Two-handed coordination in robots

By combining two one-handed trajectories based on probabilistic models of taskspace effects

by

Benjamin Blumer

B.Sc. (first-class honours), The University of Calgary, 2011

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Mechanical Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

December 2016

© Benjamin Blumer 2016
Abstract

Human environments and tools are commonly designed to be used by two-handed agents. In order for a robot to make use of human tools or to navigate in a human environment it must be able to use two arms. Planning motion for two arms is a difficult task as it requires taking into account a large number of joints and links and involves both temporal and spatial coordination. The work in this thesis addresses these problems by providing a framework to combine two single-arm trajectories to perform a two-armed task. Inspired by results indicating that humans perform better on motor tasks when focusing on the outcome of their movements rather than their joint motions, I propose a solution that considers each trajectory’s effect on the taskspace.

I develop a novel framework for modifying and combining one-armed trajectories to complete two-armed tasks. The framework is designed to be as general as possible and is agnostic to how the one-armed trajectories were generated and the robot(s) being used. Physical roll-outs of the individual arm trajectories are used to create probabilistic models of their performance in taskspace using Gaussian Mixture Models. This approach allows for error compensation. Trajectories are combined in taskspace in order to achieve the highest probability of success and task performance quality. The framework was tested using two Barrett WAM robots performing the difficult, two-armed task of serving a ping-pong ball. For
Abstract

this demonstration, the trajectories were created using quintic interpolations of joint coordinates. The trajectory combinations are tested for collisions in the robot simulation tool, Gazebo. I demonstrated that the system can successfully choose and execute the highest-probability trajectory combination that is collision-free to achieve a given taskspace goal.

The framework achieved timing of the two single-arm trajectories optimal to within 0.0389 s – approximately equal to the time between frames of the 30Hz camera. The implemented algorithm successfully ranked the likelihood of success for four out of five serving motions. Finally, the framework’s ability to perform a higher-level tasks was demonstrated by performing a legal ping-pong serve. These results were achieved despite significant noise in the data.
Preface

The author developed the research problem in consultation with Drs. Machiel Van der Loos and Elizabeth Croft. The author conducted the literature review, developed the algorithm, implemented all software, conducted all experiments, and performed the analysis.

The contents of Chapter 3 and the first experiment in Chapter 5 have been submitted for publication as: Benjamin Blumer, Machiel Van der Loos, and Elizabeth Croft. Serving up two-handed coordination. International Conference on Robotics and Automation 2017. The author wrote the submission. Editing and supervision were provided by Drs. Machiel Van der Loos and Elizabeth Croft.

All software and hardware developed for this work (described in Chapter 4) is available at the author’s BitBucket software repository [1]. Videos of the robot serving are available on the author’s YouTube page [2].
Contents

Abstract ................................................................. ii

Preface ................................................................. iv

Contents ............................................................... v

List of Tables ......................................................... ix

List of Figures ......................................................... x

Glossary ............................................................... xii

Acknowledgements .................................................. xiv

Dedication ............................................................. xv

1 Introduction ......................................................... 1
  1.1 Introduction to robot motion planning ......................... 2
  1.2 Challenges in two-arm motion planning ....................... 3
    1.2.1 Computational difficulty ................................. 4
    1.2.2 Uncertainties in robot execution and environment ...... 5
  1.3 Why a ping-pong serve? ..................................... 6

v
## Contents

1.4 Contributions ........................................... 7
  1.4.1 A conceptual framework for planning two-armed motions 7
  1.4.2 An implementation of this framework to demonstrate its capabilities and feasibility ............................ 8
  1.4.3 Validation of the framework by using it to complete a ping-pong serving task ...................................... 9

1.5 Thesis layout ................................................ 9

2 Background .................................................. 11

3 Framework .................................................. 20
  3.1 Obtaining one-armed trajectories .......................... 23
  3.2 Probabilistic taskspace trajectory representations ............. 24
  3.3 Combining two single-armed trajectories ....................... 27
    3.3.1 Goal ................................................ 28
    3.3.2 Tool - Mode finding ................................ 29
    3.3.3 Tool - Marginal distribution .......................... 29
    3.3.4 Tool - Conditional distribution ......................... 32
    3.3.5 Tool - Joint distribution .............................. 33
    3.3.6 Tool - External modifications ......................... 34
    3.3.7 Tool - Time shifting ................................ 34
    3.3.8 Ranking ............................................ 38
  3.4 Collision Avoidance ..................................... 38
  3.5 Summary ................................................ 40

4 Implementation ............................................. 41
5.3.4 Discussion .............................. 72
5.3.5 Conclusion ................................. 73
5.4 Experiments 1 & 2 and service demonstration: discussion and conclusion .... 73

6 Conclusion ........................................ 75
  6.1 Summary of contributions .................. 75
  6.2 Summary of results .......................... 77
  6.3 Future work ................................. 78
  6.4 Ping-pong serving challenge ............... 79

Bibliography ...................................... 80
List of Tables

5.1 The number of successful hits for each different throwing trajectory (Experiment 1). .............................................. 57
5.2 Standard deviation of the ball and paddle positions at the temporal mode (Experiment 1). .............................................. 57
5.3 The number of hits for different offsets from the algorithm-recommended time using trajectories $T_{s1}^5$ and $T_{s2}^5$ (Experiment 2). ............... 67
5.4 The number of hits for different offsets from the algorithm-recommended time using trajectories $T_{s1}^4$ and $T_{s2}^4$ (Experiment 2). ............... 67
5.5 The number of hits for different offsets from the algorithm-recommended time using trajectories $T_{s1}^3$ and $T_{s2}^3$ (Experiment 2). ............... 68
5.6 The number of hits for different offsets from the algorithm-recommended time using trajectories $T_{s1}^2$ and $T_{s2}^2$ (Experiment 2). ............... 68
5.7 The number of hits for different offsets from the algorithm-recommended time using trajectories $T_{s1}^1$ and $T_{s2}^1$ (Experiment 2). ............... 68
5.8 The lower and upper bound on the satisfactory performance windows for different combinations of swing and throwing trajectories (Experiment 2). .............................................. 70
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>A sample two-handed trajectory generated by the master-slave approach in [3].</td>
<td>13</td>
</tr>
<tr>
<td>3.1</td>
<td>The two-handed coordination framework.</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>A 2-component, 1-dimensional Gaussian Mixture Model (GMM) [4].</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>Marginal distribution of a 3-component, 2-dimensional GMM.</td>
<td>31</td>
</tr>
<tr>
<td>3.4</td>
<td>Conditional distribution of a 3-component, 2-dimensional GMM.</td>
<td>32</td>
</tr>
<tr>
<td>3.5</td>
<td>A joint probability distribution.</td>
<td>34</td>
</tr>
<tr>
<td>3.6</td>
<td>The spatial coordinates of the left and right hands for an example regrasping task.</td>
<td>36</td>
</tr>
<tr>
<td>3.7</td>
<td>Marginal probability density functions for each hand for the regrasp task (top and middle) and joint probability density (bottom).</td>
<td>37</td>
</tr>
<tr>
<td>3.8</td>
<td>Conditional (temporal) probability densities of the left and right hands for an example re-grasping task.</td>
<td>39</td>
</tr>
<tr>
<td>4.1</td>
<td>Laser-cut acrylic ball-throwing hand.</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>3D-printed ping-pong paddle holder.</td>
<td>42</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>4.3</td>
<td>Workflow for generating one-armed trajectories.</td>
<td>44</td>
</tr>
<tr>
<td>4.4</td>
<td>Workflow for capturing taskspace outcomes.</td>
<td>46</td>
</tr>
<tr>
<td>4.5</td>
<td>Workflow for creating Gaussian Mixture Models from taskspace outcomes.</td>
<td>49</td>
</tr>
<tr>
<td>4.6</td>
<td>Pairing trajectories and ranking them by their effect on the combined taskspace.</td>
<td>50</td>
</tr>
<tr>
<td>4.7</td>
<td>Collision-checking procedure.</td>
<td>53</td>
</tr>
<tr>
<td>5.1</td>
<td>Experimentally-determined probability of success vs algorithm-predicted probability of success.</td>
<td>59</td>
</tr>
<tr>
<td>5.2</td>
<td>Cartesian locations of the ball and paddle (Experiment 1) for $j_6 = 0.17$ radians</td>
<td>60</td>
</tr>
<tr>
<td>5.3</td>
<td>Cartesian locations of the ball and paddle (Experiment 1) for $j_6 = 0.22$ radians</td>
<td>61</td>
</tr>
<tr>
<td>5.4</td>
<td>Cartesian locations of the ball and paddle (Experiment 1) for $j_6 = 0.27$ radians</td>
<td>62</td>
</tr>
<tr>
<td>5.5</td>
<td>Cartesian locations of the ball and paddle (Experiment 1) for $j_6 = 0.32$ radians</td>
<td>63</td>
</tr>
<tr>
<td>5.6</td>
<td>Cartesian locations of the ball and paddle (Experiment 1) for $j_6 = 0.37$ radians</td>
<td>64</td>
</tr>
<tr>
<td>5.7</td>
<td>Success vs time shift from the algorithm-recommended time delay between throwing the ball and swinging the paddle.</td>
<td>69</td>
</tr>
<tr>
<td>5.8</td>
<td>Setup for the ping-pong serve demonstration.</td>
<td>72</td>
</tr>
</tbody>
</table>
Glossary

Bumblebee A stereovision camera from Point Grey Research. xii, 44, 53

Configuration A configuration of a manipulator is a complete specification of the location of every point on the manipulator [5]. xii

Configuration Space The set of all configurations is known as the configuration space [5]. xii

CSV A table of data in which columns are separated by commas. Useful format for saving and loading datasets on a computer. xii, 47

DMP The Dynamic Movement Primitive framework is a means of storing a robot trajectory as a non-linear second order differential equation. xii, 16, 23

DOF An object has $n$ degrees of freedom (DOF) if its configuration can be minimally specified by $n$ parameters [5]. xii, 14, 15

EM Expectation-maximization is an algorithm used for fitting parameters in a Gaussian Mixture Model. xii, 26, 47

GMM A Gaussian Mixture Model is a probability distribution composed of a weighted sum of Gaussians. x, xii, 26, 29–31, 47, 50, 53, 54, 59, 65
Glossary

**HMM** A statistical model. Informally, the model has the assumptions that the (hidden) state of the system probabilistically influences the output. The probability of transitioning to another state is dependent only on the current state.. xii, 18

**ITTF** The governing body for international table tennis associations.. xii, 6, 72

**ROS** An open-source robot operating system. ROS acts as a robot-agnostic communication layer, simplifying the task of having different pieces of software communicate with each other or with robot hardware.. xii, 42, 44, 46, 47, 49, 50, 52, 53

**WAM** A 7-degree of freedom robot arm.. xii, 8, 44, 50
Acknowledgements

I would like to thank my research supervisors, Drs. Elizabeth Croft and Machiel Van der Loos. They granted me the freedom to explore different fields and problems so that I could find an area that really interested me. They supported me in two formative experiences: doing research at Reykjavík University, Iceland for a semester and performing research and development at a local company. Without their patience and support, this thesis would not have come to fruition.

I am also grateful for the wonderful people I have for labmates at the CARIS lab. In particular, I would like to acknowledge AJung Moon and Navid Lambert-Shirzad for lengthy discussions that helped me narrow my research topic and Philip Wang for help modelling the paddle-holding hand depicted in Chapter 4.

My dog, Bojo Dogeo, has never failed to cheer me up with a goofy prance, licks, or cuddles. Her companionship while writing, scripting, and analysing data has been invaluable.

Laurel Anderson has been an incredible lab assistant, late-night-thesis-coding-session dog walker, sounding board, and friend.

Finally, I would like to thank two undergraduate interns: Katelyn Currie, for her collaboration on the design of the ball dispenser described in Chapter 1 and early prototypes of the ball-holding hand depicted in Chapter 4, and Mateus Andreazza for his contributions to the ball-tracking code.
To my parents, Joan, Steve, and Rosanne, and my grandparents, Rhoda and Jack, for their unending encouragement of my nerdy endeavours: personal, academic, and professional.
Chapter 1

Introduction

Robots have the potential to improve human lives in many ways. For example, robots could be used to improve productivity, perform dangerous or boring jobs, or provide assistive care to older persons or people with disabilities. However, many of these jobs require robots to function in human environments and to work with human tools. Human environments are largely designed for two-handed agents. For example, to open a door while carrying something requires the coordination of two arms. Moreover, many human devices require two arms. To open a childproof medicine container, button a shirt, or to operate a seat belt requires two handed-coordination. Human tools that require two hands include the broom and dustpan, hammer and nail, and even a fork and knife. Therefore, it is important to develop the capability for robots to use two arms.

