scenarios with different number of cameras and probability of outliers. Unlike previous robot table tennis systems that discard entire 3D observations using physics based models [23], we obtain improved accuracy and less dropped observations as the number of cameras on the vision system is increased. We also evaluate the error in the real system and compare it with the RTBlob method using the same experimental setup with four cameras, obtaining a higher accuracy and reliability.

3.3 Experiments and Results

We evaluate the proposed system in a simulation environment and in a real robot platform. In simulation, we measure the accuracy of the system as we increase the number of cameras and when we change the probability of obtaining outliers. We use the real robot platform to evaluate the interaction of all the components of the proposed system. In particular, we measure the accuracy and robustness of the proposed system and compare it with the RTBlob method. In addition, we evaluate the success rate of a method proposed in [9] to return balls to the opponent's court with the proposed vision system. We have a slightly higher success rate using the proposed vision system than using the RTBlob system even after removing all the outlier rejection heuristics implemented in [9].

3.3.1 Evaluation on a Simulation Environment

To evaluate the proposed methods in scenarios that include different number of cameras and probability of outliers, we use a simulation scenario. The advantage of evaluating in simulation is that we have access to exact ground truth data and we can easily test the robustness and accuracy of the system. In this section, we evaluate the robustness of the introduced procedure to find the 3D position of the ball from several unreliable pixel space observations. First, we want to evaluate the performance of Algorithm 7 independently of the rest of the system. In addition, we want to test the accuracy and running time of the algorithm for different amount of cameras and outlier rates.

The simulation for a scenario with c cameras and a probability of outlier p_o consists of the following steps: First, we generate randomly a 3D ball position X in the work space of the

c	Probability of Outliers p_o					
		1%	5%	10%	25%	50%
4	E	0.71 cm	0.85 cm	0.84 cm	0.79 cm	4.67 cm
	F	0.1%	0.5%	2.0%	9.7%	37.7%
8	E	0.52 cm	0.53 cm	0.59 cm	0.94 cm	6.84 cm
	F	0.0%	0.0%	0.0%	0.1%	4.5%
15	E	0.35cm	0.36 cm	0.37 cm	0.41 cm	4.72 cm
	F	0.0%	0.0%	0.0%	0.0%	0.02%
30	E	0.24cm	0.25 cm	0.25 cm	0.28 cm	0.35 cm
	F	0.0%	0.0%	0.0%	0.0%	0.0%

Table 3.1.: Estimation error (E) and failure probability (F) of the 3D position estimation procedure in the presence of outliers. A failure means that the system does not report any ball position at all because the maximum consistent set returned by Algorithm 7 consisted of less than two ball observations. Otherwise, the system return an estimated ball position and we report the distance in centimeters to the ground truth position. We simulate multiple scenarios with a different number of cameras and different probability of outliers. Note that as the number of cameras increases and the probability of obtaining outliers decreases the system becomes more reliable.

robot and project it to each camera using the calibration matrices. We add a small Gaussian noise with a standard deviation of 1.3 pixels to the projected pixel space position, because that is the average re-projection error reported by the camera calibration procedure. For each camera, we replace the obtained pixel space position by some other random position in the image plane with probability p_o . Subsequently, we attempt to obtain the 3D ball position with Algorithm 7. If it fails to obtain any position at all, we count it as a failure. Otherwise, we measure the error of the obtained position with the ground truth value X .

Table 3.1 shows the results for scenarios with a number of cameras ranging from 4 to 30 and probability of outliers ranging from 1% to 50%. For every combination of number of cameras and probability of outliers, we report the failure rate (F) and the error (E) between the ground truth position and the reported ball position. As the probability of outliers increases the error and failure rate increases as it is expected. Similarly, as more cameras are added to the system, the robustness of the system increases, obtaining smaller errors and failure rates. There are few entries in Table 3.1 that seem to contradict the trend to reduce the error as more cameras are introduced or the outlier rate drops. For example, for an outlier rate of 50% the error with four cameras is 4.67 cm whereas the error for eight cameras is 6.84 cm. Note however that the failure rate for four cameras is much higher than for eight cameras in this case.

Adding more cameras to the system improves accuracy and robustness. However, it also increases the computation cost. The cost of the image processing part grows linearly with the number of cameras, but can be run independently in parallel for every camera if necessary. Therefore, we will focus on the cost of the position estimation procedure as the number of cameras grows. As discussed in Section 3.2.2, the cost of the position estimation procedure in $O(c^3)$. Table 3.2 shows the run time in milliseconds of a sequential implementation of 7 in C++ in a Lenovo Thinkpad X2 laptop. For a target frequency rate of 200 observations per second we need a processing time smaller than 5 milliseconds. Note that even the sequential implementation of

Cameras	4	8	15	30	50
Run time (ms)	0.001	0.012	0.015	3.02	11.46

Table 3.2.: Run time in milliseconds of a sequential implementation of Algorithm 7 with respect to the number of cameras. For a system of up to 30 cameras, the sequential implementation of Algorithm 7 provides real time performance for more than 200 ball observations per second.

Algorithm 7 has the required real time performance for systems up to 30 cameras. In addition, Algorithm 7 can be easily parallelized if necessary as discussed in Section 3.2.2.

It is important to note that on a real system not all the cameras might be seeing the work space of the robot. For example, in the real robot setup we used four cameras but there are many parts of the work space that are covered only by two cameras, reducing the effective robustness of the system on those areas. However, the outlier rate of the image processing algorithms is below 1% in practice, and good results can be obtained using a small number of cameras as we discuss in the next section.

3.3.2 Evaluation on the Real Robot Platform

We evaluate the entire proposed system in the real robot platform and compare the performance to the RTBlob system presented in [11]. The evaluation on the real robot platform consisted of two experiments. First, we attach a table tennis ball to the robot end effector. We move the robot and use its kinematics to compute the position of the ball and use it as ground truth to compare against the ball positions obtained with the vision system. Finally, we evaluate the playing performance of the robot table tennis strategy introduced in Chapter 2 if we remove all the heuristics used to remove vision outliers.

We compare the performance of the proposed vision system with RTBlob [11]. The RTBlob system has been used for robot table tennis experimentation [23, 37]. In order to compare the accuracy of both systems, we need access to ground truth positions. We use the joint sensors of the robot and the robot kinematics to compute the Cartesian position of the robot end effector. We attach the ball to the robot end effector and use the Cartesian position computed with the joint measurements as ground truth.

Figure 3.4 shows a histogram of the error of the RTBlob method and the proposed method. We called the proposed system RT^2 in the figure, standing for Real Time Reliable Tracking. The error is computed as the distance between the position reported by the vision and the ground truth computed with the robot kinematics. Note that the proposed vision system outperforms the RTBlob method in terms of accuracy, but specially in terms of outliers. The distribution of errors for RT^2 concentrates the probability mass between 0 cm and 5 cm error. On the other hand, the error distribution of the RTBlob method is multimodal. The first mode corresponds to the scenario where all the cameras detected the ball correctly, and in this case the error mass is also concentrated below a 7 cm error threshold. The second mode shows a high probability of error between 25 cm and 30 cm, and it is likely to correspond to a scenario where one of the four cameras reported an incorrect ball position.

The total error of the proposed system on the real robot experiments is $1.99 \text{ cm} \pm 1.15 \text{ cm}$, which is comparable with the radius of the ball of 2 cm. During the execution of this experiment, the proposed system never reported any ball position whose error was larger than 10 cm. On the other hand, the RTBlob system reported errors on the order of tens of meters with probabil-

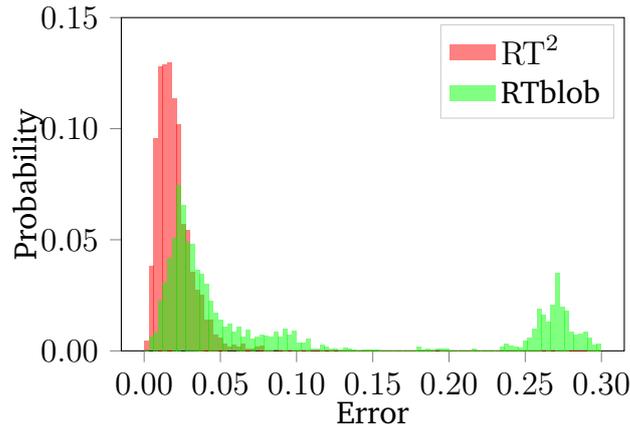


Figure 3.4.: Histogram of the error of the presented vision system and RTblob. We call the proposed vision system RT^2 , depicted in red in the histogram. The ball is attached to the robot end effector and the end effector position computed with the kinematics is used as ground truth. The accuracy of the proposed vision system is superior to the RTblob system. In terms of robustness to outliers, the proposed system (RT^2) outperforms the RTblob system as expected. The error distribution for our system is unimodal, whereas the RTblob system error is multimodal, reflecting the sensitivity of the RTblob system to the presence of outliers.

ity around 0.1%. As a result, the table tennis strategies that use the RTblob method have to incorporate strategies to filter outliers to work properly.

