Robotics and Autonomous Systems 105 (2018) 121–137

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Online optimal trajectory generation for robot table tennis

Okan Koç [a,*], Guilherme Maeda [c], Jan Peters [a,b]

[a] Max Planck Institute for Intelligent Systems, Max-Planck-Ring 4, 72076 Tübingen, Germany
[b] Technische Universität Darmstadt, FG Intelligente Autonome Systeme Hochschulstr. 10, 64289 Darmstadt, Germany
[c] ATR Computational Neuroscience Labs, 2-2-2 Hikaridai Seika-sho, Soraku-gun, Kyoto 619-0288, Japan

HIGHLIGHTS

- An optimal control framework is introduced in robot table tennis to generate strikes.
- Inverse kinematics or a fixed plane to compute joint trajectories are not needed.
- Two optimization approaches are presented that encode different styles of playing.
- The parameters of the ball prediction models are estimated from demonstrations.
- Extensive experiments are shown in simulation and on our robot table tennis platform.

ARTICLE INFO

Article history:
Received 2 September 2017
Received in revised form 19 March 2018
Accepted 30 March 2018
Available online 9 April 2018

Keywords:
Optimal control
Motion planning
Optimization
Robot table tennis

ABSTRACT

In highly dynamic tasks that involve moving targets, planning is necessary to figure out when, where and how to intercept the target. In robotic table tennis in particular, motion planning can be very challenging due to time constraints, dimension of the search space and joint limits. Conventional planning algorithms often rely on a fixed virtual hitting plane to construct robot striking trajectories. These algorithms, however, generate restrictive strokes and can result in unnatural strategies when compared with human playing. In this paper, we introduce a new trajectory generation framework for robotic table tennis that does not involve a fixed hitting plane. A free-time optimal control approach is used to derive two different trajectory optimizers. The resulting two algorithms, Focused Player and Defensive Player, encode two different play-styles. We evaluate their performance in simulation and in our robot table tennis platform with a high speed cable-driven seven DOF robot arm. The algorithms return the balls with a higher probability to the opponent's court when compared with a virtual hitting plane based method. Moreover, both can be run online and the trajectories can be corrected with new ball observations.

1. Introduction

Table tennis is a challenging game for humans to master. For robots, it also serves as a testbed to study and validate the effectiveness of different movement generation algorithms. Combining different estimation, movement generation and execution schemes and studying how close they come to imitating expert human behavior will yield important insights for robotics research.

Optimality plays an important role in the search for efficient and feasible striking trajectories. However, so far most of the research in robotic table tennis were based on specialized systems, such as Cartesian coordinate robots [1,2], that eliminate great part of the difficulties in trajectory generation. Furthermore, most algorithms for robotic table tennis focused on simplifications of the game that reduced the dimensions of the search space [3] in order to quickly come up with a movement plan. In this paper, we show the advantages of incorporating optimality in trajectory generation to create more flexible movement.

Our robotic setup with an anthropomorphic seven degree of freedom Barrett WAM arm is shown in Fig. 1. The redundant arm can achieve high speeds and accelerations. It is a good platform to study different movement generation schemes. Optimal control based approaches have the potential to make use of all degrees of freedom in planning, contributing to more natural and efficient generation of strikes. The contributions of this paper are as follows: we introduce an optimal control framework in robot table tennis where the generation of striking trajectories is the result of an optimization problem. As opposed to previous works, inverse kinematics or a fixed plane to compute joint trajectories are not needed. Two different optimization approaches are presented that encode defensive and goal-oriented styles of playing. We show extensive experiments in simulation and on our table tennis platform, where

* Corresponding author.
E-mail addresses: okan.koc@tuebingen.mpg.de (O. Koç), g.maeda@atr.jp (G. Maeda), jan.peters@tuebingen.mpg.de (J. Peters).

https://doi.org/10.1016/j.robot.2018.03.012

Fig. 1. Robotic table tennis setup with four cameras on the corners of the ceiling tracking the ball at 60 Hz. We present two optimal control based trajectory generation algorithms that encode defensive and goal-oriented styles of playing. A constrained nonlinear optimization problem is solved in both cases to find an optimal striking trajectory as well as an optimal striking time.