Aspects of two-handed coordination include: relative and absolute position, relative and absolute timing, relative and absolute velocity, and collision avoidance. Clapping is a good example to illustrate this. Imagine applauding after a performance has finished. A person needs the absolute positions of both hands to be in front of the body. Clapping to the side or behind defies social conventions, and risks hitting other performance attendees. In order to clap, the relative position of the palms of the hands must be zero at some point in time – that is, they
must touch. If the absolute timing is premature, the performance would be inter-
rupted. If it is delayed, it could result in an awkward pause. The hands must have
a relative velocity towards each other so that they reach, and an absolute velocity
such that the motion is symmetric. Unintended collisions between the arms and the
environment must be avoided.

1.1 Introduction to robot motion planning

For a robot to accomplish a task, it must be be able to plan and execute motions.
Different tasks require fundamentally different motions and considerations. Take
the examples of a robot pouring a drink [6] and a robot folding clothes [7]. The
path a hand must take to pour a can of soda into a cup is entirely different than the
path(s) required to fold a shirt.

While each task will have its own requirements, the ability to generalize a mo-
tion is important. Ideally, the same algorithm that allows a robot to fold a facecloth
would also allow it to fold a bath towel. Similarly, a drink-pouring robot should be
able to pour a drink in a cup in different locations on different tables or counters.
For this reason, it is desirable to generate appropriate trajectories as needed rather
than pre-planning every motion.

Robot motions are planned in either jointspace, workspace, or taskspace. Jointspace
refers to a (multidimensional) space consisting of angles for each of the robot’s
joints. For example, a robot arm with a single wrist, elbow, and shoulder joint
would have a 3-dimensional jointspace. Any point in that space would describe
a potential robot configuration (e.g. shoulder joint rotated at $\pi$, elbow joint bent
at $\frac{\pi}{2}$, wrist joint bent to $\frac{\pi}{4}$). Workspace refers to the area a robot is moving in.
1.2 Challenges in two-arm motion planning

For a wheeled robot, this may refer to the floor it is travelling on. For a robotic arm, this may refer to the 3D-space within reach of the hand. Taskspace can be an abstract space representing configurations of variables relevant to the task at hand. For example, the taskspace for the drink-pouring task might consist of the relative distance between the pouring and receiving cup or the amount of fluid in either.

Often, motion planning involves more than one of these spaces. For example, a basic approach to planning a motion is to identify the desired Cartesian (workspace) coordinates of the beginning and end of the motion that will achieve a (taskspace) goal. Then, use an inverse-kinematics approach, e.g., taking the inverse of the Jacobian [5], to obtain the start- and end-coordinates in the robot’s jointspace. Then a trajectory can be created by linearly interpolating between the start- and end-joint coordinates. This set of time-stamped coordinates is then sent to a lower-level controller to determine the appropriate power to send to each motor.

1.2 Challenges in two-arm motion planning

Generally, planning motions for two arms is more difficult than planning motions for a single arm. Motions must be planned for twice the number of joints and this increases the computational complexity. These motions must be coordinated spatially and temporally; this increases the importance of accounting for uncertainties in robot performance and the environment.

My overall contribution through this thesis is a proposed framework for two-handed coordination in robots that addresses these challenges. Subsections 1.2.1 and 1.2.2 below detail these challenges. Section 1.3 provides context for addressing
1.2. Challenges in two-arm motion planning

this challenge, and 1.4 provides details of the specific contributions of this work.

1.2.1  Computational difficulty

Planning two-arm motions tends to be more complex than planning one-arm motions at every layer of motion planning: the taskspace, workspace, and jointspace. Examples of difficulties in each of these areas are described below.

Tasks that require two arms typically have more sophisticated taskspaces than those that only involve one arm. In addition to having to plan for a second arm, the task spaces of each arm may need to be coordinated, as described in the opening of this chapter. For example, in hitting nails in with a hammer, it is not sufficient to bring nails to their desired location, since the hammer strike must be spatially coordinated with this location.

The workspaces of two-arm robots can also be more complex. Robots with different geometries will also have different workspaces. If the same robots are used, the workspaces will still differ because the robots must be placed in different locations. Additionally, if the workspaces of the arms overlap, unintended collisions must be avoided.

Consider the jointspace for a robot with \( j \) joints, each of which may be at any of \( N \) discrete values during \( m \) time steps. The jointspace then contains \( N^{mj} \) options.\(^2\) Therefore, the jointspace grows exponentially with the number of joints that require simultaneous planning.

\(^2\)If one makes the simplifying assumption that a robot can move from any configuration to any other in one timestep.
1.2. Challenges in two-arm motion planning

1.2.2 Uncertainties in robot execution and environment

The execution of robot motions does not always follow the planned motion. For example, if planning does not account for the physical limitations of the robot’s motors and joints, the performed trajectory can differ from the planned trajectory temporally and/or spatially. Additionally, the control software may not be capable of executing the desired trajectory. Static friction can also impact the trajectory between executions.

The environment may be unreliable. If operating outdoors, weather may affect trajectory performance. Inside a factory, variations in noise or light could affect sensor readings and therefore servoing performance.

Other collaborating robots may perform differently depending on their job queue or available processing power. Collaborating humans may perform differently depending on their current mental state or how much priority they give the task at hand.

Planning motions to affect objects in the task space is an even more difficult task; these plans require an understanding of the dynamics of external objects. In some cases, the tool may have complex dynamics such that a minor variation in trajectory performance creates a large variation in task performance. In other cases, the tool may not be identical; for example, if a previously used wrench was replaced with a different wrench.

Perfect situational knowledge, modelling, and control would be required to entirely overcome these substantial obstacles. However, all approaches should consider these challenges and attempt to mitigate them.
1.3 Why a ping-pong serve?

To assess the performance of two-armed motion-planning schemes, I propose the task of serving a ping-pong ball. In this task, a robot must use one hand to toss a ball in the air, and use the other arm to strike it using a ping-pong paddle. For the serve to be legal by International Table Tennis Federation (ITTF) rules [8], the ball must bounce exactly once on the server’s side of the table, clear the net, and bounce at least once on the opponent’s side of the table.

This task is chosen for its difficulty. It is very sensitive to timing, positioning, and velocity of each arm and their coordination. If the paddle is swung too early or too late, the ball will be missed. If the paddle is swung to the wrong location, the ball will be missed. If the paddle and ball do not have appropriate relative and absolute velocity, the ball will not follow a legal trajectory. Additionally, the physics and strategy of ping-pong remain an open problem. This is demonstrated by the abundance of research investigating, e.g., numerical trajectory prediction of a spinning, bouncing ball [9], human biomechanics during ping-pong [10, 11], and, explicitly, ping-pong serving strategy [12]. Ping-pong serving provides a rich area of study in developing two-handed strategy, motion-planning, and actuation.

Several other works have touched on the robot serving problem. In [13], a ball-launching mechanism is described for the “tossing” portion of the task. Wu and Kong, at the Institute of Cyber-Systems and Control, simply drop the ball from one hand at a set height and swing the paddle using the other.\(^3\) However, in both cases workarounds have been used in place of coordinating a throw with a hit.

It is worth noting that ping-pong has a long history as a benchmark in robotics.\(^3\) A peer-reviewed source is not available at the time of writing. However, a video of the robot’s serve can be seen at [14].
1.4 Contributions

The task was proposed at least as early as 1984 by John Billingsley [15]. At the time of writing, a search for “robot ping pong” returns over 400 results on the academic article database, Google Scholar. A book has been published on the topic [16]. Despite this attention, interesting problems remain including ball-spin tracking [17], movement generation [18], and much more.

Extending this benchmark to a two-armed task is a natural progression. The starting point is to serve a ball legally. However, competitive ping-pong players will try to serve the ball with:

- Spin, so that its trajectory curves in the air or so that it changes directions when it bounces.
- Varying speeds.
- Varying placement.

These are natural extensions to the challenge. An additional challenge is decision making, e.g., choosing the best serve given the opponent’s past performance and positioning. In this thesis, basic, legal serves are demonstrated.

1.4 Contributions

1.4.1 A conceptual framework for planning two-armed motions

I develop a 4-component framework to build a collection of one-armed trajectories, model their outcomes in taskspace, and combine the trajectories to perform a two-armed task. The framework includes procedures for assessing the relative likelihood of success of different combinations, producing timing offsets for the
1.4. Contributions

two one-armed motions, and testing, in advance, for collisions between the arms. This framework is applicable to any task in which:

- Relevant single-arm trajectories can be generated.\textsuperscript{4}

- The relationship between the taskspace outcomes of each arm’s movement to the desired two-armed taskspace goal can be quantified.

1.4.2 An implementation of this framework to demonstrate its capabilities and feasibility

I present my open-source software suite that implements the framework on two WAM (Barrett Technology, LLC. Newton, MA, USA) arms. This software suite includes modules to:

- Capture the taskspace outcome (ping-pong ball and paddle location) using a stereo-vision camera.

- Model taskspace outcomes probabilistically.

- Combine two single-arm trajectories to complete a two-armed task. This includes providing a time-offset for the start times of the two trajectories.

- Predict the relative success rates of different combinations.

- Check combinations of single-arm trajectories for collisions.

- Visualize multi-dimensional taskspace outcomes.

- Simultaneously control two WAM robots.

\textsuperscript{4}The implementation in this thesis is open-loop. As such, the task must be such that these trajectories can be generated before the motion is executed.
I also provide open-source hardware designs for:

- A laser-cut robot end-effector that can be used to toss a ball.
- A 3D-printed robot end-effector that can be used to grip a ping-pong paddle
- A ball dispenser that allows a robot to autonomously retrieve balls.\(^5\)

1.4.3 Validation of the framework by using it to complete a ping-pong serving task

To show the framework’s application to a problem, I demonstrate its capabilities in a ping-pong serving task. I conduct experiments and analyses of the ability to predict which combinations of trajectories are more likely to succeed as well as its ability to pick the optimal time delay between the start times of the two single-armed trajectories.

1.5 Thesis layout

In Chapter 2, I discuss existing approaches to two-arm motion planning and highlight some of the key remaining challenges. Chapter 3 describes the concept of my proposed framework. This includes the generation of trajectories, creating taskspace probability distributions, higher-level decision making, and collision avoidance. The discussion in this chapter is designed to be task-agnostic. Chapter 4 discusses how the framework is applied to the ping-pong serving task. Task-specific hardware and software is also discussed in this chapter. Chapter 5 presents

\(^5\)Unfortunately, the ball dispenser was not compatible with the hand used for the final analysis. However, it is a novel design and a hand could be designed to be compatible with it. The design of the ball dispenser is available at the author’s BitBucket software repository [1].
1.5. Thesis layout

the results of the ping-pong serving task. This includes commentary on the efficiency of the scheme and task-performance. Chapter 6 discusses the conclusions that can be drawn from this thesis as well as ideas for extensions to the framework.
Chapter 2

Background

Two-handed coordination is crucial to completing many tasks. In designing control schemes for two-handed robots, one must overcome the challenges of computational difficulty, collision avoidance, and uncertainty in robot performance and environment (Subsection 1.2.2). Researchers have attacked this problem from different angles. This chapter is dedicated to reviewing approaches to two-armed motion planning. The strengths and scope of these approaches are discussed and compared with the proposed approach. The review in this chapter focuses on the aspects of motion planning that are directly relevant to the system developed within this thesis. See Section 1.3 for the origin of the robot ping-pong challenge and approaches to robot ping-pong serving. Necessary mathematical tools are discussed in Chapter 3. For a thorough review of more aspects of two-handed motion planning, including kinematics and control equations, see [19] and [20].

This section compares concepts used in the previous work to decisions made about the proposed work. While the full description of the framework is in the subsequent chapter, for this discussion, it suffices to know:

- Each arm has a collection of trajectories.
- Each trajectory is associated with a taskspace outcome.
- Decisions are made about which trajectories to combine based on probabilis-
tic models of the taskspace outcomes.

- Optimal timing of the two one-armed trajectories is calculated using these probabilistic models.

Closed-form solutions have been devised for certain two-armed tasks. These solutions have a specific set of instructions that guarantee the completion of a task. For example, in [21], an algorithm for completing a two-arm pick-and-place task is described for two SCARA robots. They describe an order for the movement of each joint of each robot that ensures the objects can be picked off a conveyor belt and placed in a bin without the two arms colliding. In particular, the planning is broken into multiple, lower-dimensional configuration spaces; one for each robot and one for each part on the table. They also make several assumptions, such as the robot being clear of the table and obstacles when at maximum height, that allows them to reduce the configuration space of each robot to two dimensions. Closed form solutions are typically efficient, as they don’t require iterative approaches. However, they are problem-specific and robot-specific.