In the last part of this section, we present a final experiment where we use the proposed vision system to return table tennis balls with the robot to the opponent's court. We use a method presented in Chapter 2 that was originally designed to use the RTblob method as the vision system. The outliers produced by the RTblob vision system were detected using RANSAC by fitting a second order polynomial to a small set of initial observations, and subsequently using the initial position and velocity of the polynomial with a Kalman filter to reject as outlier any observation more than 3 standard deviations away from the mean predicted position.

We decided to remove the heuristics to filter outliers, accepting all ball observations as valid, and test the method with the proposed vision system. We define "success" as the robot hitting the incoming ball and sending it back to the opponent's court according to the table tennis rules. The average success rate using the RTblob vision system and all the outlier rejection heuristics was of **68 %**, whereas using the proposed vision system and no outlier rejection heuristics the average success rate was **70 %**. Given the variability of the table tennis performance between multiple experiments, we can not say that the improvement with the new vision system is statistically significant. However, we find remarkable that the success rate of robot table tennis method did not decrease after the outlier rejection heuristics were removed. We think that the slight improvement on the success rate by using the proposed vision system is due to the improved frame rate, that is about 3 times as high as that of the RTblob implementation provided by the authors [11].

3.4 Epilogue

This chapter introduces a vision system for robot table tennis focused on reliability and real time performance. The implemented system is released as an open source project [38] to facilitate its usage by the community. The proposed vision system can be easily adapted for different tracking tasks by labeling a new data set and training the object detection algorithm. For the object detection part, we evaluate two different approaches used commonly in the computer

vision community that are known for obtaining real time performance. We decided to use the simpler approach for tracking table tennis balls due to its high throughput.

For the position estimation procedure we proposed an algorithm that focuses on reliability, by assuming that some times the object detection methods will report wrong ball positions on the provided images. We evaluate the proposed method thoroughly in simulation and in the real robot platform. In simulation, we test the accuracy of the system under different probability of outliers and number of cameras. In the real system, we evaluate the complete proposed system in a four camera setup and compare it with the RTBlob vision system. We show that our system provides higher accuracy, and outperforms the RTBlob system in robustness to outliers. Finally, we test an existing technique to return table tennis balls to the opponent's court with our vision system. We removed all the outlier detection techniques used by the table tennis algorithm and obtained a small increase in success rate compared to the RTBlob system with all the outlier detection techniques present. We believe the proposed approach will help future research in robot table tennis by allowing the researchers to focus on the table tennis policies instead of techniques to deal with an unreliable vision system.



4 Real Time Trajectory Prediction Using Deep Conditional Generative Models

Data driven methods for time series forecasting that quantify uncertainty open new important possibilities for robot tasks with hard real time constraints, allowing the robot system to make decisions that trade off between reaction time and accuracy in the predictions. Despite the recent advances in deep learning, it is still challenging to make long term accurate predictions with the low latency required by real time robotic systems. In this chapter, we propose a deep conditional generative model for trajectory prediction that is learned from a data set of collected trajectories. Our method uses an encoder and decoder deep networks that maps complete or partial trajectories to a Gaussian distributed latent space and back, allowing for fast inference of the future values of a trajectory given previous observations. The encoder and decoder networks are trained using stochastic gradient variational Bayes. In the experiments, we show that our model provides more accurate long term predictions with a lower latency than popular models for trajectory forecasting like recurrent neural networks or physical models based on differential equations. Finally, we test our proposed approach in a robot table tennis scenario to evaluate the performance of the proposed method in a robotic task with hard real time constraints.

4.1 Introduction

Dynamic high speed robotics tasks often require accurate methods to forecast the future value of a physical quantity based on previous measurements while respecting the real time constraints of the particular application. For example, to hit or catch a flying ball with a robotic system we need to predict accurately and fast the trajectory of the ball based on previous observations that are often noisy and might include outliers or missing observations. Note that the time it takes to compute the predictions, called latency, is as important for the application as the accuracy in the prediction. In our previous example, the prediction of the future ball positions are only useful if the computation time is significantly faster than the ball itself.

Both physics based [57] and data-driven [8] models are used for trajectory forecasting. Physical models based on differential equations have been typically preferred to model and predict trajectories in high speed robotic systems [20], because they are relatively fast for predictions and are well studied models known to provide reasonably good predictions for many problems. However, in some applications like pneumatic muscle robots [58], the best known physics based models are not accurate enough to be useful for control. Even in cases where the physics are relatively well known, estimating all the relevant variables to model the system can be difficult. In table tennis, for example, estimating the spin of the ball in real time from images is hard. In addition, small lens distortion on the vision system makes the position estimates not equally accurate in all the robot work space, rendering the estimation of the initial position and velocity less accurate. A data-driven approaches, on the other hand, may have the potential to estimate the spin from its effect on the trajectory and ignore the lens distortion as long as it is present both at training and test time. However, popular data-driven methods for time series modeling

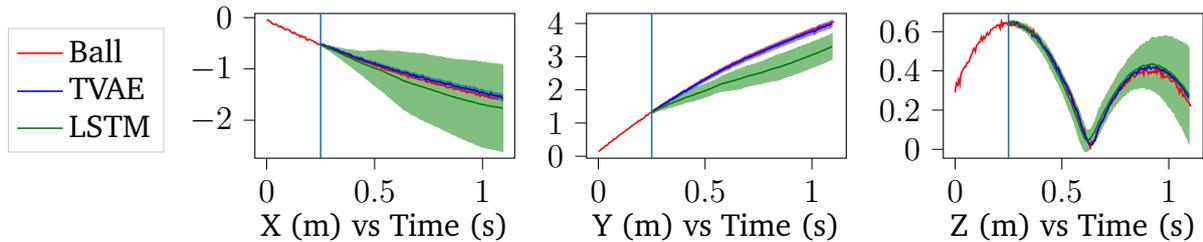


Figure 4.1.: Example of a ball trajectory in X, Y and Z (in meters) with respect to time (in seconds) and the respective prediction using LSTMs and the proposed method (TVAE). The observed ball trajectory is depicted in red, the prediction using a LSTM is depicted in green, and the prediction using the proposed model is depicted in blue. The shaded area corresponds to one standard deviation. Both models are given input observations until $t = 0.25$ s, and are asked to predict the rest of the ball trajectory. Note the cumulative error effect in the LSTM forecasting, the error grows very large as we predict farther into the future. The proposed method is more accurate for long term predictions than the LSTM.

like recurrent neural networks [7] and auto-regressive models [8] suffer from cumulative errors that render trajectory forecasting inaccurate as we predict farther into the future.

We propose a novel method for trajectory prediction that mixes the power of deep learning and conditional generative models to provide a data-driven approach for accurate trajectory forecasting with the low latency required by real time applications. We follow a similar approach to conditional variational auto-encoders [59], using a latent variable z to represent an entire trajectory, as well as an encoder and decoder network to map trajectories to and from the latent representation z . Our model is trained to maximize the conditional log-likelihood of the future observations given the past observations, using stochastic gradient descent and reparametrization for the optimization [60] of the variational objective. In addition, we introduce strategies to make the model robust to missing observations and outliers. We evaluate the proposed approach on a robot table tennis setup in simulation and in the real system, showing a higher prediction accuracy than a LSTM recurrent neural network [61] and a physics based model [45], while achieving real time execution performance. An open-source implementation of the method presented in this chapter is provided [62].

4.2 Trajectory Prediction

The term trajectory is commonly used in the robotics community to refer to a realization of a time series or Markov decision process. Formally, we define a trajectory $\tau_n = \{\mathbf{y}_{nt}\}_{t=1}^{T_n}$ of total length T_n as a sequence of multiple observations \mathbf{y}_{nt} , where the index t represents time and n indexes the different trajectories in the data set. For example, for the table tennis ball prediction problem, the observation \mathbf{y}_{nt} is a 3 dimensional vector representing the ball position at time index t of the ball trajectory n .

Each trajectory $\tau_n \sim P(\tau)$ is assumed to be independently sampled from the trajectory distribution $P(\tau)$. For trajectory prediction, we need to be able to predict the future trajectory based on previous observations. Let us use \mathbf{y}_t to denote the random variable representing the observation indexed by time t in any trajectory. Formally, the goal of trajectory forecasting is to compute the conditional distribution $p(\mathbf{y}_t, \dots, \mathbf{y}_T | \mathbf{y}_1, \dots, \mathbf{y}_{t-1})$, representing the distribution of the future values of a trajectory $\{\mathbf{y}_t, \dots, \mathbf{y}_T\}$ given the previous observations $\{\mathbf{y}_1, \dots, \mathbf{y}_{t-1}\}$. From this point on, we will use $\mathbf{y}_{1:t}$ to denote the set of variables $\{\mathbf{y}_1, \dots, \mathbf{y}_t\}$ compactly.