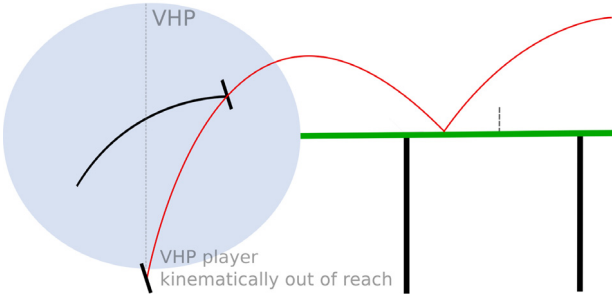


Fig. 2. Fixing a virtual hitting plane (VHP) can make the generated trajectories unnecessarily restrictive and the resulting inverse kinematics may be infeasible. Instead the whole ball trajectory should be considered in a trajectory generation framework and the hitting time as well as the hitting point should be optimized. The predicted ball trajectory and a feasible racket trajectory are shown in red and black, respectively. VHP is shown as a dotted gray line, the workspace of the robot is shown as an ellipsoidal light blue region. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

we evaluate and compare the performance of the algorithms. We do not rely on pure physical modeling to compute desired ball and racket parameters. Instead, the parameters of the prediction models are estimated based on offline human ball–racket demonstrations and the angular velocity (spin) of the ball is estimated online from actual ball data.

In the remainder of this paper, the framework is described in detail. A brief survey of robot table tennis research is given and related work on trajectory generation is introduced in Section 2. Robot trajectory generation for table tennis is formalized as an optimal control problem in Section 3. Two efficient solvers are presented in Sections 4 and 5 for optimizing the cost functional under additional constraints. The performance of the two resulting players are evaluated in Section 6 and it is shown that they compare favorably with an inverse kinematics based approach in simulation. Finally, real robot experiments are performed, where the algorithms run online in the table tennis setup. Based on this evaluation, conclusions are given with several promising extensions which might be necessary to increase performance further. Parts of this paper appeared in [4], where one of the algorithms (*Focused Player*) was proposed and evaluated in simulation. A lookup table based approach was suggested to implement it online in the table tennis platform.

2. Related work

Robot table tennis started as a challenge [5] and Anderson was the first in 1988 to construct a table tennis playing robot [6]. Dexterous motion displayed by expert table tennis players as well as the challenges in accurate ball state prediction piqued the curiosity of robot researchers. Since 1988, interest in robot table tennis has continued with various robotic platforms, for example, [7] and [8]. Earlier Cartesian coordinate robots ([1], [2], among others) were followed by industrial arms and humanoid robots with a seven degrees of freedom arm (e.g., [9], [10], [11]). Different control techniques for humanoid table tennis robots were proposed in [12] and [11]. A comprehensive categorization and summary of robot table tennis research was given in [13].

Research in robot table tennis considered ball estimation and prediction algorithms as well. Physical flight models without spin were considered in [1], [3]. Flight, rebound and racket contact models incorporating spin effects were proposed, for example, in [14] and in [15]. Frameworks estimating spin from cameras include [16] and [17]. Recently, a framework for estimating the spin of the table tennis ball using offline clustering and an online Expectation–Maximization based state estimation algorithm was introduced in [18]. The authors argue that the change of spin is very slow and they assume spin to be constant.