Another common approach to coordinating two arms is a master/slave approach (e.g., [22], [23], and [24]). One arm is the leader, and the other arm follows to reduce loading of the object being manipulated. The authors of one of the earlier works, [3], used this approach for a box carrying application. Two arms were given the instruction to grasp a box. The master arm was given a trajectory to a final destination. The second arm’s path is updated in increments to reduce the loading on first arm’s end-effector. In particular, the position of the second arm, $P'$ is incremented on the $j + 1$ iteration by
Chapter 2. Background

Figure 2.1: A sample two-handed trajectory generated by the master-slave approach in [3].

\[
\Delta P'_{G,j+1} = \begin{pmatrix}
F_{xj} / (F_{xj} - F_{xj-1}) & 0 & 0 \\
0 & F_{yj} / (F_{yj} - F_{yj-1}) & 0 \\
0 & 0 & F_{zj} / (F_{zj} - F_{zj-1})
\end{pmatrix} \Delta P'_{G,j}
\]

where \(F_j = (f_{xj} \ f_{yj} \ f_{zj})\) is the force on the first arm’s end-effector on iteration \(j\). A trajectory generated by this scheme is depicted in Figure 2.1. This approach has been useful for many tasks; for example, similar methods have been used to allow a humanoid to maintain balance while using both arms to slide an object in a plane [25]. While these techniques are very useful for certain tasks, the force and/or position following algorithms must be designed on a per-task basis, and they tend to work better for tasks where the two arm motions are similar. This excludes many tasks where the arms have asymmetric roles.

Because this thesis aims to address coping with uncertainties in the robots...
and the environment, it is worth highlighting two particular papers using the master/slave approach. In [26] and [27], the authors present fuzzy control schemes to compensate for model uncertainties and external disturbances. These papers focus on force and position control of an end-effector for jointly manipulating an object.

The fuzzy-control approaches are responding to one of the key challenges of two-handed motions: multi-robot systems are inherently high-degree of freedom (DOF) systems and require compensation for uncertainty in the environment and the other robot’s performance. The cited works develop force and position controllers for the task of jointly manipulating a single object. In this framework, we cope with uncertainty by building probabilistic models of taskspace outcomes. The monitored taskspace variables are chosen on a per-task basis. This has the advantage of being applicable to a wider-variety of tasks, including those with asymmetric roles for each arm.

For more generic two-arm tasks, often machine learning is used to simultaneously plan the motions of both arms. To generate trajectories using machine learning, the joint space is explored until a desirable outcome is achieved. For example, [28] proposes using a co-evolutionary algorithm to plan two-armed motions. Each arm starts with its own set of trajectories that co-evolve with the other arm’s trajectories until a suitable, collision-free two-arm trajectory is found. In their implementation, each chromosome consists of alleles representing configurations at each discretized point in time: \( \{ q_{11}^{(\Delta t,G)}, \ldots, q_{ij}^{(\Delta t,G)} \}, \ldots, \{ q_{11}^{((n-2)\Delta t,G)}, \ldots, q_{ij}^{((n-2)\Delta t,G)} \} \) where \( n \) is the total number of time steps, and \( q_{ij} \) represents the \( j \)th DOF of the \( i \)th robot, and \( G \) is the current generation. These chromosomes are assessed for fitness using an optimization function based on the number of collisions between the two robots, total joint-distance travelled, total Cartesian-distance travelled, and acceleration.
Chapter 2. Background

This requires $O(\text{population size} \times \text{number of generations} \times n \times \text{number of DOF})$ operations. While this is only linearly dependent on the DOF term, the required population size and number of generations are implicitly dependent on DOF; the more complicated the configuration space, the more generations and the larger the population size are required to reach a satisfactory trajectory. In fact, the implementation of the algorithm uses the simplified case of two robots working in a plane – using only two DOF each – and the algorithm still required approximately 1000 iterations to converge. This is a general difficulty with computing two-handed trajectories. As mentioned in Subsection 1.2.1, the total number of configurations of a robot, or several robots, tends to grow exponentially with the number of joints. As a result, machine learning can be slow for planning two-arm trajectories.

In the co-evolution work, each robot’s one-armed trajectories are considered separately. However, they still must evolve to be mutually useful – requiring many iterations. In the proposed framework, the trajectories are treated separately until a decision must be made about which combination to use. The proposed approach has the advantage that is is easier to generate the one-armed trajectories and the disadvantage that it applies only to tasks in which the desired outcome of the two-armed task can be described in terms of the outcomes of the one-armed tasks. However, many useful tasks meet this requirement.

To expedite the search, a technique called teach by demonstration can be used. In teach by demonstration, humans demonstrate a possible solution to provide a starting point for the search. This can be done by physically manipulating the robots (kinesthetic teaching). For example, in [29], subjects guide a humanoid robot’s end-effectors in an effort to complete two tasks: flipping a box using chopsticks and hitting a ball with a pool cue. They use Policy Improvement with Path
Chapter 2. Background

Integrals (\(\text{PI}^2\)) [30] to adjust parameters of the robot’s movement. The robot’s trajectories are encoded as Dynamic Movement Primitives (DMP) [31] to reduce the number of parameters to describe the motion. DMPs encode movement using a dynamic system with a series of perturbations in a non-linear function. For the pool task, they choose to encode relevant taskspace variables such as roll, pitch, and yaw of the cue around the bridge. \(\text{PI}^2\) adds noise to the weights of different components of this non-linear function to explore the trajectory space. While encoding the movements as DMPs decreases the size of the search space, it is still substantially larger, and slower to search, than the space that would be required for machine learning of one-armed trajectories.

The encoding of movements in taskspace variables in [29] is useful for reducing the number of dimensions that need to be explored. It also focuses computational effort on what is, arguably, most important: what happens in the taskspace as a result of the robot’s movement. Pastor et al. also take advantage of this parameterization to enforce certain constraints on the generated trajectories – such as the pool cue remaining in the bridge. In [29], they encode intermediate steps in the taskspace (i.e. the pool-cue movement). The proposed framework stores trajectories with their final taskspace outcomes (i.e. where the ball goes, rather than how the throwing-hand moves). While enforcing constraints is not a priority in this thesis, reducing dimensionality and making decisions using taskspace outcomes is.

One challenge to kinesthetic teaching, is that physically manipulating two arms at once can be cumbersome, especially if the timing needs to be precise. Indeed, in [29], each subject achieved an average of three successes out of twenty attempts for the box-flipping task. A consortium of European universities has begun the X-act project [32] to expand the applications of two-handed robots. They propose
teaching two-handed motions via body gestures recognized by a Microsoft (Redmond, WA, USA) Kinect and through 3D mice. These techniques are promising; however, even with a good seed for the machine learning algorithm, exploration of this space will still be significantly more computationally expensive and challenging than exploring the jointspace of a single-arm trajectory with an equally good seed due to the size of the parameter space.

One approach to combining two one-arm trajectories to accomplish a two-arm task is known as prioritized planning, in which an ordering is assigned to multiple robots [33]. The robot with the highest priority generates a trajectory, then the second robot generates a trajectory that treats the first robot as a dynamic obstacle with a known trajectory. This process can be extended for additional robots. [34] explores prioritizing by total path distance, i.e., the robot with the longest distance from its start to end position is given highest priority. A variation on prioritized planning is to change the speeds at which trajectories are executed rather than generating new trajectories [35]. This allows, for example, speeding up the trajectory of a robot in a shared workspace, so it will exit the space more quickly, and slowing the trajectory of a robot en route to the shared workspace. The authors of [35] present an optimized method of manipulating the speed that generates continuous velocity profiles that respect dynamic constraints from given trajectories.

The prioritized planning works manipulate timing to achieve collision-free trajectories. This can be an effective way to perform given multi-arm spatial trajectories for situations in which timing is not critical. However, for tasks in which the two-armed motions must have a particular relative timing, this method cannot be used. In the proposed work, timing offsets between movements are calculated to optimize the likelihood of success.
Chapter 2. Background

Researchers have attempted to generalize two-handed motions. Hidden Markov Models have been used to determine key temporal, joint-space, and work-space points in two-handed motions demonstrated by humans [36]. To do this, a series of demonstrations are performed, and key points are extracted based on certain heuristics (e.g. a change in the direction of the tool center point). These key points are used to train HMMs for the movement. If a key point is representative of states in 4/5 of the demonstrations, it is considered a common key point for the arm. This is done for both hands to create two sets of common key points. If, in every demonstration, a key point for one arm precedes a key point for the arm, it is assumed to represent a temporal dependency in the two-arm task. These common key points are interpolated to generate trajectories for a simulated robot. They generated feasible-looking trajectories for a box pick-and-place task as well as a water-pouring task. This work demonstrate a technique for identifying temporal relationships between two-armed motions demonstrated for a robot. This could be a valuable tool for researchers working on two-handed motion generation. Though, this work has not yet been incorporated in motion planning outside of replaying demonstrated trajectories.

Ureche and Billard have studied automated methods to extract relationships between each arm’s movement in two-armed tasks [37] [38]. In their research, [37], they determine the key variable of interest at each point in time. Transitions between these variables indicate segmentation points. For each segment, the active and passive arm in the two armed task is determined allowing them to switch modelling schemes for the two arms – with the passive arm being modelled with respect to the forces generated by the first arm. In [38] they describe the control of a melon-scooping task in which one arm holds a melon, and the other a scoop.
Chapter 2. Background

The task is performed by a human. The human uses one of her arms, wearing a position- and force-sensing glove, directly to perform half of the task, and the other manipulates a robot arm to complete the other half of the task. Throughout the demonstrations, they record the position and force on the glove as well as the robot end-effector. They optically track the position of the melon. They use Granger Causality, a tool that determines if one variable can predict another, to determine causality between robot pose, wrench, object (scoop) pose, sensor signals, between each of the two robot hands. This work also has yet to be incorporated in a motion planning scheme.

This chapter has discussed the scope of different two-handed motion planning schemes. Some of the key challenges that have not been addressed in a single framework include: compatibility with tasks that have asymmetrical movements between the two arms, computational complexity, and optimizing timing for task performance. Chapter 3 presents a novel framework for two-handed coordination that addresses these areas. Additionally, this chapter has introduced a few valuable analytical tools for ascertaining important relationships between the motions of two arms. In the concluding chapter, I propose incorporating these techniques with the motion-planning framework presented in this thesis.
Chapter 3

Framework

In this chapter, I present the proposed framework for combining two single-armed trajectories to perform a two-armed task. This framework is designed to be applicable to a broad range of two-arm trajectories; that is, any task for which single-arm trajectories can be generated, and the relationship between the taskspace outcomes of each arm’s movement to the desired two-armed taskspace goal can be quantified. For such tasks, this framework overcomes some of the challenges discussed in the previous chapter.

The aim of this chapter is to demonstrate different ways in which the framework can be applied, as such it is intentionally unspecific. For the sake of simplicity, I use low-dimensional, simulated data. However, Chapter 4 includes full implementations and concrete steps to address the robot ping-pong serving problem and invokes many strategies in this chapter. Chapter 5 reports the results of real-robot experiments.

Each of Sections 3.1 through 3.4 addresses one component of the 4-component framework for creating two-handed trajectories. Each section explains the role of the component in the framework and introduces relevant notation. Examples unrelated to ping-pong serving are used to demonstrate how this framework could be applied to other tasks. Where relevant, the advantages of the approach taken are
Sections 3.1 and 3.2 explain how to build a trajectory library and model the effects of these trajectories on the taskspace. Building this library is a prerequisite to executing two-armed tasks. Section 3.3 describes problem-generic tools for combining trajectories based on their effect on the taskspace. Section 3.4 discusses avoiding collisions between the arms and the environment. Figure 3 depicts how these components fit together and serves as a roadmap/guide for the chapter.
Figure 3.1: The two-handed coordination framework. The headers in each box/circle loosely define each of the variables listed; for formal definitions see Sections 3.1, 3.2, 3.3, and 3.4.
3.1 Obtaining one-armed trajectories

As discussed above, the proposed framework combines two one-armed motions to create a two-armed motion. This section discusses the process of obtaining one-armed trajectories.

To start, the two-armed task must be divided into two one-armed tasks. Each one-armed task will be referred to as a subtask. For example in a two-armed sweeping task, one subtask might be manipulating a dustpan, the other manipulating a broom. The first subtask will be denoted $s_1$, the second, $s_2$.

Multiple one-armed trajectories are generated for each subtask. A trajectory is defined as a time-dependent path that can be followed by a robot, either in task- or joint-space. Often, these take the form of a list of time-stamped joint coordinates. Each trajectory can correspond to different ways to accomplish the same goal, or different goals. For example, two trajectories could move the dustpan to the same location through different paths, whereas another trajectory might move it to a different location. We define $T_{ij}$ to represent the $i$th trajectory for subtask $j$. Typically, each arm performs only one subtask, so the subtask numbering also identifies the arm being used.