Trajectory or time series forecasting methods is an active research area of machine learning. Examples of popular approaches for time series forecasting include recurrent neural

networks [7], auto-regressive models [8, 63] and state space models [6]. Some of which include real time performance considerations [64]. All these approaches share in common that they model $p(\mathbf{y}_t | \mathbf{y}_{1:t-1})$, and use the factorization property of probability theory

$$p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1}) = \prod_{i=t}^T p(\mathbf{y}_i | \mathbf{y}_{1:i-1}),$$

to model and predict the entire future trajectory from past observations. Note that these models predict directly only one observation into the future y_{t+1} given the past $\mathbf{y}_{1:t}$. To make predictions farther into the future, the predictions of the model are fed back into the model as additional input observations. We will call an approach for trajectory forecasting “recursive” if it uses its own predictions as input to predict farther into the future.

An advantage of the recursive approaches is that they can model sequences of arbitrary length by design. It is always possible to make predictions with any given number of observations for any arbitrary number of time steps into the future. On the other hand, the recursive approaches have the disadvantage that errors are cumulative. Note that the predictions of the recursive approaches are fed back into the model. As a result, early small prediction errors can cause big forecasting errors as we try to predict farther into the future. For problems with high stochasticity like traffic [65], weather or stock market price prediction [8], where some of these models are commonly applied, it is reasonable to assume that no method will ever make almost exact long term predictions based only in previous observations.

However, for trajectory prediction in physical systems, where we are measuring all the relevant variables, we would expect long term prediction to be more accurate. For example, we know that the model used to generate the table tennis ball trajectories in simulation is deterministic once the initial state is set. However, the long term prediction error using an LSTM [61] recurrent neural network is about twice as large as using the physics based model. Figure 4.1 shows an example ball trajectory and the model predictions using an LSTM, depicted in green. The cumulative error effect for the LSTM model is easy to notice, specially in the Y coordinate, where the predictions deviates early from the ground truth ball trajectory depicted in red.

4.3 Deep Conditional Generative Models for Trajectory Forecasting

We have discussed how recursive methods like recurrent neural networks suffer from cumulative errors that render long term predictions less accurate. Therefore, our goal is to find a way to represent the conditional distribution $p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1})$ directly, in a way where the model predictions are not fed back into the model. In addition, we want to use a powerful model that can capture non linear relationships between the future and the past observations.

Note that for a fixed value t , we could model $p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1})$ directly as a regression problem. To deal with an arbitrary number of input observations t , we could train T different regression models

$$\{p(\mathbf{y}_{2:T} | \mathbf{y}_1), p(\mathbf{y}_{3:T} | \mathbf{y}_{1:2}), \dots, p(\mathbf{y}_T | \mathbf{y}_{1:T-1})\}$$

for all possible number of input observations, using at test time the model for the particular number of observations t given as input. Such an approach would not be able to exploit the correlation and redundancy between all the different regression models that are performing so similar tasks. In addition, this approach requires to train and store a large number of models.

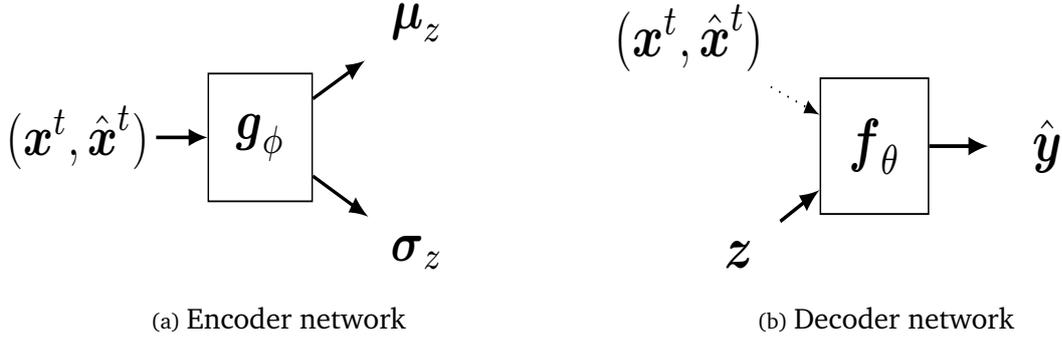


Figure 4.2.: Encoder and decoder networks for the proposed approach. The encoder network takes as input the past observations encoded in the variables $(\mathbf{x}^t, \hat{\mathbf{x}}^t)$ and produces a Gaussian distribution for the latent variable z with mean μ_z and standard deviation σ_z . The decoder network takes a sample z and the past observations and produces a trajectory estimate $\hat{\mathbf{y}}$ including both the future $\hat{\mathbf{y}}_{t:T}$ and the past $\hat{\mathbf{y}}_{1:t-1}$.

4.3.1 Deterministic Regression Using Input Masks

Note that for a fixed value t , we could model $p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1})$ directly as a regression problem. If we use a complex non-linear regression model such as a neural network, we can capture non linear relations between the past and future observations. To deal with a variable number of inputs t and outputs $T - t$, we use two auxiliary input variables \mathbf{x}^t and $\hat{\mathbf{x}}^t$ that represent a zero-padded input observations and an observation mask. Given a set of observations $\mathbf{y}_{1:t-1}$ we set $\mathbf{x}_{1:t-1}^t = \mathbf{y}_{1:t-1}$, $\mathbf{x}_{t:T}^t = \mathbf{0}$, $\hat{\mathbf{x}}_{1:t-1}^t = 1$ and $\hat{\mathbf{x}}_{t:T}^t = 0$. The variable \mathbf{x}^t , represents the observations seen so far, padding the non-observed part of the trajectory with zeros. Similarly, the variable $\hat{\mathbf{x}}^t$ represents a $\{0,1\}$ mask indicating which values were observed and which values were not. Using the auxiliary variables \mathbf{x}^t and $\hat{\mathbf{x}}^t$ we can make predictions with any number of input observations $t \in \{0, 1, \dots, T\}$ using a single regression model even if we have missing observations.

The proposed approach assumes a fixed maximum prediction horizon T for all trajectories. This is a limitation for our approach compared to all the recursive models, that can model trajectories of any duration. We trade the flexibility of being able to model trajectories of arbitrary duration for higher accuracy in the predictions and faster computation times. In Section 4.3.6, we discuss ideas to mix the power of the proposed method with recursive approaches to be able to make predictions of any arbitrary duration.

4.3.2 Capturing Uncertainty and Variability

Quantifying the uncertainty of the trajectory predicted by the model is important for decision making. A self-driving car, for example, could reduce the speed if there is high uncertainty about the trajectory of a pedestrian crossing the street. For real time systems, the ability to quantify uncertainty allows the agent to make decisions that compromise between accuracy and time to react. In robot table tennis, for example, the robot could wait for more ball observations if there is high uncertainty about the ball trajectory, but waiting too long will result in failure to hit the ball.

We capture uncertainty about the predictions of a trajectory τ_n using a latent variable \mathbf{z}^n , that can be mapped to a trajectory using a complex non-linear function, similarly to other deep generative models approaches [66] like variational auto-encoders. We assume that the fu-

ture observations $\mathbf{y}_{t:T_n}^n$ are independent given the latent variable z^n and the previous observation $\mathbf{y}_{1:t-1}^n$, and are distributed by

$$p(\mathbf{y}_{t:T}^n | \mathbf{y}_{1:t-1}^n, z) = \prod_{i=t}^{T_n} \mathcal{N}(\mathbf{y}_i^n | \hat{\mathbf{y}}_i^n, \Sigma_y), \quad (4.1)$$

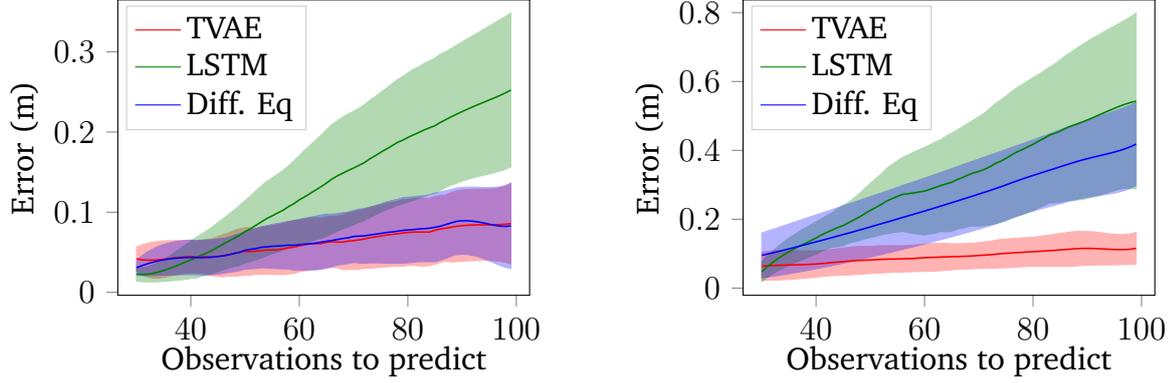
where $\hat{\mathbf{y}}^n$ is the estimated trajectory produced by the decoder network f_θ and Σ_y represents the observation noise learned also from data.

We want to emphasize that the limitation of a fixed prediction horizon T means that we do not have a principled approach to make predictions beyond T , but we can train our model with trajectories of any length T_n . If a trajectory τ_n with length $T_n < T$ is sampled in the training mini-batch, the model still predicts a trajectory of length T but the predictions with $t > T_n$ are not “penalized” as can be seen in (4.1). Note that the likelihood is evaluated for observations until T_n . If on the other hand $T_n \geq T$, we cut a random time interval $[t_a, t_b]$ such that $T = t_b - t_a$ for that particular mini-batch. When the same trajectory is drawn in a mini-batch later in the training process, a different random time interval $[t_a, t_b]$ is used. The training procedure is explained with greater detail in Section ??, the main message of this paragraph is that we can train our model with a data set of trajectories of any length. In Section 4.3.6, we mention possible ideas to make predictions beyond the time horizon T by mixing the advantages of recursive approaches with the method presented on this paper. We will drop the trajectory index n from the notation from this point forward to avoid notational clutter.

Our approach, based on variational auto-encoders, will use an encoder and decoder network to make predictions about the future value of a trajectory. The decoder network, depicted in Figure 4.2b, takes as input the previous observations $\mathbf{y}_{1:t-1}$ represented by $(\mathbf{x}^t, \hat{\mathbf{x}}^t)$ as well as the latent variable z that encodes one of the possible future trajectories. The output of the decoder network $\hat{\mathbf{y}}$ contains the predicted value for the future observations $\mathbf{y}_{t:T}$. We also use an encoder network g_ϕ that produces the variational distribution $q_\phi(z | \mathbf{y}_{1:t})$. The encoder network encodes a partial trajectory with observations $\mathbf{y}_{1:t}$ to the latent space z .

4.3.3 Inference

At prediction time, we typically want to draw several samples of the future trajectory conditioned on the previous observations $p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1})$. To do so, we compute first the latent space distribution $p(z | \mathbf{y}_{1:t-1})$ by passing the given observations through the encoder network g_ϕ . Figure 4.2a shows a diagram of the encoder network. Given the previous observations, the encoder provides us with a mean $\boldsymbol{\mu}_z$ and standard deviation $\boldsymbol{\sigma}_z$ vector for the latent variable z . Subsequently, we sample several values $z^l \sim \mathcal{N}(z | \boldsymbol{\mu}_z, \boldsymbol{\sigma}_z)$. Each sample z^l and the previous ball observations are passed through the decoder network to obtain a sample future trajectory. The inference process at prediction time is therefore very efficient. It requires a single pass through the encoder and the decoding process for every sample z^l can be done in parallel. In contrast with recurrent neural networks, the prediction process can be easily parallelized, allowing fast execution even for relatively large deep learning models.



(a) Prediction error in simulation

(b) Prediction error in the real system

Figure 4.3.: Distribution of the error in the test set for simulated data and real data as a function of the number of observation to predict into the future. The error is measured as the distance between the predicted trajectory and the ball trajectory measured with the cameras. For each model, the thick line represents the mean error and the shaded area is one standard deviation. Note that in simulated data our model performs as well as the differential equation based prediction, which was the model used in simulation and therefore is the best we can get. In real data, our model outperforms both the LSTM and differential equation models, specially as we predict farther into the future.

4.3.4 Training Procedure

The conditional likelihood using the latent variable \mathbf{z} is given by

$$p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1}) = \int p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1}, \mathbf{z}) p(\mathbf{z} | \mathbf{y}_{1:t-1}) d\mathbf{z}, \quad (4.2)$$

with $p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1}, \mathbf{z})$ given by (4.1). We use the decoder network \mathbf{g}_ϕ to compute $p(\mathbf{z} | \mathbf{y}_{1:t-1})$. In many applications, it is important to make sure the latent variable encodes all the relevant information about the previous observations, in which case the decoder distribution $p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1}, \mathbf{z}) = p(\mathbf{y}_{t:T} | \mathbf{z})$. We present the math of the model without making the previous assumption of conditional independence for generality. Incorporating the conditional independence assumption is trivial: simply ignore the input $(\mathbf{x}^t, \hat{\mathbf{x}}^t)$ when evaluating the decoder network \mathbf{f}_θ . In the experimental section, we compare the results with and without assuming conditional independence for the table tennis ball prediction problem, showing a slight improvement in generalization performance using the conditional independence assumption. The integral required to evaluate the conditional likelihood is intractable. We follow the approach used for Conditional Variational Auto-Encoders [59] (CVAE), optimizing instead a variational lower bound on the conditional log likelihood given by

$$\begin{aligned} \log p_\theta(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1}) &\geq -\text{KL}(q_\phi(\mathbf{z} | \mathbf{y}_{1:T}) || q_\phi(\mathbf{z} | \mathbf{y}_{1:t-1})) \\ &+ \mathbb{E}_{q_\phi(\mathbf{z} | \mathbf{y}_{1:T})}[\log p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t}, \mathbf{z})], \end{aligned} \quad (4.3)$$

where $q_\phi(\mathbf{z} | \mathbf{y}_{1:T})$ is the variational distribution given by

$$q_\phi(\mathbf{z} | \mathbf{y}_{1:T}) = \prod_{k=1}^K \mathcal{N}(z_k | \mu_z^k, \sigma_z^k),$$

with μ_z and σ_z produced by the encoder network and K is the size of the latent vector \mathbf{z} . The derivation of this objective function is presented in the supplementary material. The first term

of the objective keeps the distributions of z for partial trajectory and complete trajectories close. The second term forces the latent representation to be a good predictor for the future trajectory. The KL divergence term can be computed in closed form since both $q_\phi(z | \mathbf{y}_{1:T})$ and $q(z | \mathbf{y}_{1:t-1})$ are Gaussian distributions. The expectation is approximated with Montecarlo by sampling z from the variational distribution $q_\phi(z | \mathbf{y}_{1:T})$. Note that the only difference between optimizing the second term on (4.3) and optimizing (4.2) is that the expectation is computed over a complete or a partial trajectory respectively. This difference is key to compute the expectation using Montecarlo. The distribution over z using partial trajectories is typically too broad to be efficiently and accurately approximated using Montecarlo, specially when the cut point t is small.

Similarly to other deep generative models like variational auto-encoders, the lower bound on (4.3) can be optimized using stochastic gradient descent. The “reparametrization trick” [60] is used to compute gradients. We provide an open source implementation of the proposed method available in [62]. The training set consists of a set of trajectories τ_n each of a possibly different length T_n . When we sample mini-batches to train our model, we randomly select a cut point t for the trajectory τ_n with $0 < t \leq \tau_n$, and compute the lower bound for $p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1})$ using the particular cut point t . That way, our model will learn to make predictions for any number of given observations, including an empty set of observations. Finally, to make our model more robust to missing observations or outliers, we can randomly generate missing observations and outliers for the previous observations $\mathbf{y}_{1:t-1}$ in each of the trajectories included in the training mini-batch. To generate outliers we simply replace an observation with a random value within the domain of the input.

4.3.5 Network Architecture

The approach presented on this paper can be used with any regression method for \mathbf{f}_θ and \mathbf{g}_ϕ as long as we can compute derivatives $\frac{d\mathbf{f}_\theta}{d\theta}$ and $\frac{d\mathbf{g}_\phi}{d\phi}$. We used neural networks for this purpose in our experiments. We think that convolutional network architectures have great potential for time series or trajectory forecasting applications, since observations close in time are typically more strongly correlated than observations farther apart. Note that convolutional architectures have also been quite successful on image recognition applications, where pixels spatially close are typically also more strongly correlated.

For the experiments on this paper, we only used two layer architectures with dense connections for simplicity. The number of neurons on the hidden layer and the dimensionality of the latent space z were selected by testing multiple powers of two and selecting the hyper-parameters with better performance on the validation set.

4.3.6 Predicting Beyond the Horizon T

There may be many applications where making predictions arbitrarily far into the future is very important. Suppose for example that you use the presented approach to learn a forward model of a robot, and the goal is to use it to train a reinforcement learning agent in simulation. In such a case it is important to be able to make predictions much farther into the future, even at the expense of loosing accuracy.

The presented approach could be extended with ideas from recursive approaches to make predictions far into the future. The simplest recursive idea would be to use our model in an auto-regressive way, as it would require no change to the math or software implementation. In auto-regressive mode, we would simply use the predictions of our model as input observations. A possibly better approach would be to use a state space model or recurrent neural network over the latent variable z representing block of observations. This approach would require to modify the encoder to receive the latent representation of the previous block $i - 1$ of T observations in addition to the observations seen so far on the current block i . The encoder distribution would be therefore represented as $p(z_i | z_{i-1}, \mathbf{y}_{iT:iT+t})$. We do not explore any of these options in this paper but consider evaluating these approaches important future work.