Independently generating trajectories for each subtask has several advantages over simultaneously creating trajectories for both subtasks.

- There are many methods that already exist to create and modify one-armed trajectories. Any method to generate a one-armed trajectory can be used within this framework (e.g. optimal control methods [39] or learning from demonstration [40]).

- The trajectories can be stored in any representation. Some of these repre-
3.2. Probabilistic taskspace trajectory representations

sentations allow for the generation of multiple additional trajectories. For example, using DMP allows for generating trajectories that are qualitatively similar, but with different end points [41].

- Trajectories for one subtask can be combined with multiple trajectories for other subtasks. This creates a quadratic growth in possible combinations when compared to training independent sets of two-armed trajectories.\(^8\)

- For any training method that requires physically manipulating the robots, this approach eliminates the challenge of moving two robots accurately simultaneously.

3.2 Probabilistic taskspace trajectory representations

The result of a robot’s movement in taskspace can be difficult to calculate and inconsistent. In this framework, instead of predicting the taskspace outcomes of robot movements, they are empirically determined. These outcomes are modelled as probability distributions.

Many tools and objects have complex dynamics that are difficult to calculate. In some cases, a minor variation in initial conditions can result in a drastically different outcome. Even if the dynamics of the object and the initial conditions are perfectly known, calculations must also involve the dynamics and kinematics of the robots. Assuming an appropriate trajectory can be generated, the robot may not be able to successfully execute it due to limitations in the stiffness, power, sensing

\(^8\)That is, if there are \(n\) trajectories for the left arm and \(n\) trajectories for the right arm there are \(n^2\) combinations. Compared to creating \(n\) pairs of two-armed trajectories, this is a quadratic growth. In some cases, however, not every left-arm trajectory will be able to be combined with every right-arm trajectory which results in a sub-quadratic growth.

24
accuracy and control bandwidth of the robot.

The exact motion of the robot and the taskspace outcome may vary from execution to execution. The motion of the robot can vary with temperature, static, and dynamic friction. Additionally, environmental factors can affect the outcome. Varying lighting could affect sensor servoing. Wind and precipitation could also affect the taskspace outcome. For example, consider an environment where air currents vary due to doors and windows being opened or closed. An air current could interfere with the trajectory of a thrown object. For one trajectory, the taskspace effects could differ greatly depending on whether or not the currents are strong. However, for another trajectory that moves the arm in such a way that the projectile is shielded from the air current, the performance will be more consistent.

To compensate for these uncertainties, I use a probabilistic model of the taskspace outcome. This approach allows for tracking the overall effect of robot, tool, and environmental inconsistencies.

Modelling outcomes in taskspace has parallels to human motor learning. Studies suggest that focusing on the effect of a movement, rather than how to execute the movement, results in improved task performance [42]. Considering the outcome in task space allows for higher-level planning to be done. Instead of considering the exact joint angles required to move the arm in a certain way, computational effort can be dedicated to determining the most effective trajectory for completing the task.

To model the effects in taskspace, each trajectory is executed multiple times. The method of capturing the taskspace effects could involve, e.g., accelerometers, position sensors, cameras, magnetometers, etc. The taskspace outcome for the $m$th execution of trajectory $T_{s_j}^i$ is denoted $mO_{s_j}^i$. These outcomes can store the taskspace
3.2. Probabilistic taskspace trajectory representations

state after the motion has finished executing, or the state throughout the motion. If only the final state is of interest, a vector can be used:

\[
m^i_{O_s} = \begin{pmatrix}
s_j q_1 \\
s_j q_2 \\
\vdots \\
s_j q_{d_j}
\end{pmatrix}
\]

where \(d_j\) is the dimensionality of the taskspace associated with subtask \(s_j\) and each \(q_i\) is a taskspace variable.

If one is interested in how the state changes during and after the movement, from the initial time \(t_0\) to final time \(t_f\), this can be represented as a matrix:

\[
m^i_{O_s} = \begin{pmatrix}
s_j q_1(t_0) & s_j q_1(t_1) & \cdots & s_j q_1(t_f) \\
s_j q_2(t_0) & s_j q_2(t_1) & \cdots & s_j q_2(t_f) \\
\vdots & \vdots & \ddots & \vdots \\
s_j q_{d_j-1}(t_0) & s_j q_{d_j-1}(t_1) & \cdots & s_j q_{d_j-1}(t_f) \\
t_0 & t_1 & \cdots & t_f
\end{pmatrix}
\]

A probabilistic model is built for the outcome of each trajectory. \(m^i_{O_{s_j}}, \) for each \(m\), are used to train the probabilistic model for the effect of a particular trajectory. This model is denoted \(O^i_{s_j}\). This is done for all \(i\) such that each trajectory \(T^i_{s_j}\) has an associated probabilistic model, \(O^i_{s_j}\). This model can be queried to determine the probability of trajectory \(T^i_{s_j}\) yielding taskspace state

---

9The model need not be fitted to the raw captured data. The captured data can be modified or used to calculate additional data before a probabilistic model is created. For example, if a tool’s position is captured over time, one could pre-compute the velocity by taking the derivative. The velocity could then be modelled in addition to the position. Another example would be transforming the reference frame of captured Cartesian coordinates.
3.3. Combining two single-armed trajectories

\[
\mathbf{s_j q} = \begin{pmatrix}
    s_j q_1 \\
    s_j q_2 \\
    \vdots \\
    s_j q_d_j 
\end{pmatrix}.
\]

This probability is denoted \(O^i_{s_j}(s_j q)\).

In this work, I use full-covariance GMMs to model the outcomes. A GMM is a weighted sum of arbitrary-dimensional Gaussians. Each taskspace variable, \(s_j q_i\), can be represented by one of these dimensions. An expectation-maximization (EM) [43] algorithm is used to fit the means and standard-deviations of the Gaussians.

Because GMMs are sums of distributions (opposed to a single, e.g., normal distribution), they can be used to represent multi-modal distributions.\(^\text{10}\) The trajectory of a thrown object in an environment with a sporadic air current is an example of a multimodal outcome. The projectile will either veer in the direction of the current, or it will maintain its normal trajectory. Probabilistic models that do not allow for bi-modal distributions may try to average the two outcomes, resulting in a trajectory between the normal and the offset trajectories – something that is not at all representative of either actual taskspace outcome.

### 3.3 Combining two single-armed trajectories

Sections 3.1 and 3.2 discuss how to build a library of one-armed trajectories (\(T^i_{s_j}\)) for different subtasks (\(s_j\)) and associated probabilistic models (\(O^i_{s_j}\)). This section

\(^{10}\)These multi-modal distributions have found other uses in robotics. For example, in [44], Chan uses them to model different pathways for a manipulator to take through a nonconvex environment.
3.3. Combining two single-armed trajectories

discusses how to combine two one-arm trajectories to complete a two-armed task. In doing this, we must consider: maximizing the probability of success, the quality of the outcome, and timing.

The exact approach to combining trajectories is inherently task-dependent. However, many problems can be organized in a way that is compatible with this framework. First, subsection 3.3.1 explains the desired form for the problem. Subsections 3.3.2 through 3.3.8 introduce tools that can be used to phrase the problem in the desired form. Subsection 3.3.7 offers advice on coordinating relative and absolute timing of the two arms’ motions. Finally, 3.3.8 discusses ranking combinations of single-arm trajectories.

3.3.1 Goal

The goal of this framework is to combine two-single arm trajectories into a two-arm trajectory in order to achieve some goal. Due to the uncertainties and inconsistencies in performing a two-armed task, this is done by modelling the effects of the trajectories probabilistically.

Ultimately, the goal is to rank combinations of single-arm trajectories in order of the desirability of the taskspace outcome and the likelihood that the combination will produce that outcome. Using the earlier notation, the aim is to choose trajectories $T_{s1}^i$ and $T_{s2}^i$ based on optimizations of their respective taskspace models, $O_{s1}^i$ and $O_{s2}^i$, or a combined space, $O_{s1s2}^{i,\phi}$.

Criteria for desirability of taskspace outcome can include:

- Areas/points reached in each arm’s individual taskspace.
- Areas/points reached in a combined taskspace.
3.3. Combining two single-armed trajectories

- Probability of reaching these areas.

3.3.2 Tool - Mode finding

A *mode* of a probability density function refers to a local or global maximum. If all the modes are found, one can compare the probabilities in order to determine the global maximum. See Figure 3.2.

In mixture models, finding the modes is generally a numerical problem and different optimization techniques are used for different probability models. However, in this section I use the notation `mode()` to indicate an operation used to find all modes and their corresponding probabilities.

3.3.3 Tool - Marginal distribution

In statistics, a *marginal distribution* is the probability distribution of a subset of variables of the original distribution. Consider a distribution of variables $O_{s_j}(s_j q_1, s_j q_2)$. The marginal distribution, $O_{s_j}(s_j q_1)$, is a representation of the distribution’s dependence on $s_j q_1$. An example is illustrated in Figure 3.3.

When one wishes to ignore a variable in the task space, it can be marginalized out of the distribution. For example, if one captures the velocity of a tool as well as the position, but only wishes to use the position information in the optimization process for a particular task. Marginal distributions can also be used in determining when to start a trajectory. This is explained in subsection 3.3.7.
3.3. Combining two single-armed trajectories

Figure 3.2: A 2-component, 1-dimensional GMM. The black crosses indicate the modes of the distribution.
3.3. Combining two single-armed trajectories

Figure 3.3: A marginal distribution. The black dots are 1000 samples from a 3-component, 2-dimensional GMM. The green line indicates the marginal probability distribution. Note, the marginal distribution is one-dimensional, as it represents only the dependence of the distribution on $s_j q_1$. 
3.3. Combining two single-armed trajectories

In statistics, a \textit{conditional distribution} is used to find the probability of an event when it is known that the outcome is in some part of the sample space. For example, consider a probability distribution that is a function of two variables $O_{ij}(s_jq_1, s_jq_2)$. If one wants to determine the probability distribution over $s_jq_1$ for a given value, $C$, of $s_jq_2$, one may consider the conditional distribution $O_{ij}(s_jq_1|s_jq_2 = C)$. An example is illustrated in Figure 3.4.

12This figure is adapted from the documentation of the software [45].

Figure 3.4: A conditional distribution. The black dots are 1000 samples from a 3-component, 2-dimensional GMM. The red line indicates the conditional probability distribution. This represents the probability for different values of $s_jq_1$ given that $s_jq_2 = -1.$
3.3. Combining two single-armed trajectories

Conditional distributions are useful for considering the distribution of other state variables when one or several are constrained. For example, consider a robot pouring a drink from a bottle into a stationary cup. The taskspace might consist of bottle velocity and bottle position. Presumably, one wants the bottle velocity to be zero to initiate the pour. One could then consider the conditional distribution \( O_{s_j}^i(s_jq_{bp}|s_jq_{bv}=0) \), where \( q_{bp} \) is the variable representing the bottle position and \( q_{bv} \) is the variable representing the bottle velocity, to examine where the bottle is most likely to be when it comes to a rest in a particular trajectory.

Conditional distributions can also be used in determining when to start a trajectory. This is explained in subsection 3.3.7.

3.3.5 Tool - Joint distribution

A joint distribution represents the probability distribution of multiple events occurring. If these events are uncorrelated, the joint distribution is the product of each event’s probability distribution. An example is illustrated in Figure 3.5. Note that the mode of the joint distribution corresponds with the most likely value of \( s_jq_1 \) to be picked simultaneously from both distributions.

Creating the joint distribution of two taskspaces allows for optimizing relationships between the two subtask outcomes.

Consider a nail-hammering task. One arm swings a hammer (and its taskspace is the location of the face of the hammer) and another arm is holding a nail (and its taskspace is the location of the head of the nail). The joint probability distribution would represent overlaps between the location of the hammer and the location of the nail.
3.3. Combining two single-armed trajectories

Figure 3.5: A joint probability distribution. The joint distribution is the product of each individual distribution.

3.3.6 Tool - External modifications

In some cases, the taskspaces are easy to manipulate in a predictable way.

Consider the nail-hammering example, but with the hammer-holding arm on a mobile platform. One can find the most likely location of the hammer for trajectory \( i \) using \( \text{mode} (O_{1i}) \) and the most likely location of the nail for trajectory \( i' \) using \( \text{mode} (O_{2i'}) \). If the locations of the two maxima differ, the mobile platform could be moved until they agree.

3.3.7 Tool - Time shifting

To achieve a different starting time for the two-handed motions, one can choose to launch the robots earlier or later. One can also use relative time shifts between the two robots to optimize the probability of success.
3.3. Combining two single-armed trajectories

For example, consider the task of passing an item from one hand to the other (“regrasping”).\textsuperscript{13} For the sake of clarity in the accompanying figures, we will confine the movement of each hand to the $x$ axis.\textsuperscript{14} In this discussion we assume the existence of a separate re-grasping routine that can be executed when the hands are in close proximity and travelling slowly.