4.4 Experiments

We evaluate the proposed method to predict the trajectory of a table tennis ball in simulation and in a real robot table tennis system. Predicting accurately the trajectory of a table tennis ball is difficult mostly because the spin is not directly observed by the vision system [67], but is significant due to the low mass of the ball.

We measure the prediction error and the latency, both important factors for real time robot applications. We use “TVAE” (Trajectory Variational Auto-Encoder) to abbreviate the name of the proposed method. We compare the results with an LSTM and the physics-based differential equation prediction. For the differential equation method, we use the ball physics proposed in [20]. This physics model considers air drag and bouncing physics but ignores spin. To estimate the initial position and velocity, we use the approach proposed in [45], that consists on fitting a polynomial to the first n observations and evaluating the polynomial and its derivative in $t = 0$. We tried different values of n and chose the one with better results.

4.4.1 Prediction Accuracy in Simulation

The simulation uses the same differential equation we used by on physic based model. The results should be optimal for the physics-based model on simulation, where the only source of error is the initial position and velocity estimation from noisy ball observations. To simulate the average position estimation error of the vision system [67] in simulation, we added Gaussian white noise with a standard deviation of 1 cm.

We generated 2000 ball trajectories for training, and another 200 for the test set. Figure 4.3a shows the prediction error (mean and standard deviation) in simulation over the test set. Note that the error distribution of the proposed method and the physics based model is almost identical, which is remarkable provided that the physics model used for the simulator and the differential equation predictor are the same. The results of the LSTM are slightly better than the proposed model for the first 10 observations into the future, but the error for long term prediction is about three times as large as the error for the physics or the proposed model.

4.4.2 Prediction Accuracy in the Real System

The real system consists of four RGB cameras taking 180 pictures per second attached to the ceiling. The images are processed with the stereo vision system proposed in [67], obtaining

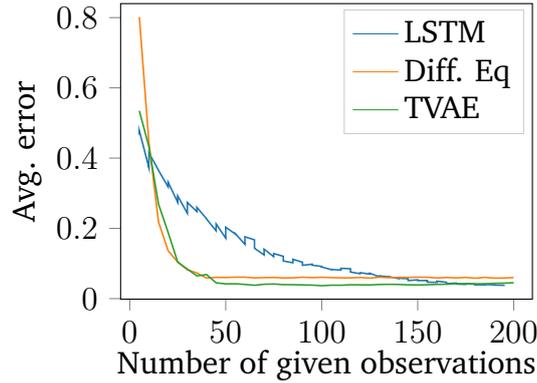


Figure 4.4.: Prediction error on simulated data as a function of the number of given observations. The error for the proposed model and the physics based model converge with between 30 and 50 observations, whereas the LSTM model needs between 100 and 150 observations to obtain a similar error rate.

estimations of the position of the ball. There are several issues that make ball prediction harder on the real system: There are missing observations, the error is not the same in all the space due to the effects of lens distortion, and the ball spin can not be observed directly. We used the vision system to collect 614 trajectories for the training set and 35 trajectories for the test set. Figure 4.3 shows the prediction error on the test set as a function of the number of observations to predict, feeding the model the first 30 observations. For each model, we present the average error as a thick line and shade the area around one standard deviation. Note that in the real system, our model outperforms the long term prediction accuracy of the other models. The LSTM, as expected, is very precise at the beginning but starts to accumulate errors and becomes quickly less accurate. The physics based system is less accurate than the proposed model, but is more accurate than the LSTM. The reason is that the physics model without spin is a good approximation in cases where the spin of the ball is very low.

4.4.3 Number of Input Ball Observations

The trajectory prediction task consists on estimating the future trajectory $\mathbf{y}_{t:T}$ given the past observations $\mathbf{y}_{1:t-1}$. Accurate predictions with a relatively low number of input observations t is important to allow for reaction time for the robot. Figure 4.4 shows the average prediction error over the entire ball trajectories as we vary the number of input observations t . Note that the prediction error converges for the proposed model with t between 40 and 50 observations. Similarly for the physics based model. On the other hand, the LSTM error converges after approximately 150 input observations, allowing a very low reaction time for the robot.

4.4.4 Robot Table Tennis

The robot table tennis approach presented in Chapter 2, consists of learning a Probabilistic Movement Primitive (ProMP) from human demonstrations and subsequently adapt the ProMP to intersect the trajectory of the ball. The trajectory of the ball must be represented as a probability distribution for three main reasons: First, the initial time and duration of the movement primitive are computed by maximizing the likelihood of hitting the ball. Second, the movement primitive is adapted to hit the ball using a probability distribution by conditioning the racket distribution to intersect the ball distribution. Third, to avoid dangerous movements, the robot does

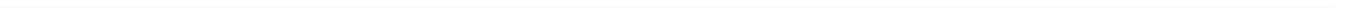
not execute the ProMP if the likelihood is lower than a certain threshold. All these operations would not work if only the mean ball trajectory prediction is available.

We modified the ProMP based policy to use the proposed ball model. To compute the ball distribution we took 30 trajectory samples from our model and computed empirically its mean and covariance. We obtained a hitting rate of **98.9%**, improving slightly the hitting rate of **96.7%** obtained using a ProMP as well for the ball model. One important difference is that we do not need to retrain the ball model every time the ball gun position or orientation changes. Using a ProMP as a ball model is only accurate if all the trajectories are very similar. Whereas our approach can accurately predict ball trajectories with high variability. This experiment also shows that the presented approach can be used in a system with hard real time constraints. Our system can infer the future ball trajectory from past observations with a latency between **8 ms** and **10 ms**.

4.5 Epilogue

We introduce a new method to make prediction of time series with neural networks, using a Gaussian distributed latent variable that encodes different trajectory realizations. We can draw trajectory samples from the learned trajectory distribution conditioned on any arbitrary number of previous observations. The proposed method is suitable for real time performance applications such as robot table tennis. We discussed why our method does not suffer from the cumulative error problem that popular time series forecasting methods such as LSTM have, and showed empirically that our method provides better long term predictions than other competing methods on a ball trajectory prediction task.

One of the main limitations of the proposed method for general time series forecasting is the fixed duration T . For robotics applications, for example, we might want to learn a dynamical system from real robot data and use it to learn a task in simulation, requiring simulating sequences of arbitrary length. Perhaps the best way to extend the proposed model to support sequences of arbitrary length is to mix it ideas of recursive methods. The simplest approach, which does not require any modification to the proposed method, would be to use an auto-regressive TVAE. An auto-regressive TVAE would simply use its own predictions as inputs to predict trajectories even farther into the future. A slightly more complicated and perhaps more promising approach, would mix a recurrent neural network such as an LSTM of a TVAE. The encoder of a RNN-TVAE would take as input the latent variable z of the previous block of length T and the partial observations of the current block to generate the latent space distribution of the current block. As we already discussed, LSTMs are very accurate when the number of steps to look into the future is not too large, making a mixture of TVAEs and LSTMs attractive to predict a few block ahead into the future with relatively high accuracy.





5 Conclusions and Future Work

In this thesis, we use robot table tennis as an inspiring application to explore probabilistic learning techniques for robotics. We review existing techniques like Probabilistic Movement Primitives, extending the available operators and improving the learning procedure. We also introduce a new method based on variational auto-encoders for trajectory forecasting, showing a higher accuracy than traditional time series forecasting methods like recurrent neural networks and differential equations. We show that the proposed techniques could capture uncertainty or variability, and we proposed different methods to take advantage of the uncertainty estimates. We focused on real-time performance and reliability of our methods. We assume that prediction errors and unexpected events will be likely to happen in the robot environment, and proposed methods that are resilient to failures in all the stages of the robot strategy. For example, on our table tennis setup we have outlier rejection techniques in the vision system, we added resilience to outliers again in the ball model using the trajectory variational auto-encoder, and we use the likelihood of our probabilistic method to detect when the environment of the robot is too different from the available training data to avoid executing dangerous actions.

In Chapter 2, we introduced a learning method and adaptation operators for ProMPs in joint space and in task space. We compared the proposed learning algorithm with a previous learning method based on least squares, showing that the least squares method is a special case of the proposed inference algorithm. The proposed learning algorithm uses prior distributions to find numerically stable ProMP parameters even with very few demonstrations. Taking advantage of the numerical stability provided by the new learning algorithm, we proposed adaptation operators that follow directly from probability theory without requiring to add artificial noise. We also proposed an algorithm to adapt a ProMP learned on joint space to achieve a particular task space objective based on approximate probabilistic inference. We showed that all the proposed adaptation operators have very low computation latencies, as required for many real time tasks such as robot table tennis. We tested the proposed learning and adaptation operators in a robot table tennis task and in a coffee preparation task. We used the coffee task to validate better the generalization capacity of the proposed operators and to show that although table tennis is our motivating application, the techniques proposed on this thesis can be easily applied to other tasks.

In Chapter 3, we proposed an object tracking system using RGB cameras focusing on reliability and real time performance. We showed that the proposed vision system has a higher accuracy and resilience to outliers than previous work, while providing real time performance. We tested the robot table tennis strategy proposed in Chapter 2 removing all the outlier detection heuristics and obtained a small increase in performance.

In Chapter 4, we introduced a novel method for trajectory forecasting based on conditional variational auto-encoders. We compare the proposed method with other machine learning time series forecasting model such as recurrent neural networks, showing a better prediction accuracy specially as we predict farther into the future. We trained the proposed model on a data set of table tennis ball trajectories, evaluating the prediction performance and latency. We show that the proposed method can has a long term prediction error about 3 times smaller than competing

methods and an average prediction latency of about 9 milliseconds. We evaluated again the table tennis strategy proposed in Chapter 2 without re-training the ball model every time the ball gun position was changed, and we still obtained a small improvement on the hitting rate of about 2.2%. We also tried to play table tennis with a human opponents instead of using a ball gun, but we noticed that using a single ProMP to deal with all the variability required to play with a human opponent is not enough. To use multiple ProMPs, we would need a policy that chooses which movement primitive to use depending on the ball trajectory. However, we leave the implementation of such a policy as proposed future work.

5.1 Discussion and Future Work

During this thesis, we used robot table tennis as an inspiring application to develop probabilistic techniques for real time robot applications. In this section, we mention some open questions and promising directions of future research.

5.1.1 Robot Table Tennis Policy using Trajectory Variational Auto-Encoders

The table tennis strategy implemented in this thesis uses one single ProMP to represent possible robot trajectories. As a result, the robot can play for example forehand or backhand, but not both. To be able to play with a human opponent, even if the human opponent tries to play always to the same side, we noticed we need a more powerful policy representation than using a single ProMP. The main limitations that we would need to overcome to use multiple ProMPs is to decide how many ProMPs are necessary and which ProMP to use for a particular ball trajectory.

One possible solution is to use the trajectory variational auto-encoder latent representation of the ball trajectory z to decide what action does the robot need to take. Note that the optimization procedure presented in Chapter 4 will try to find a representation variable z that is useful to predict the future ball trajectory. The intuition is that the tasks of predicting the future ball trajectory and deciding the action that the robot needs to take are likely to be strongly correlated, and some transfer between the two tasks is possible requiring less data to learn the policy from demonstrations.

5.1.2 Apply Reinforcement Learning to Improve the Policy Learned from Demonstrations

Once a policy is learned from demonstrations, reinforcement learning could be used to further improve performance. While running the experiments on the table tennis platform, we discovered that it is not easy to make several successful demonstrations in a row moving the robot in gravity compensation mode. Assuming that a policy learned only from demonstrations can only be as good as the teacher, the robot would not be able to play a long game based solely on learning from demonstrations.

Applying reinforcement learning has the potential to improve the policy learned from demonstrations beyond the performance of the teacher. The key challenges to solve are how to improve the policy in a sample efficient way leveraging on the teacher demonstrations, the robot own experience and the provided reward function.

5.1.3 Learning Control with Trajectory Variational Auto-Encoders

In Chapter 4, we showed that Trajectory Variational Auto-Encoders (TVAE) could be used to learn to predict a future trajectory given the past observations. This model can be directly applied to predict the response of a robot given the past observations and the actions that are currently planned, acting as a forward dynamics model of the robot behaviour. We have already showed that TVAE models have a better long-term prediction accuracy than other popular machine learning methods such as recurrent neural networks. The question is if we can use these relatively accurate predictive models to control complex robot systems such as a pneumatic muscle actuated robot, where accurate physic-based models are not yet available.

To use a TVAE for control, we need to solve the inverse problem: given the previous joint measurements and the desired future joint measurements, predict the required actions. Note that there is nothing preventing a TVAE to be used on this way, except that collecting a representative data set is not easy, and smart exploration strategies are required. A possible solution would be to train a TVAE as a reinforcement learning problem. The user provides a desired trajectory to execute and the TVAE policy is used to predict the action trajectory that the robot needs to execute. At the beginning the predicted action trajectories should be mostly random, and the tracking performance should be poor. Feedback is provided by computing some distance between the desired trajectory and the obtained trajectory, and the reinforcement learning algorithm goal should be to minimize the distance between the planned and the obtained trajectories. Note that the user can give different desired trajectories every time to the TVAE, and therefore, the TVAE policy should be able to generalize well.

5.1.4 Measuring and Predicting Spin

The spin of the ball is very important for table tennis due to the low weight of the ball. However, the vision system proposed in Chapter 3 only tracks the position of the ball and not its spin, because detecting spin from low resolution images is not easy. In Chapter 4, we tested a TVAE model to forecast the future trajectory of the ball given past observations. The accuracy of TVAE was superior than the physic-based model on real data, and we assumed that is partially because the TVAE could infer the ball spin from the trajectory itself, whereas in the physic-based model we had to assume that there was not spin at all.

The vision system proposed in Chapter 3 could be extended to detect also the spin from the label printed on the ball. The system could be trained by attaching a ball to a motor where the spin of the ball can be carefully controlled. Another approach would be to build a robotic ball gun where the initial spin of the ball could be controlled.

5.2 Publications

Excerpts of the research presented in this thesis have led to the following publications.

5.2.1 Articles in Scientific Journals

1. S. Gomez-Gonzalez, G. Neumann, B. Schölkopf, and J. Peters, “Adaptation and robust learning of probabilistic movement primitives,” Accepted on IEEE Transactions on Robotics, March 2019. arXiv preprint 1808.10448, Aug. 2018.
2. S. Gomez-Gonzalez, Y. Nemmour, B. Schölkopf, and J. Peters, “Reliable real time ball tracking for robot table tennis,” Accepted on Robotics MDPI, October 2019.
3. S. Gomez-Gonzalez, S. Prokudin, B. Schölkopf, and J. Peters, “Real Time Trajectory Prediction Using Deep Conditional Generative Models,” In evaluation on ICRA RAL 2020, arXiv preprint 1909.03895, September 2019.

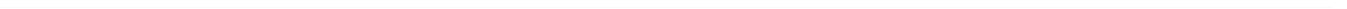
5.2.2 Articles in Conference Proceedings

1. S. Gomez-Gonzalez, G. Neumann, B. Schölkopf, and J. Peters, “Using probabilistic movement primitives for striking movements,” IEEE-RAS 16th International Conference on Humanoid Robots, 2016.
2. D. Agudelo-España, S. Gomez-Gonzalez, S. Bauer, B. Schölkopf, and J. Peters, “Bayesian Online Detection and Prediction of Change Points,” In evaluation on AI STATS 2020, arXiv preprint 1902.04524, September 2019.

5.2.3 Software Implementations

The research of this thesis also led to the following open source software implementation repositories:

1. “Open source implementation of the ball tracking system.” https://gitlab.tuebingen.mpg.de/sgomez/ball_tracking.
2. “Trajectory Variational Auto-Encoders implementation on Tensor-Flow,” https://github.com/sebasutp/trajectory_forecasting.
3. “Probabilistic Movement Primitives in Python,” <https://github.com/sebasutp/promp>.
4. “Probabilistic Movement Primitives in C++,” <https://github.com/sebasutp/promp-cpp>.





Bibliography

- [1] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, “Probabilistic movement primitives,” in *Advances in Neural Information Processing Systems*, pp. 2616–2624, 2013.
- [2] 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 (Cybernetics)*, vol. 37, no. 2, pp. 286–298, 2007.
- [3] M. Deisenroth and C. E. Rasmussen, “Pilco: A model-based and data-efficient approach to policy search,” in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.
- [4] N. Vlassis, M. Toussaint, G. Kontes, and S. Piperidis, “Learning model-free robot control by a monte carlo em algorithm,” *Autonomous Robots*, vol. 27, no. 2, pp. 123–130, 2009.
- [5] M. Alvarez, D. Luengo, and N. Lawrence, “Latent force models,” in *Artificial Intelligence and Statistics*, pp. 9–16, 2009.
- [6] S. Chiappa, J. Kober, and J. R. Peters, “Using bayesian dynamical systems for motion template libraries,” in *Advances in Neural Information Processing Systems*, pp. 297–304, 2009.
- [7] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” *arXiv preprint arXiv:1601.06759*, 2016.
- [8] A. Borovykh, S. Bohte, and C. W. Oosterlee, “Conditional time series forecasting with convolutional neural networks,” *arXiv preprint arXiv:1703.04691*, 2017.
- [9] S. Gomez-Gonzalez, G. Neumann, B. Schölkopf, and J. Peters, “Adaptation and robust learning of probabilistic movement primitives,” *ArXiv e-prints*, Aug. 2018.
- [10] K. Mülling, J. Kober, O. Kroemer, and J. Peters, “Learning to select and generalize striking movements in robot table tennis,” *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013.
- [11] C. Lampert and J. Peters, “Real-time detection of colored objects in multiple camera streams with off-the-shelf hardware components,” *Journal of Real-Time Image Processing*, vol. 7, no. 1, pp. 31–41, 2012.
- [12] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato, “Learning from demonstration and adaptation of biped locomotion,” *Robotics and Autonomous Systems*, vol. 47, no. 2, pp. 79–91, 2004.
- [13] A. Ude, A. Gams, T. Asfour, and J. Morimoto, “Task-specific generalization of discrete and periodic dynamic movement primitives,” *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, 2010.