Let $s_1$ represent the Cartesian position of the left hand. Let $s_2$ represent the Cartesian position of the right hand. Consider trajectories $T_{s_1}^1$ and $T_{s_2}^1$ with probabilistic models $O_{s_1}^1(x, y, z, t)$ and $O_{s_2}^1(x, y, z, t)$. Sample data has been generated and plotted in Figure 3.6. Note that, unaltered, they intersect at $t \approx 1.6$ s. Observing the tangents of the position-time curves, we can obtain approximate velocities. The right hand has its lowest velocity at $t \approx 0.0$ s to $t \approx 0.5$ s and is at a higher velocity at the point of intersection. The left hand is at its slowest from $t \approx 3.0$ s to $t \approx 3.5$ s (after the intersection). To increase reliability of the re-grasp, it makes sense to execute the transfer when the hands have their lowest velocities.\textsuperscript{15}

To find the spatial location where the trajectories are most likely to overlap: one can marginalize out time from each distribution and multiply the resulting distributions. These are plotted in Figure 3.7. One can see that the two are most likely to overlap at $x \approx 0.78$ dm (i.e. $\text{mode}(O_{s_1}^1(x) \ast O_{s_2}^1(x)) \approx 0.78$ dm). This is consistent with our previous observations – this is the location where each arm is travelling the slowest.

We now strive to determine the optimal time delay for the two trajectories in

\textsuperscript{13}The authors of [46] consider a gradient-descent approach to the inverse kinematics of both arms to solve this problem.

\textsuperscript{14}This technique is used in Chapter 4 in order to time the movements of the ping-pong serve. In that case, all three spatial dimensions and time are considered.

\textsuperscript{15}The relative velocity between the end-effectors is the relevant quantity, but for simplicity, we consider velocity of each end-effector relative to a fixed frame. Considering the relationship of two arms’ state-spaces is dealt with in Chapter 4.
3.3. Combining two single-armed trajectories

Figure 3.6: The spatial coordinates of the left and right hands for an example re-grasping task.
3.3. Combining two single-armed trajectories

Figure 3.7: Marginal probability density functions for each hand for the regrasp task (top and middle) and joint probability density (bottom).
order for them to intersect at \( x \approx 0.78 \text{dm} \). To do this, we consider the conditional distributions \( O_{s1}^1(t|x = 0.78 \text{dm}) \) and \( O_{s2}^1(t|x = 0.78 \text{dm}) \). These distributions are plotted in Figure 3.8. We find that \( T_{s1}^1 \) is most likely to be at this coordinate when \( t \approx 3.32 \text{s} \) and \( T_{s1}^2 \) is most likely to be at this coordinate when \( t \approx 0.18 \text{s} \). This is again consistent with our initial observations. So the right hand should be delayed from the left hand by \( \Delta t \approx 3.32 \text{s} - 0.18 \text{s} = 3.14 \text{s} \) to optimize the probability of the two hands meeting.

The agreement of these results with the intuition from Figure 3.7 demonstrates that this is a powerful technique. This approach provides an algorithmic process for determining delays.

### 3.3.8 Ranking

These tools can all be used to generate lists of combinations of single-armed trajectories. Once this list is generated, the pairs should be ranked.

For example, the procedure in Subsection 3.3.7 could be iterated for every combination of trajectories forming a list of re-grasping points and associated probabilities of success. These could be ranked based on proximity to a desired re-grasping point, total amount of time required, total amount of distance travelled by each arm.

### 3.4 Collision Avoidance

Because coordination is only determined by taskspace compatibility, the robot arms could collide with each other or with their environment. In this framework, we check the top-ranked combinations for collisions before executing them.
3.4. Collision Avoidance

Figure 3.8: Conditional (temporal) probability densities of the left and right hands for an example re-grasping task.
3.5. Summary

Checking can be accomplished in many ways. Proposed schemes include neural networks [47, 48], GPU-based numerical algorithms [49], OBB trees [50], or physics-simulators (see Section 4.4), and many more.

If the top ranked combination is found to produce a collision, the next highest-ranked pair can be tested. This can be repeated until a non-colliding pair is found.

3.5 Summary

This chapter described the framework as a whole and several tools that can be used with it. This discussion was intentionally general and only considered simplified cases. However, the following chapter will demonstrate a concrete, real-robot application of the framework. Many of the tools discussed here are implemented and applied to the challenging case of a ping-pong serve.
Chapter 4

Implementation

The previous chapter described the concept of the framework and touched on various example applications. This chapter gives a complete implementation of the framework to achieve the challenging goal of coordinating two robot arms to serve a ping-pong ball. Sections 4.1 through 4.4 explain the specific implementation of the concept described in the corresponding sections of Chapter 3. The methods described in this chapter are used to execute the experiments described in Chapter 5. The final section provides a summary of these methods.

The ping-pong serving task is divided into two subtasks: $s_1$: throwing the ping-pong ball and $s_2$: swinging the paddle to hit the thrown ball. A laser-cut acrylic ball-holder is used for the throwing task, see Figure 4.1. A 3D-printed paddle holder is used for the second subtask with a modified ping-pong paddle.\textsuperscript{16} The paddle is depicted in Figure 4.2.

All software described developed by the author is available in a Bitbucket repository \cite{1}. For each reference to a software package, the ROS package (if applicable) and the source filename are included in a footnote.

\textsuperscript{16}The rubber is stripped from one face of the paddle. This is done to increase the sound of contact between the ball and paddle to improve accuracy for the measurements reported in Chapter 5.
Chapter 4. Implementation

Figure 4.1: Laser-cut acrylic ball-throwing hand. The joints were reinforced with J-B Weld (J-B Weld Company, Sulphur Springs, TX, USA) and the ball-holding cup was shimmed to a loose fit using cardboard and construction paper.

Figure 4.2: 3D-printed ping-pong paddle holder. The joints were reinforced using hot-melt glue. The 4 "fingers" that encompass the paddle handle are tightened using a strip of Velcro.
4.1 Obtaining one-armed trajectories

The control framework performs the two-armed ping-pong service task by choosing a set of two optimal single-armed trajectories from a library. This section presents a procedure for generating the one-armed trajectories in this library.

4.1.1 Generating robot motions

This section describes how to obtain joint trajectories for the robot arms. The workflow is depicted in Figure 4.3.

For both \( s_1 \) and \( s_2 \), one-armed trajectories are generated by hand-tuning start and end joint-coordinates using my WAM control software\(^{17} \) and GUI front end.\(^{18} \)

The GUI front end includes dials for start and end joint-coordinates and a dial for total trajectory time. After setting the coordinates, the user presses a button in the GUI to create a trajectory and send it to the WAM control software via ROS. The trajectory is a quintic interpolation\(^{19} \) [52] between the start and end joint-coordinates over the total trajectory time. First, the throwing trajectories are created. Using the GUI, the user tunes the quintic trajectory parameters until satisfied that the ball launches in an arc that intersects possible paths for the paddle. Subsequently, paddle trajectories are generated that intersect the ball trajectories.

This tuning process is similar to coaching practices in tennis where the athlete will practice, and the coach will tune, through observation and correction, the individual joint movements.

---

\(^{17}\)Robot Operating System (ROS) [51] package: wam_control, source files: ros_one_wam.cpp and ros_one_wam_right.cpp

\(^{18}\)ROS package: serve_ping_pong_balls source files: gui_generate_traj.py

\(^{19}\)ROS package: g_c, source file: quintic_trajectories.py. To construct the trajectory, the coefficients \((a_0, a_1, a_2, a_3, a_4, \text{ and } a_5)\) of the quintic trajectory for each joint, \( q(t) = a_0 + a_1 \cdot t + a_2 \cdot t^2 + a_3 \cdot t^3 + a_4 \cdot t^4 + a_5 \cdot t^5 \) where \( t \) is time, are set to constrain the initial and final velocity and acceleration to zero.
4.1. Obtaining one-armed trajectories

Figure 4.3: Workflow for generating one-armed trajectories. The diamond represents the user’s best-guess if the throw/swing is in range of the swing/throw from the other arm. The decision represented by the diamond does not need to be perfect as the algorithm can cope with extraneous trajectories. The number of trajectories, $I$ and $I'$ for subtasks $s_1$ and $s_2$, respectively, varies for the different experiments performed in Chapter 5.
4.1. Obtaining one-armed trajectories

vidual throwing and striking performances of the athlete with repeated drills before bringing the two together to practice the combined two-arm service motions.

The WAM control software is written in C++ and launches a ROS node. The node listens for a message that contains a joint trajectory in the form of time-stamped joint-coordinates. Once it receives a trajectory, it encodes it as a time-dependent spline\textsuperscript{20} and executes the trajectory while using internal joint sensors to record the achieved joint positions.\textsuperscript{21}

4.1.2 Capturing the effects of robot motions in taskspace

In order to make decisions about which trajectories to combine to achieve the two-armed goal, the framework must have information about the effect of each one-armed trajectory on the taskspace. This section discusses the approach used to capture the effects of robot trajectories on the taskspace. The process is depicted in Figure 4.4.

A Bumblebee (Point Grey Research, Vancouver, BC, Canada) stereo-vision camera is used to capture the effects in taskspace. A ROS node\textsuperscript{22} is used to capture images using the FireWire-camera protocol, libdc1394 [55] and to stream them as a ROS topic. These images are read and rectified using a second ROS node\textsuperscript{23}.

A ROS node to process the rectified images is launched.\textsuperscript{24} The image-processing

\textsuperscript{20}This encoding is required by and done using the WAM native library for servoing, Libbarrett [53].

\textsuperscript{21}The achieved joint positions are not necessarily equal to the commanded positions. The achieved positions are recorded for the sake of collision checking as described in Section 4.4.

\textsuperscript{22}ROS package: bumblebee_original, source file: grab_and_stream.cpp. This software is modified from the original software [54] to allow capturing from a stereo camera and to publish the images via ROS.

\textsuperscript{23}ROS package: stereo_image_proc, source file: stereo_image_proc.cpp. This node is included in my repository for convenience but is a verbatim copy of [56].

\textsuperscript{24}ROS package: camera_control, source file: time.py
4.1. Obtaining one-armed trajectories

Figure 4.4: Workflow for capturing taskspace outcomes. This procedure is repeated for every $i$ for both $s_1$ and $s_2$. 

Repeat $M = M' = 30$ times for each $i, j$. 

Legend

- State / Quantity / Data
- Tool / Technique
4.1. Obtaining one-armed trajectories

node awaits a request specifying a start time, a recording length, and a filename. This node checks the global ROS time. Once the time has reached the message time, videos are recorded in RAM for the specified duration. The node then applies HSV filtering to the rectified video frames in order to track the orange colour of the ball. The disparity of the center location of this colour blob in the left vs right images is used to calculate the 3D position of the ball. The video, with visual indicators of the detected ball location, is then written to disk using the specified file name. The calculated Cartesian ball coordinates are also written to disk as a comma-separated-values file. Having a node await a start time allows the node to complete all set up work in advance. This improves consistency in start time and allows for a global synchronizing with other nodes involved in the serving task.

An additional node for robot control is launched.\(^{25}\) This robot-control node differs from that discussed in Subsection 4.1.1 in two key ways. First, this node can control two robots simultaneously. This is the same software used to control the robots during the task execution discussed in Chapter 5. Using identical software helps limit the variance between the captured taskspace outcome and the taskspace performance during task execution. Secondly, this node uses the same scheme for time-synchronization as the image-processing node. It awaits a message containing two trajectories (a dummy, stationary trajectory is generated for the other arm), and a start time. The node completes all set-up work, then once the global ROS clock reaches the start time, executes the trajectory.

To capture the motion of the paddle swing for \(s_2\), a ping-bong ball is attached to the center of the paddle using Velcro.

After separately running \(M\) and \(M'\) executions of, respectively, \(I\)

\(^{25}\)ROS package: wam_control, source file: ros_two_wam.cpp
and $I'$ different robot trajectories for $s_1$ and $s_2$, we have a collection of CSV (Comma Separated Value) files of taskspace (time-stamped Cartesian) trajectories representing $\{O^1_{s_1}, \ldots, M O^1_{s_1}\}$, $\{1 O^I_{s_1}, \ldots, M O^I_{s_1}\}$ and $\{1 O^1_{s_2}, \ldots, M O^1_{s_2}\}$, $\{1 O^I_{s_2}, \ldots, M O^I_{s_2}\}$.

### 4.2 Probabilistic taskspace trajectory representations

In order to compensate for variances in the taskspace effect of a particular trajectory, their outcomes must be modelled probabilistically. This section explains how to create probabilistic models to represent the outcomes of each robot trajectory from the collection of individual outcomes. The process is depicted in Figure 4.5.