-
- [14] J. Kober, “Learning motor skills: from algorithms to robot experiments,” *it-Information Technology*, vol. 56, no. 3, pp. 141–146, 2014.
- [15] P. Kormushev, S. Calinon, and D. G. Caldwell, “Robot motor skill coordination with em-based reinforcement learning,” in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 3232–3237, IEEE, 2010.
- [16] H. Müller and D. Sternad, “Motor learning: changes in the structure of variability in a redundant task,” in *Progress in motor control*, pp. 439–456, Springer, 2009.
- [17] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, “Robot programming by demonstration,” in *Springer Handbook of Robotics*, pp. 1371–1394, Springer, 2008.
- [18] S. Calinon, D. Bruno, and D. G. Caldwell, “A task-parameterized probabilistic model with minimal intervention control,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3339–3344, IEEE, 2014.
- [19] J. R. Medina, D. Lee, and S. Hirche, “Risk-sensitive optimal feedback control for haptic assistance,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1025–1031, IEEE, 2012.
- [20] K. Mülling, J. Kober, and J. Peters, “A biomimetic approach to robot table tennis,” *Adaptive Behavior*, vol. 19, no. 5, pp. 359–376, 2011.
- [21] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT Press, 2012.
- [22] A. Paraschos, E. Rueckert, J. Peters, and G. Neumann, “Model-free probabilistic movement primitives for physical interaction,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2860–2866, IEEE, 2015.
- [23] S. Gomez-Gonzalez, G. Neumann, B. Schölkopf, and J. Peters, “Using probabilistic movement primitives for striking movements,” in *IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pp. 502–508, IEEE, 2016.
- [24] D. A. Belsley, E. Kuh, and R. E. Welsch, *Regression diagnostics: Identifying influential data and sources of collinearity*. John Wiley & Sons, 2005.
- [25] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer Science & Business Media, 2010.
- [26] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.
- [27] C. H. Lampert and J. Peters, “Real-time detection of colored objects in multiple camera streams with off-the-shelf hardware components,” *Journal of Real-Time Image Processing*, vol. 7, no. 1, pp. 31–41, 2012.
- [28] M. Matsushima, T. Hashimoto, M. Takeuchi, and F. Miyazaki, “A learning approach to robotic table tennis,” *IEEE Transactions on robotics*, vol. 21, no. 4, pp. 767–771, 2005.
- [29] P. Bromiley, “Products and convolutions of gaussian probability density functions,” *Tina-Vision Memo*, vol. 3, 2003.

-
- [30] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *The international journal of robotics research*, vol. 4, no. 3, pp. 109–117, 1985.
- [31] D. Tolani, A. Goswami, and N. I. Badler, "Real-time inverse kinematics techniques for anthropomorphic limbs," *Graphical models*, vol. 62, no. 5, pp. 353–388, 2000.
- [32] O. Koç, G. Maeda, and J. Peters, "Online optimal trajectory generation for robot table tennis," *Robotics and Autonomous Systems*, vol. 105, pp. 121 – 137, 2018.
- [33] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.
- [34] A. H. M. Campbell and F. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.
- [35] C. M. D. Silver, A. Huang *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [36] Y. K. I. N. H. Kitano, M. Asada and E. Osawa, "Robocup: The robot world cup initiative," in *International conference on Autonomous agents*, pp. 340–347, ACM, 1997.
- [37] Y. Huang, B. Scholkopf, and J. Peters, "Learning optimal striking points for a ping-pong playing robot," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 4587–4592, IEEE, 2015.
- [38] "Open source implementation of the ball tracking system." https://gitlab.tuebingen.mpg.de/sgomez/ball_tracking.
- [39] H. K. Y. Seo, S. Choi and K. Hong, "Where are the ball and players? soccer game analysis with color-based tracking and image mosaick," in *International Conference on Image Analysis and Processing*, pp. 196–203, Springer, 1997.
- [40] H. L. X. Tong and Q. Liu, "An effective and fast soccer ball detection and tracking method," in *International Conference on Pattern Recognition*, vol. 4, pp. 795–798, IEEE, 2004.
- [41] Y. C. W. T. H. Chen, M. Tien and S. Lee, "Physics-based ball tracking and 3d trajectory reconstruction with applications to shooting location estimation in basketball video," *Journal of Visual Communication and Image Representation*, vol. 20, no. 3, pp. 204–216, 2009.
- [42] A. O. G. Pingali and Y. Jean, "Ball tracking and virtual replays for innovative tennis broadcasts," in *International Conference on Pattern Recognition*, vol. 4, pp. 152–156, IEEE, 2000.
- [43] B. W. Y. Z. H. Liu, Z. Li and Q. Zhang, "Table tennis robot with stereo vision and humanoid manipulator ii: Visual measurement of motion-blurred ball," in *International Conference on Robotics and Biomimetics*, pp. 2430–2435, IEEE, 2013.
- [44] L. L. K. K. H. Li, H. Wu and O. Ravn, "Ping-pong robotics with high-speed vision system," in *International Conference on Control Automation Robotics & Vision*, pp. 106–111, IEEE, 2012.

-
- [45] W. W. M. Z. Z. Y. X. Chen, Q. Huang *et al.*, “A robust vision module for humanoid robotic ping-pong game,” *International Journal of Advanced Robotic Systems*, vol. 12, no. 4, p. 35, 2015.
- [46] C. S. M. Z. J. Huang, V. Rathod *et al.*, “Speed/accuracy trade-offs for modern convolutional object detectors,” in *IEEE CVPR*, 2017.
- [47] D. E. C. S. S. R. C. F. W. Liu, D. Anguelov and A. Berg, “Ssd: Single shot multibox detector,” in *European conference on computer vision*, pp. 21–37, Springer, 2016.
- [48] B. C. D. K. A. Howard, M. Zhu *et al.*, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.
- [49] M. M. T. Pohlen, A. Hermans and B. Leibe, “Full-resolution residual networks for semantic segmentation in street scenes,” *arXiv preprint*, 2017.
- [50] A. Heyden and M. Pollefeys, “Multiple view geometry,” *Emerging topics in computer vision*, pp. 45–107, 2005.
- [51] V. Hodge and J. Austin, “A survey of outlier detection methodologies,” *Artificial intelligence review*, vol. 22, no. 2, pp. 85–126, 2004.
- [52] O. Chum, “Two-view geometry estimation by random sample and consensus,” *PhD Dissertation, CTU*, 2005.
- [53] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.
- [54] B. Tordoff and D. W. Murray, “Guided sampling and consensus for motion estimation,” in *European conference on computer vision*, pp. 82–96, Springer, 2002.
- [55] D. Nasuto and J. B. R. Craddock, “Napsac: High noise, high dimensional robust estimation—it’s in the bag,” in *Proc. Brit. Mach. Vision Conf.*, pp. 458–467, 2002.
- [56] R. Raguram, O. Chum, M. Pollefeys, J. Matas, and J.-M. Frahm, “Usac: a universal framework for random sample consensus,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 2022–2038, 2012.
- [57] Y. Zhao, R. Xiong, and Y. Zhang, “Model based motion state estimation and trajectory prediction of spinning ball for ping-pong robots using expectation-maximization algorithm,” *Journal of Intelligent & Robotic Systems*, vol. 87, no. 3-4, pp. 407–423, 2017.
- [58] D. Büchler, H. Ott, and J. Peters, “A lightweight robotic arm with pneumatic muscles for robot learning,” in *International Conference on Robotics and Automation (ICRA)*, pp. 4086–4092, IEEE, 2016.
- [59] K. Sohn, H. Lee, and X. Yan, “Learning structured output representation using deep conditional generative models,” in *Advances in Neural Information Processing Systems*, pp. 3483–3491, 2015.
- [60] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.

-
- [61] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [62] G. Repository, “Trajectory forecasting implementation repository.” https://github.com/sebasutp/trajectory_forecasting.
- [63] A. Van Den Oord, S. Dieleman, and Others, “Wavenet: A generative model for raw audio,” *CoRR abs/1609.03499*, 2016.
- [64] N. Nikhil and B. Tran Morris, “Convolutional neural network for trajectory prediction,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 0–0, 2018.
- [65] R. Yu, S. Zheng, A. Anandkumar, and Y. Yue, “Long-term forecasting using tensor-train rnns,” *arXiv preprint arXiv:1711.00073*, 2017.
- [66] C. Doersch, “Tutorial on variational autoencoders,” *arXiv preprint arXiv:1606.05908*, 2016.
- [67] S. Gomez-Gonzalez, Y. Nemmour, B. Schölkopf, and J. Peters, “Reliable real time ball tracking for robot table tennis,” *arXiv preprint arXiv:1908.07332*, 2019.



A Appendix

A.1 Derivation of the ELBO for TVAEs

In this section, we show the derivation of the lower-bound on the conditional log-likelihood for Trajectory Variational Auto-Encoders. The probability distribution over the latent variable \mathbf{z} given an entire trajectory $\mathbf{y}_{1:T}$ is given by

$$p(\mathbf{z} | \mathbf{y}_{1:T}) = p(\mathbf{z} | \mathbf{y}_{1:t-1}, \mathbf{y}_{t:T}) = \frac{p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1}, \mathbf{z})p(\mathbf{z} | \mathbf{y}_{1:t-1})}{p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1})},$$

where $p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1})$ is the conditional likelihood, whose computation requires evaluating an intractable integral. If we use a variational distribution $q(\mathbf{z} | \mathbf{y}_{1:T})$ to approximate the intractable distribution $p(\mathbf{z} | \mathbf{y}_{1:T})$, we can write the log conditional likelihood as

$$\log p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1}) = -\text{KL}(q(\mathbf{z} | \mathbf{y}_{1:T}) || p(\mathbf{z} | \mathbf{y}_{1:T})) + \mathbb{E}_{q(\mathbf{z} | \mathbf{y}_{1:T})}[\log p(\mathbf{y}_{t:T}, \mathbf{z} | \mathbf{y}_{1:t}) - \log q(\mathbf{z} | \mathbf{y}_{1:T})].$$

Provided that the KL divergence is a non-negative number, and using other standard rules of probability theory, we have

$$\begin{aligned} \log p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1}) &\geq \mathbb{E}_{q(\mathbf{z} | \mathbf{y}_{1:T})}[\log p(\mathbf{y}_{t:T}, \mathbf{z} | \mathbf{y}_{1:t}) - \log q(\mathbf{z} | \mathbf{y}_{1:T})] \\ &= E_{q(\mathbf{z} | \mathbf{y}_{1:T})}[\log p(\mathbf{z} | \mathbf{y}_{1:t}) + \log p(\mathbf{y}_{t:T} | \mathbf{z}, \mathbf{y}_{1:t-1}) - \log q(\mathbf{z} | \mathbf{y}_{1:T})] \\ &= E_{q(\mathbf{z} | \mathbf{y}_{1:T})}[-\log q(\mathbf{z} | \mathbf{y}_{1:T}) + \log p(\mathbf{z} | \mathbf{y}_{1:t})] + E_{q(\mathbf{z} | \mathbf{y}_{1:T})}[\log p(\mathbf{y}_{t:T} | \mathbf{z}, \mathbf{y}_{1:t-1})] \\ &= \text{KL}(q(\mathbf{z} | \mathbf{y}_{1:T}) || p(\mathbf{z} | \mathbf{y}_{1:t-1})) + \mathbb{E}_{q(\mathbf{z} | \mathbf{y}_{1:T})}[\log p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t}, \mathbf{z})]. \end{aligned}$$

Note that both $q(\mathbf{z} | \mathbf{y}_{1:T})$ and $p(\mathbf{z} | \mathbf{y}_{1:t-1})$ are distributions that map from observation space to the latent space, acting both as encoders, one for full trajectories and the other for partial trajectories. Using the masked variables \mathbf{x}^t and $\hat{\mathbf{x}}^t$ we can use a single encoder network \mathbf{g}_ϕ to encode both partial and full trajectories. We can write the lower bound as

$$\log p_\theta(\mathbf{y}_{t:T} | \mathbf{y}_{1:t-1}) \geq -\text{KL}(q_\phi(\mathbf{z} | \mathbf{y}_{1:T}) || q_\phi(\mathbf{z} | \mathbf{y}_{1:t-1})) + \mathbb{E}_{q_\phi(\mathbf{z} | \mathbf{y}_{1:T})}[\log p(\mathbf{y}_{t:T} | \mathbf{y}_{1:t}, \mathbf{z})],$$

arriving to the resulting lower bound presented on Chapter 4.



Sebastian Gomez

sebastian@robot-learning.de

+49 (172) 981 4937

Tübingen, Germany

Education

2015.9– **PhD on Machine Learning**, *Max Planck Institute for Intelligent Systems*, Tübingen – Germany.

Research in machine learning. Topics include probabilistic generative models, time series modeling and forecasting, robotics and computer vision.

Projects:

- Learning robot movement policies from human demonstrations.
- Vision system and robot policies for robot table tennis.

Example applications and videos:

- Learning robot table tennis striking movements (<https://youtu.be/KLK4IwpgEB4>)
- Learning to prepare coffee with the robot (<https://youtu.be/oHnA2RYIWxg>)

Tools and programming languages:

- Python, Numpy and Tensorflow to train and evaluate models
- C++ for the components that require real time execution
- ZeroMQ message passing library and JSON to communicate the different components

2011.8– **MS in Computer Science**, *Universidad Tecnológica de Pereira*, Colombia.

2014.8 Master studies with emphasis on machine learning. Research project on discriminative training procedures for convolved multiple output Gaussian Processes.

2006.2– **BS in Computer Science**, *Universidad Tecnológica de Pereira*, Colombia.

2011.7 Undergraduate studies on engineering. Topics include programming, data structures, math, software engineering, TCP/UDP/IP networking and digital electronics.

Publications

- 1 Adaptation and Robust Learning of Probabilistic Movement Primitives, Accepted on IEEE Transactions on Robotics, Arxiv Preprint 1808.10648, 2018
- 2 Real Time Trajectory Prediction Using Deep Conditional Generative Models, In evaluation on IEEE ICRA-RAL, Arxiv Preprint 1909.03895, 2019
- 3 Reliable Real Time Ball Tracking for Robot Table Tennis, In evaluation on MDPI Robotics & Automation, Arxiv Preprint 1908.07332, 2019
- 4 Using probabilistic movement primitives for striking movements, IEEE RAS International Conference on Humanoid Robots, 2016
- 5 Discriminative training for convolved multiple-output Gaussian processes, Iberoamerican Congress on Pattern Recognition, 2015

Awards and Achievements

2012 **Classification to the ACM-ICPC Programming Competition World Finals**, *Saint Petersburg - Russia*.

Representing Colombia in a University programming competition world finals sponsored by ACM, after obtaining the second place in the South American-North contest. Our team ranked third in the ICPC challenge of artificial intelligence during the world finals event.

-
- 2010 **Ranked first in Colombia in the official state exam for university students.**
Ranked first in Colombia in the official state exam ECAES on Computer Engineering in 2010. This exam must be presented by all the undergraduate university students in Colombia before graduation.
 - 2009 **ELAP Scholarship.**
Obtained a full scholarship during my bachelor studies from the Canadian government to study one semester in Quebec-Canada.
 - 2012 **ROA Scholarship.**
Obtained a full Scholarship for my Master Studies due to outstanding undergraduate GPA of 4.7 out of 5.
 - 2008 **Ranked second in Microsoft Imagine Cup competition.**
Ranked second in Colombia in the Microsoft ImagineCup competition on the programming category.

Work Experience

- 2019.4–2019.8 **Software Engineer Internship, Google, Switzerland.**
Experiment, develop and specify parts of a new image compression format Google PIK. The Google PIK format is a modernized variant of JPEG designed for high quality and fast decoding developed by the Google Compression Team in Zurich.
- 2018.8–2018.12 **Machine Learning Consultant, Converge IO, United Kingdom.**
Develop and evaluate predictive models for time series sensor data in construction sites, using miscellaneous machine learning techniques ranging from linear regression to neural networks. Implementation in Python using Numpy and Tensorflow.
- 2014 **Developer - Linux Kernel and cryptography, NetHa, Colombia.**
Developed a Linux Kernel module to encrypt/decrypt network packages automatically while being routed. Implementation in C language.
- 2011.8–2015.6 **Computer Science Lecturer, Universidad Tecnológica de Pereira, Colombia.**
Teaching data structures, programming and math to computer science students.
- 2011–2013 **Developer - Web, Universidad Tecnológica de Pereira, Colombia.**
Development of a web site in the Ruby on Rails framework to teach and evaluate programming skills.

Academic and Technical skills

- Machine Learning
- Computer Vision
- Math
- Python
- Probabilistic Inference
- Robotics
- C++
- CUDA

References

- * Jan Peters, mail@jan-peters.net. Professor at Technische Universität Darmstadt.
 - * Mauricio Alvarez, mauricio.alvarez@sheffield.ac.uk. Professor at University of Sheffield.
 - * Santiago Gutierrez-Alzate, santigutierrez1@gmail.com. Software Engineer at Google.
-