A GMM, $O^i_{s_1}$, is created for the outcomes of each arm trajectory associated with a ball throw, $T^i_{s_1}$. Similarly, a GMM, $O^i_{s_2}$, is created for the outcomes of each arm trajectory associated with a swing, $T^i_{s_2}$. To do so, all the CSV files representing taskspace outcomes are loaded into a python script. For the throw trajectories, this script first truncates each file to the section of values representing the downward travel of ball. The script then fits the GMM using the EM algorithm implemented in Scikit-learn [57]. The GMM consists of 5, 4-dimensional Gaussians (one for each Cartesian direction and one for time).

---

26ROS package: serve_ping_pong_balls, source file: gui_controller.py.


28This is determined by establishing the velocity of the ball by taking the derivative of the ball position found using the camera, projecting the velocity on a unit vector known to correspond to the negative $z$-direction in the world frame, determining its sign, and taking the longest temporal stretch with positive signs.

29This number was chosen by experimentation with the aim of keeping the number low while still capturing the trajectory shape. When two GMM are multiplied, as is done in finding a joint probability, the product GMM has as many Gaussians as the product of each of its constituents’ Gaussian counts. This increases computational difficulty of mode finding substantially.
4.2. Probabilistic taskspace trajectory representations

Figure 4.5: Create Gaussian Mixture Models from taskspace outcomes. This procedure is repeated to create a probabilistic model for every trajectory.
4.3 Choosing trajectory combinations

Having each available single-arm trajectory’s effect on taskspace quantified, the goal is now to use this information to combine single-arm trajectories with the appropriate timing to achieve a taskspace goal, in this case, performing a legal ping-pong serve. The procedure is demonstrated in Figure 4.6.

![Diagram of trajectory combinations](image)

Figure 4.6: Pairing trajectories and ranking them by their effect on the combined taskspace.

The immediate goal is to identify the throwing and paddle-swinging trajectories that effect an overlap between the ping-pong ball and the paddle.

To do so, we marginalize out time from each distribution\(^{30}\) (see Subsection

\(^{30}\)ROS package: gmm_center, source file: matlab_gmm_tools.py . This package calls the Matlab script [58].

50
4.3. Choosing trajectory combinations

3.3.3), create the joint probability of each pair of time-free models $O_{s1}^i$ and $O_{s2}^{i'}$ for each $i$ and $i'$ (see Subsection 3.3.5), and find their modes (see Subsection 3.3.2). After this step, we have acquired a list of spatial coordinates for the modes of each swing and throw combination, $\text{mode}(O_{s1}^i * O_{s2}^{i'})$ for each $i$ and $i'$.

For each spatial mode, we find the optimal offset, $\Delta t$, in starting times of the two trajectories as described in Subsection 3.3.7. To calculate the probability of obtaining contact between the ball and paddle, one can use a Monte Carlo approach. First, time can be discretized into intervals of $dt$. For each interval, the probabilistic models for the throw and swing (with the appropriate time offset) can be sampled $L$ times. For each pair of samples, the Cartesian distance can be calculated. If this distance is smaller than the width of the ping-pong paddle, that sample pair can be considered a hit. Let $H$ represent the number of samples that are hits. Then the probability of a collision in time interval, $ti$, is given by $p_{\text{collision}}^{ti} = \frac{H}{L}$. Therefore, the probability of no collision in that time interval is given by $p_{\text{no collision}}^{ti} = 1 - p_{\text{collision}}^{ti}$. Finally, to calculate the probability of a collision in any time window, we compute $p_{\text{collision}} = 1 - p_{\text{no collision}}^0 * p_{\text{no collision}}^1 * \cdots$. This leaves us with a probability of the ball and paddle intersecting at, approximately,

---

31 ROS package: gmm_center, source file: gmm_python_tools.py.
32 ROS package: gmm_center, source file: interface_mode_findcpp.cpp. I ported this code from the Matlab script [58] to C++ to improve performance. The C++ performs 10x faster than the matlab script on the computer used for the experiment described in Chapter 5.
33 The numbers produced here do not correspond to actual probabilities, however are an indicator of relative probability of success as compared to other trajectory combinations. See Chapter 5 for experimental results.
34 Note: for small time windows and large numbers of samples, this method is very time consuming. However, one can quickly calculate a non-normalized approximation to these values using the equation: $p_{\text{collision}}^{\text{normalized}} = \int \int \int \int_\text{all} O_{s1}^i(x, y, z, t) * O_{s2}^{i'}(x, y, z, t + \Delta t) dx dy dz dt$. This equation can be efficiently computed using the technique for integration of Gaussians presented in [59]. This approach has been tested on the experimental data presented in the following chapter as well as on simulated data. It yields different probabilities, but the same probability ranking as the Monte Carlo approach. Both approaches are implemented in ROS package: gmm_center, source file: integration_techniques.py.
that spatial mode when the robot trajectories are executed with the calculated time offset.

4.4 Collision checking

Having pairs of trajectories and a time-offset to perform the two-armed task, we must ensure that the arms will not collide with each other or obstacles in the environment. This is done using the robot simulation tool, GazeboSim [60]. For a flowchart of this procedure, see Figure 4.7.

The GazeboSim world file created for this purpose contains the two WAM arms, a ping-pong table, and the desks the robots are mounted on. This simulation is controlled via a ROS node. This node mirrors the interface for the node used to control the robot arms discussed in Subsection 4.1.2.

The trajectory pairs (and corresponding time-delays) produced as described in the last section are ranked according to probability. In order of highest to lowest probability, the trajectory pairs are run in the simulation environment. Any trajectory combination that results in an unintended collision is dismissed, and the next best choice is investigated. The first pair of throw and swing trajectories found that do not collide then represent the combination with the highest probability of producing a paddle-ball hit and may be executed on the actual robots.

---

35ROS package: caris_ping_pong_setup_gazebo.
36ROS packages: left_wam_description, right_wam_description.
37ROS package: ping_pong_table.
38ROS packages: robot_island_wood_bridge, robot_island_table.
39ROS package: g_c, source file: gazebo_control.py.
40We use the achieved joint trajectories rather than the commanded joint trajectories.
4.4. Collision checking

Figure 4.7: Collision-checking procedure. Each pair of trajectories is tested with their optimal, relative time delay until a collision-free pair is found.
4.5 Summary of contributions

This chapter presents an implementation of the framework explained in Chapter 3. This includes a novel algorithm for combining two single-arm trajectories, novel hardware, and novel software. In this section, I summarize the contributions discussed in this chapter.

In the introduction to the chapter, I introduced novel hardware designs for robot-attachments. I introduced a laser-cut, acrylic ball thrower that attaches to a WAM robot. I also introduced a 3D-printed paddle holder that attaches to a WAM robot.

In Section 4.1, I introduced a novel GUI for tuning the parameters of a quintic trajectory for the WAM robot. I also discussed novel software to control a WAM arm using ROS. The section also describes novel software to capture images from a Bumblebee and publish them via a ROS topic, and software to perform HSV filtering and track the ball location. The section also discusses a scheme for synchronizing the camera and robot start times.

Section 4.2 describes a front-end for controlling existing GMM libraries and a method for capturing only the downward portion of ball trajectory.

Section 4.3 explains my software implementation of my novel algorithm for combining two-arm trajectories. This algorithm includes methods for estimating the best pair of single-arm trajectories to perform a two-armed task as well as optimal time delays between the two single-arm trajectory starts.

Finally, Section 4.4 explains my novel plugins to check two WAM trajectories for collisions in simulation.

---

41This is important as a ping-pong serve is only legal if the paddle strikes the ball while it is travelling downwards.
Chapter 5

Experiments and Demonstrations

In the previous chapters, several challenges in two-arm motion planning were discussed. In this chapter, I make specific hypotheses about the framework’s capability of addressing these challenges and experimentally test them.

5.1 Experiment 1: Ability to choose the best trajectory combination

For each pair of $s_1$ (throwing) and $s_2$ (paddle-swinging) trajectories, the algorithm provides a probability of ball-paddle contact. As discussed in Section 4.3, this value is found using a Monte Carlo technique.\(^{42}\) This provides a measure of how likely the ball and paddle are to intersect for a given throw, swing, and time delay. In this experiment, we assess if these probabilities can be used to rank the swing/throw combinations in terms of likelihood of a ball/paddle hit. Subsection 5.1.4 provides a discussion on why only the ordering of these probabilities is meaningful.

\(^{42}\)In particular, these values were used: $dr = 0.01s$, $L = 1000000$, and the integration was conducted over $-55m < x, y, z < 55m$ and within $0.1s$ of the temporal mode. The spatial limits were chosen to encompass all recorded position values. The temporal limits were chosen to encompass the time with the highest probability of overlap (while keeping computation times manageable).
5.1. Experiment 1: Ability to choose the best trajectory combination

5.1.1 Hypothesis

Claim 1. The algorithm outputs a probability for each swing/throw combination to successfully strike a ball; the ordering of these probabilities corresponds to the actual ordering of swing/throw combination success.

5.1.2 Method

A throw and swing trajectory were created such that the ball’s trajectory intersected, approximately, the paddle’s center point.

The ball’s trajectory was changed by adjusting both the start- and end- coordinates of the throwing robot’s joint 6 together. By changing this joint value from 0.17 rad to 0.37 rad in increments of 0.05 rad, the intersection of the ball and paddle ranged from one side of the paddle to the other. These trajectories were chosen because of the paddle’s oval shape. Changing the mean horizontal point of contact brings the ball closer to the edge of the paddle, simultaneously bringing it closer to one side, the top, and the bottom. Therefore, this array of contact points corresponds to an array of allowed variance while still obtaining a hit.

For each value of joint 6, a probabilistic model of the throw was trained using \( M = 30 \) executions. Additionally, for each run, the swing model was retrained using \( M' = 30 \) runs.\(^\text{43}\) From these models, a temporal mode was found. A total of \( n = 30 \) executions of the combination were performed with the algorithm-recommended time delay.

The number of hits was counted by an experimenter positioned close to the

\(^{43}\)The same swing trajectory was used for each throw. This retraining was done to compensate for any possible deviation in the calibration of the robot.
5.1. Experiment 1: Ability to choose the best trajectory combination

Table 5.1: The number of successful hits for each different throwing trajectory (Experiment 1).

<table>
<thead>
<tr>
<th>Joint 6 (rad)</th>
<th>0.17</th>
<th>0.22</th>
<th>0.27</th>
<th>0.32</th>
<th>0.37</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contacts between ball and paddle (out of 30 attempts) (c)</td>
<td>0</td>
<td>18</td>
<td>29</td>
<td>29</td>
<td>24</td>
</tr>
<tr>
<td>Algorithm-predicted probability (p)</td>
<td>$4.00 \times 10^{-6}$</td>
<td>$9.90 \times 10^{-5}$</td>
<td>$8.55 \times 10^{-3}$</td>
<td>$2.24 \times 10^{-1}$</td>
<td>$2.77 \times 10^{-1}$</td>
</tr>
<tr>
<td>Time delay ($\Delta t$) (s)</td>
<td>-0.561</td>
<td>-0.511</td>
<td>-0.547</td>
<td>-0.499</td>
<td>-0.570</td>
</tr>
</tbody>
</table>

Table 5.2: Standard deviation of the ball and paddle positions at the temporal mode (Experiment 1).

<table>
<thead>
<tr>
<th>Joint 6 (rad)</th>
<th>0.17</th>
<th>0.22</th>
<th>0.27</th>
<th>0.32</th>
<th>0.37</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard deviation of paddle location ($x, y, z$) ($\sigma_p$) (m)</td>
<td>(0.053, 0.093)</td>
<td>(0.077, 0.058)</td>
<td>(0.068, 0.051)</td>
<td>(0.060, 0.100)</td>
<td>(0.028, 0.037)</td>
</tr>
<tr>
<td>Standard deviation of ball location ($x, y, z$) ($\sigma_b$) (m)</td>
<td>(0.096, 0.176, 0.080)</td>
<td>(0.076, 0.177, 0.117)</td>
<td>(0.067, 0.177, 0.079)</td>
<td>(0.880, 0.745, 1.27)</td>
<td>(0.114, 0.387, 0.121)</td>
</tr>
<tr>
<td>Magnitude of $\sigma_b$ (m)</td>
<td>0.216</td>
<td>0.226</td>
<td>0.205</td>
<td>1.712</td>
<td>0.421</td>
</tr>
<tr>
<td>Magnitude of $\sigma_p$ (m)</td>
<td>0.076</td>
<td>0.111</td>
<td>0.160</td>
<td>0.140</td>
<td>0.123</td>
</tr>
</tbody>
</table>

contact point using both audio\textsuperscript{44} and visual cues.

5.1.3 Results and analysis

The results are documented in Table 5.1. Additionally, for each pair of throwing and swinging trajectories, the standard deviations of the position of the paddle and ball at the temporal mode were calculated.\textsuperscript{45}

Let the number of contacts be denoted $c$, and the number of attempts be denoted $n (= 30)$. Let the true (experimental) probability be denoted $r = \frac{c}{n}$. For our

\textsuperscript{44}The rubber was stripped off the contact face of the paddle to increase the sound of a hit.

\textsuperscript{45}Specifically, the temporal modes of the throw and swing trajectories refer to the time when the ball and the paddle, respectively, are most likely to be at the spatial mode where the two trajectories are most likely to intersect. Refer to Subsection 3.3.7 for details.
5.1. Experiment 1: Ability to choose the best trajectory combination

analysis, we wish to find the uncertainty in the experimentally-determined probabilities. This problem is equivalent to finding the true probability and variance in a binomial distribution. The standard variance is given by

$$\sigma = \sqrt{\frac{(1-r)r}{n}}.$$ 

Multiplying the standard variance by a z-score of 2 gives us the 95% confidence window. Let the uncertainty in the experimental probability be denoted $\Delta r$. Then,

$$\Delta r = 2\sqrt{\frac{(1-r)r}{n}}.$$ 

These results are plotted in Figure 5.1.

5.1.4 Discussion

The results show that the algorithm successfully predicted the order of swing-throw combinations from most likely to least likely to achieve ball-paddle contact for four out of five combinations (with the exception being within experimental uncertainty of being correct).

With the exception of the anomalous point, the two swing-throw combinations with the highest predicted probabilities ($2.24 \times 10^{-1}$ and $2.77 \times 10^{-1}$) had the best performance with, identically, $c = 29$ contacts out of 30 tries. It is possible that there is a threshold probability beyond which performance no longer improves.

The probability was generated using a very simple metric – overlapping points in taskspace. It is possible that a more complex metric – perhaps including paddle orientation and relative and absolute velocities – could eliminate the one inaccurate prediction.

Some of the algorithm-predicted probabilities are several orders of magnitude

---

46 Note: this formula fails for $r = 0$. In this case, a more conservative estimate on the error is used and $r$ is set to $r = \frac{1}{30}$. 

58
5.1. Experiment 1: Ability to choose the best trajectory combination

Figure 5.1: Experimentally-determined probability of success vs algorithm-predicted probability of success. Each datapoint corresponds to $M = 30$ and $M' = 30$ training trials for the throw and swing respectively and $n = 30$ attempted serves. The annotations on the plot indicate the angle of the robot’s joint 6. The error bars indicate a 95.45% confidence interval. See text for the derivation of the uncertainty.
smaller than the experimentally-determined probabilities. To determine the cause of this, the standard deviation for the ball and paddle location were calculated at the temporal modes – the time when the ball and paddle were expected to contact. These values are given in Table 5.2. One can note that the standard deviations were very large. The smallest standard deviation in ball position is 0.205 m – a distance approximately 10 times the radius of the ping-pong ball. The greatest standard deviation is over 85 times the radius. This indicates that the data captured by the camera was noisy so the GMM had very widely spatio-temporally distributed data yielding only a small probability of sampled values overlapping. Indeed, this can be seen from the ball and paddle position data plotted as Figures 5.2 through 5.6.

Figure 5.2: Cartesian locations of the ball and paddle (Experiment 1) for $j_0 = 0.17$ radians captured during all $M = 30$ executions. The blue line indicates the time when the paddle and ball are most likely to intersect.
5.1. Experiment 1: Ability to choose the best trajectory combination

Figure 5.3: Cartesian locations of the ball and paddle (Experiment 1) for $\theta_6 = 0.22$ radians. Data represent all $M = 30$ executions. The blue line indicates the time when the paddle and ball are most likely to intersect.
5.1. Experiment 1: Ability to choose the best trajectory combination

Figure 5.4: Cartesian locations of the ball and paddle (Experiment 1) for $\theta_6 = 0.27$ radians. Data represent all $M = 30$ executions. The blue line indicates the time when the paddle and ball are most likely to intersect.
5.1. Experiment 1: Ability to choose the best trajectory combination

Figure 5.5: Cartesian locations of the ball and paddle (Experiment 1) for $j_6 = 0.32$ radians. Data represent all $M = 30$ executions. The blue line indicates the time when the paddle and ball are most likely to intersect.
5.2. Experiment 2: Timing

Figure 5.6: Cartesian locations of the ball and paddle (Experiment 1) for $j_6 = 0.37$ radians. Data represent all $M = 30$ executions. The blue line indicates the time when the paddle and ball are most likely to intersect.

5.1.5 Conclusion

These results support the claim that the algorithm-generated probabilities are useful for predicting relative performance between throw/swing trajectory combinations. The algorithm was successful in the rank prediction for 4 out 5 pairs despite extensive noise in the taskspace data captured by the camera.

5.2 Experiment 2: Timing

For each pair of $s1$ and $s2$ trajectories, the algorithm calculates a relative timing offset using the corresponding probabilistic models. The swing execution is delayed from the throw by this amount of time. This is calculated as discussed in
Subsection 3.3.7. In this experiment, we assess the performance of the algorithm-recommendations for offsets.

5.2. Hypothesis

Claim 2. The algorithm can determine time offsets between the start of the left-arm trajectory and the right-arm trajectory that allow it to perform two-handed tasks.

5.2.2 Method

The aim of this experiment was to determine how close the algorithm-recommended time delay between the throw and the swing trajectories is to the optimal delay. In particular, the aim was to determine the time window (within \( \pm 0.0075 \) s) that would yield satisfactory performance (defined as achieving \( c \geq 25 \) ball and paddle contacts out of \( n = 30 \) attempts) and compare this to the algorithm-recommended time.

The following procedure\(^{47,48}\) is used to determine the upper bound of the window:

From the time recommended by the algorithm, \( T_0 \), a search (each trial consisted of adjusting the time delay, and 30 serve attempts) was conducted in increments of 0.003 s until 25 or more ball/paddle contacts were achieved (\( c \geq 25 \)). The search

Note: the \( T_3^3 \) and \( T_2^3 \) combination (results in Table 5.3) has more data points than the procedure dictates. The investigation of this combination was done while still characterizing the system and extra data points were taken to ensure the time-window precision and the definition of satisfactory performance were appropriate for the ping-pong serving task.

This procedure is similar to binary search on an infinite-length array in the field of data structures and algorithms. Binary search is a notoriously difficult algorithm to implement correctly. In a study of 25 computer science textbooks, only 5 implementations were found to be correct \([61]\). The procedure in this Section is meant to outline an efficient technique to find the time bounds of satisfactory performance but is not guaranteed to be the most efficient or cover all corner cases. The important part of the procedure is to ensure the bounds are found within 0.0075 s, which has been done for this experiment.
was continued with the same increment until performance dropped to \( c < 25 \). Then another search was conducted in between the last point before performance fell \((c < 25)\) and the last point with \( c \geq 25 \) using an increment of 0.015 s. This process yields two points within 0.015 s of each other straddling the point where performance became less than satisfactory \((c < 25)\). The mean between the two numbers indicates the upper edge of the timing window with an uncertainty of \( \pm 0.0075 \) s. The procedure was repeated in reverse to determine the lower time bound of the satisfactory performance window.

The data are reported in Subsection 5.2.3 in the order they were collected. The symbol \( \Delta t \) is used in the tables and figures to denote the offset from the algorithm-recommended time. As noted in the footnote above, the first swing/throw pair data was collected using a different procedure.

### 5.2.3 Results and analysis

The experiment was carried out for 5 different swing and throw combinations. These results are shown in Tables 5.3 through 5.7 and plotted in Figure 5.7. Table 5.8 summarizes the satisfactory-performance time window for each pair of throwing and swinging trajectories.\(^{49}\)

In three of the five pairs \((T_{s_j}^1, T_{s_j}^4, \text{and } T_{s_j}^5)\) the suggested time is within the satisfactory performance window. The best performing pair, \( T_{s_j}^5 \), was optimal to within less than the accuracy of the experiment; \( t_m^5 = -0.0008 \) s was less than the smallest increment used to determine the window size, \((0.015 \) s\). The worst performing pair, \( T_{s_j}^2 \), varied from the optimal time by \( t_m^2 = 0.0750 \) s. The average (absolute)

---

\(^{49}\)If a measurement was made that yielded \( c = 25 \), the time for that measurement is used instead of an average.
Table 5.3: The number of hits for different offsets from the algorithm-recommended time using trajectories $T_{s1}^5$ and $T_{s2}^5$. The data points are reported in the order they were collected. See Footnote 47 for more information on this dataset.

<table>
<thead>
<tr>
<th>Delay from recommended time ($\Delta t$) (s)</th>
<th>Number of contacts (out of 30) (c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>30</td>
</tr>
<tr>
<td>-0.001</td>
<td>30</td>
</tr>
<tr>
<td>-0.003</td>
<td>28</td>
</tr>
<tr>
<td>-0.005</td>
<td>30</td>
</tr>
<tr>
<td>-0.007</td>
<td>30</td>
</tr>
<tr>
<td>-0.015</td>
<td>29</td>
</tr>
<tr>
<td>-0.031</td>
<td>27</td>
</tr>
<tr>
<td>-0.063</td>
<td>7</td>
</tr>
<tr>
<td>-0.047</td>
<td>14</td>
</tr>
<tr>
<td>-0.039</td>
<td>21</td>
</tr>
<tr>
<td>0.030</td>
<td>27</td>
</tr>
<tr>
<td>0.060</td>
<td>8</td>
</tr>
<tr>
<td>0.045</td>
<td>21</td>
</tr>
<tr>
<td>0.037</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 5.4: The number of hits for different offsets from the algorithm-recommended time using trajectories $T_{s1}^4$ and $T_{s2}^4$. The data points are reported in the order they were collected.

<table>
<thead>
<tr>
<th>Delay from recommended time ($\Delta t$) (s)</th>
<th>Number of contacts (out of 30) (c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>30</td>
</tr>
<tr>
<td>-0.030</td>
<td>22</td>
</tr>
<tr>
<td>-0.015</td>
<td>29</td>
</tr>
<tr>
<td>0.030</td>
<td>30</td>
</tr>
<tr>
<td>0.060</td>
<td>28</td>
</tr>
<tr>
<td>0.090</td>
<td>11</td>
</tr>
<tr>
<td>0.075</td>
<td>23</td>
</tr>
</tbody>
</table>
5.2. Experiment 2: Timing

Table 5.5: The number of hits for different offsets from the algorithm-recommended time using trajectories $T_{s1}^3$ and $T_{s2}^3$. The data points are reported in the order they were collected.

<table>
<thead>
<tr>
<th>Delay from recommended time ($\Delta t$) (s)</th>
<th>Number of contacts (out of 30) (c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>10</td>
</tr>
<tr>
<td>-0.030</td>
<td>27</td>
</tr>
<tr>
<td>-0.060</td>
<td>30</td>
</tr>
<tr>
<td>-0.090</td>
<td>29</td>
</tr>
<tr>
<td>-0.120</td>
<td>12</td>
</tr>
<tr>
<td>-0.015</td>
<td>25</td>
</tr>
<tr>
<td>-0.105</td>
<td>27</td>
</tr>
</tbody>
</table>

Table 5.6: The number of hits for different offsets from the algorithm-recommended time using trajectories $T_{s1}^2$ and $T_{s2}^2$. The data points are reported in the order they were collected.

<table>
<thead>
<tr>
<th>Delay from recommended time ($\Delta t$) (s)</th>
<th>Number of contacts (out of 30) (c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0</td>
</tr>
<tr>
<td>0.030</td>
<td>15</td>
</tr>
<tr>
<td>0.060</td>
<td>30</td>
</tr>
<tr>
<td>0.090</td>
<td>30</td>
</tr>
<tr>
<td>0.120</td>
<td>17</td>
</tr>
<tr>
<td>0.105</td>
<td>28</td>
</tr>
<tr>
<td>0.045</td>
<td>23</td>
</tr>
</tbody>
</table>

Table 5.7: The number of hits for different offsets from the algorithm-recommended time using trajectories $T_{s1}^1$ and $T_{s2}^1$. The data points are reported in the order they were collected.

<table>
<thead>
<tr>
<th>Delay from recommended time ($\Delta t$) (s)</th>
<th>Number of contacts (out of 30) (c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>27</td>
</tr>
<tr>
<td>0.030</td>
<td>30</td>
</tr>
<tr>
<td>0.060</td>
<td>30</td>
</tr>
<tr>
<td>0.090</td>
<td>4</td>
</tr>
<tr>
<td>0.075</td>
<td>24</td>
</tr>
<tr>
<td>-0.030</td>
<td>0</td>
</tr>
<tr>
<td>-0.015</td>
<td>6</td>
</tr>
</tbody>
</table>
5.2. Experiment 2: Timing

Figure 5.7: Success vs time shift from the algorithm-recommended time delay between throwing the ball and swinging the paddle. Vertical lines are placed at 0.0 s to indicate the algorithm-recommended time. If the time was exactly optimal, the peak in performance would occur at this x-value. Horizontal lines are placed at 0.83 indicating \( c = 25 \) successful hits out of \( n = 30 \) attempts.
5.2. Experiment 2: Timing

Table 5.8: The lower and upper bound on the satisfactory performance windows for different combinations of swing and throwing trajectories. These values are given as additions to the algorithm-recommended time.

<table>
<thead>
<tr>
<th>Trajectory pair</th>
<th>Lower bound ( (t^l_j) ) (s)</th>
<th>Upper bound ( (t^u_j) ) (s)</th>
<th>Middle of window ( (t^m_j) ) (s) (average of ( t^l_j ) and ( t^u_j ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_{s1}^1 ) and ( T_{s2}^2 )</td>
<td>-0.0075</td>
<td>0.0675</td>
<td>0.0300</td>
</tr>
<tr>
<td>( T_{s1}^2 ) and ( T_{s2}^1 )</td>
<td>0.0375</td>
<td>0.1125</td>
<td>0.0750</td>
</tr>
<tr>
<td>( T_{s1}^3 ) and ( T_{s2}^2 )</td>
<td>-0.015</td>
<td>-0.1175</td>
<td>-0.0663</td>
</tr>
<tr>
<td>( T_{s1}^4 ) and ( T_{s2}^3 )</td>
<td>-0.0225</td>
<td>0.0675</td>
<td>0.0225</td>
</tr>
<tr>
<td>( T_{s1}^5 ) and ( T_{s2}^2 )</td>
<td>-0.0350</td>
<td>0.0335</td>
<td>-0.0008</td>
</tr>
</tbody>
</table>

distance from the optimal time taken over all 5 pairs was 0.0389 s.

5.2.4 Discussion

This experiment has provided insight into the accuracy of the algorithm’s suggested time offsets. The camera used to capture the ball and paddle locations operated at 30 hz. The average distance from the optimal time was then only 1 \( \frac{1}{3} \) frames. On average then, the algorithm performed at least as well as the data from the camera would allow. In some cases, the algorithm performed as well as 0.024 frames. The worst performance was 2.25 frames. This suggests that harnessing multiple data sets and creating a GMM allowed for performance on the order of the sensor capabilities.

5.2.5 Conclusion

The algorithm performed well at recommending timings. In three throw/swing combinations, the algorithm’s time was more precise than sensor data from a single trial would have allowed. This indicates that using a probabilistic model improves performance.
5.3 Demonstration: Ping-pong serve

Executing a ping-pong serve is a difficult task requiring precise timing. Throughout this work, this task has been used as a benchmark for the algorithm. In the previous two experiments, the task was used to demonstrate the ability of the algorithm to choose the best available trajectory combinations and to choose a correct time offset. Those two experiments measured the success at achieving a ball and paddle contact. This leaves the question: Can this framework be used to accomplish tasks with more sophisticated objectives? To illustrate that it can, the algorithm was used to perform a legal ping-pong serve.

5.3.1 Hypothesis

*Claim 3.* This method of generating trajectories and the algorithms ability to combine two one-armed trajectories can be used to perform a legal ping-pong serve.

5.3.2 Method

One trajectory for each subtask was trained such that the throw caused the ball to intersect the paddle path. A probabilistic model was trained for each.

A Stiga (Eskilstuna, Sweden) ping-pong table with regulation Joola (Gödramstein, Germany) net was set up in front of the robot arms (Figure 5.8).

The algorithm-recommended offset was used to perform a serve. Upon seeing the trajectory of the serve, a correction of 0.2 radians was added to the start and end coordinates of the swinging robot’s joint 7. This adjustment made the serve follow a legal trajectory. Thirty serves were attempted.
5.3. Demonstration: Ping-pong serve

Figure 5.8: Setup for the ping-pong serve demonstration.

5.3.3 Results

Of the 30 serves attempted, all yielded contact between the ball and the paddle. In 25 of the serve attempts, the serve was legal – the throw was approximately vertical and was at least 16 cm high. The ball was struck while falling, bounced once on the robot’s side of the table, cleared the net, and landed on the opposite side of the table. This is an 83% success rate.

5.3.4 Discussion

With minor adjustment to one of the initial trajectories, the algorithm was capable of coordinating a legal ping-pong serve. In this case, the experimenter made the adjustment. This tweak was specific to the ping-pong task, though the algorithm was designed to have broader application. However, if one were to focus on solely the ping-pong serving task, such single-armed trajectory tweaks could be incorporated into design and the parameters could be tuned automatically considering, for example, the orientation of the face of the paddle, relative velocity between the paddle...
and ball, etc. The framework has no dependency on how one-armed trajectories are generated or modified. Therefore, more sophisticated, automated methods than manually tuning quintic start- and end- points could be used for different tasks.

5.3.5 Conclusion

The algorithm correctly identified the timing offset to perform a legal ping-pong serve 25 times out of 30 attempts. This demonstrated the ability of the algorithm to be used for tasks with more complicated taskspace objectives.

5.4 Experiments 1 & 2 and service demonstration: discussion and conclusion

In this chapter, we have demonstrated that the algorithm is capable of predicting the relative performance of different combinations of trajectories, accurately determining the timing offset between two single-armed trajectories, and using these abilities to perform a two-armed task.

Additionally, it was shown that the algorithm could perform well with limited data. In one case, the recommended timing was more precise than the frame rate of the camera. The average precision in recommended timing was on the order of the frame rate of the camera.

The algorithm achieved these successes in recommending timing offsets and predicting the relative success of throw/swing combinations despite significant noise in the taskspace data captured by the camera. This demonstrates the usefulness of modelling taskspace outcomes probabilistically and using a collection of outcomes to make predictions about future outcomes.
5.4. Experiments 1 & 2 and service demonstration: discussion and conclusion

While the benchmark used in this thesis is performing a ping-pong serve, the algorithm is agnostic to the task being performed. The optimization was based solely on the relationship between two taskspace states – with no knowledge of the task being performed. It is important to note that the optimization was done with no knowledge of the physical process that generated the data; time-dependent lists of taskspace coordinates were generated by a stereovision camera and given to the timing and probability-predicting routines without any knowledge of how the data sets were generated. These results suggest that the algorithm could be used for a variety of two-armed tasks.
Chapter 6

Conclusion

6.1 Summary of contributions

This thesis proposed a four-component framework for coordinating two-handed motions for robots. Chapter 2 presented a comparison of the concepts behind the proposed framework and existing schemes. I advocated for particular design choices to allow for reduced computational cost, the ability to optimize the timing of the coordinated movement, and compensating for uncertainties. Throughout this work, this framework has been developed and implemented. Ping-pong serving tasks were used as experiments to prove that the proposed approaches were viable in practice. The experimental results were generally positive and are invoked in this section to support claims made about the contributions of this work. The following section presents a summary of the experiments and the results.

This framework combined two one-armed trajectories to perform a two-armed task. This is more computationally efficient than planning two-armed trajectories, even when good seeds are provided to machine learning algorithms using methods such as, e.g., teach by demonstration. A key requirement to using this approach is the ability to decide which one-armed trajectories should be combined to achieve the two-armed task. The framework’s ability to predict the relative success
of different combinations of ball-throwing and paddle-swinging trajectories for the serving task was demonstrated.

The framework leveraged relative timing of the two one-armed motions to achieve optimal two-handed task performance. This contrasts with the approach taken in prioritized planning, which manipulates relative timing to avoid collisions. The implementation achieved strong success in manipulating the timing of the two motions. It was ensured that no trajectories that would collide were used together via novel plugins written for robotic simulation software. The framework’s ability to optimize timing was demonstrated by comparing performance using the algorithm-recommended times against other times in their neighbourhoods.

Uncertainty in robot execution, the environment, and the tools being used pose a significant challenge in coordinating two robot arms. For this reason, each single-armed robot trajectory was paired with a probabilistic representation of its taskspace outcome. Considering the final effect on the taskspace of each arm removes the necessity for modelling particular sources of uncertainty. This is in contrast to, for example, fuzzy-logic force/position controllers developed for a particular task, wherein the controllers are developed for a particular task and compensate for particular errors. The framework’s ability to compensate for uncertainty was demonstrated through an analysis that showed the successes in the ping-pong serving task were achieved despite large variances in the data used for decision making.

The implementation includes an open-source software suite for capturing taskspace outcomes using a stereovision camera, modelling taskspace outcomes probabilistically, predicting the relative success of different combinations of one-armed trajectories, optimizing relative start times of one-armed trajectories, visualizing multi-
dimensional taskspace outcomes, and simultaneously controlling two Barrett WAM robots. While some of the lower-level software is task- and robot-specific, the probabilistic modelling, success-prediction, and timing generator are designed to be task-agnostic.

Overall, the contributions of this thesis were a novel conceptual framework that allows for circumventing the difficulties in planning two-armed trajectories and compensating for uncertainties in robot performance, tool dynamics, and environment. An open-source software suite allowing others to utilize the framework was developed. The software suite and the framework were experimentally tested.

6.2 Summary of results

The framework’s performance was tested using a ping-pong serving task. In particular, tests were carried out to assess two important components of combining single-armed trajectories: predicting the relative performance of different one-armed trajectory combinations and the algorithm’s ability to generate relative timing offsets in the start times of the one armed trajectories.

To test the framework’s ability to combine one-arm trajectories, the algorithm-recommended timing offsets were compared to the optimal time offsets. It was found that the difference between the optimal and algorithm-recommended times were approximately equal to the time between frames of the 30Hz camera. In other words, the optimization was accurate to the limitation of the ball-tracking sensor.

The algorithm’s ability to predict the relative success of different combinations of one-armed trajectories was measured. It correctly predicted the relative success of four out of five combinations (with the fifth combination within uncertainty of
6.3 Future work

These results were achieved despite significant noise in the data recorded by the camera. The recorded positions of the ball and paddle varied from trial to trial by significant amounts (standard deviations greater than 1 m and 0.16 m respectively). The algorithm’s performance in spite of such noisy data shows the framework’s ability to leverage probabilistic models to achieve good performance despite noise in the data collected.

It should be noted that the tasks were carried out without computing the physics of the interaction. This offers support to the idea that the algorithm would be compatible with other two-armed tasks.

Finally, using this controller architecture approach, the first-ever ITTF-legal ping-pong serve was demonstrated by a robot.\textsuperscript{50} For the serving demonstration, the ball was thrown in the air by one robot, contacted the paddle held by the other robot during its descent, bounced once on the server’s side, cleared the net, and bounced on the opposing side of the regulation-size table. The ability to perform such a spatio-temporally sensitive task is indicative of the framework’s ability to address other challenging two-handed coordination tasks.

6.3 Future work

One of the limitations to the framework is that it requires a human to specify the desired relationship between the taskspace states of each hand. Having to find and supply this information limits the applications of the framework to tasks where the taskspace relationship has a known, analytic form. There are many tasks that do

\textsuperscript{50}To the author’s knowledge, this is the first legal serve. Robots have served ping-pong balls using workarounds as described in Section 1.3.
meet this requirement – for example, hammering a nail, using a fork and knife, and buttoning a shirt. However, this requirement could make it difficult for an end-user to teach the robot to perform different types of tasks. It would be desirable to automate this process.

As discussed in Chapter 2, Ureche and Billard [37] [38] as well as Asfour et al. [36] have studied automated methods to extract relationships between each arm’s movement in two-armed tasks. In future work, methods for determining the relationship between single-arm taskspaces could be incorporated with the framework. This could allow applications of the framework to tasks in which the relationship is not easy to determine. Additionally, it could allow for non-technical end-users to teach new abilities to a two-armed robot system. This would allow for applications of the framework to many new tasks and new robots.

### 6.4 Ping-pong serving challenge

In my closing remarks, I would like to encourage other roboticists to engage in the robot ping-pong serve challenge, either through improvements on the work presented here, or entirely novel frameworks.

Much progress has been made by researchers answering John Billingsley’s 1984 call [15] to create hardware and algorithms to allow a robot to return a ping-pong ball. I believe that through the serving challenge, we can achieve great results and further the goal of having robots help humans in both functional and fun endeavours.
Bibliography


Bibliography


[38] A.-L. Pais Ureche and A. Billard, “Learning Bimanual Coordinated Tasks From Human Demonstrations,” in ACM/IEEE International Conference on
Bibliography


Bibliography