One of the most popular frameworks for trajectory generation in table tennis is the Virtual Hitting Plane (VHP) method, which is based on the virtual hitting point hypothesis [19]. In this approach, the trajectory of an incoming ball is first estimated from a stream of ball position observations. Usually, a physical flight model is then used to predict the intersection point of the future ball trajectory with an appropriately chosen plane. This procedure determines the striking time as well as the striking point. The remaining task-space parameters, the desired racket velocity and normal at striking time, are determined by running the physical flight model backwards from a desired ball landing position and velocity, and inverting the ball–racket contact model. For a more general discussion, see [1] and [3]. A clear limitation of the method is shown in Fig. 2. A player fixing the VHP may not generate feasible trajectories for some ball trajectories. By means of trajectory optimization, trajectories can be generated that are not constrained to a hitting plane.

Another framework that uses a mixture of movement primitives and reinforcement learning (RL) is given in [20]. Initialized with a set of dynamic movement primitives (DMP) extracted from demonstrations, RL is applied to select and generalize between the teach-in movements. A problem with this approach is that not all robots can be trained well this way. For example, the shoulder of the robot shown in Fig. 1 weighs 10 kg alone, and the wrist weighs about 2.5 kg. It is more difficult to move the links with heavy inertia, whereas it is easier to find optimization algorithms that make use of them. A different approach in [21] uses RL to learn robot movements as a response to predicted ball trajectories. The learned ball trajectories are second order polynomials, restricting the validity of the proposed approach to the front side of the robot workspace.

A related dynamic framework involving moving targets is the ball catching robot of Bäuml et al. [22] where a computationally demanding optimization problem is solved online. It includes also the catching time as another parameter to be optimized. The framework of Kim et al. [23] considers generating catching movements for more general objects. Another application of optimal control showing the benefits of spatio-temporal optimization is given in [24] on a brachiating robot. The computed solutions require lower torques when compared with traditional optimal control approaches fixing the time interval.

3. Problem statement

Most of the algorithms for robotic table tennis need to specify when, where and how to intercept the incoming ball trajectory

$\mathbf{b}(t)$. In [1] and [3] for example, the authors calculate the intersection point of a predicted ball trajectory $\mathbf{b}(t)$ with a virtual hitting plane (VHP) at $y = y_{\text{VHP}}$ to determine the space and time coordinates of the hitting event. Although additional constraints like the VHP can simplify trajectory generation, they can also lead to awkward or infeasible movements. It is possible to eliminate this plane altogether and include the striking time as another parameter to be determined in an optimization problem.

When generating striking trajectories for a robot with n degrees of freedom, trajectories with minimal acceleration can be preferred for safety and efficiency reasons. Consider the following *free-time* optimal control problem [25]

$$\min_{\ddot{\mathbf{q}},T} \int_0^T \ddot{\mathbf{q}}(t)^{\mathrm{T}} \ddot{\mathbf{q}}(t)\, \mathrm{d}t \tag{1}$$

$$\text{s.t. } \Psi_{\text{hit}}\big(\mathbf{q}(T), T\big) \in \mathcal{H}, \tag{2}$$

$$\Psi_{\text{net}}\big(\mathbf{q}(T), \dot{\mathbf{q}}(T), T\big) \in \mathcal{N}, \tag{3}$$

$$\Psi_{\text{land}}\big(\mathbf{q}(T), \dot{\mathbf{q}}(T), T\big) \in \mathcal{L}, \tag{4}$$

$$\mathbf{q}(0) = \mathbf{q}_0, \tag{5}$$

$$\dot{\mathbf{q}}(0) = \dot{\mathbf{q}}_0, \tag{6}$$

where the final hitting time T is an additional variable to be optimized along with the joint accelerations $\ddot{\mathbf{q}}(t) : [0, T] \to \mathbb{R}^n$. Initial conditions for the robot are the joint positions $\mathbf{q}_0$ and joint velocities $\dot{\mathbf{q}}_0$. The inequality constraints (2)–(4) ensure that the task requirements for table tennis are satisfied. The hitting constraint $\Psi_{\text{hit}} \in \mathcal{H}$ ensures impact of the racket with the ball at striking time T. The net constraint $\Psi_{\text{net}} \in \mathcal{N}$ makes sure the ball passes over the net and finally, the landing constraint $\Psi_{\text{land}} \in \mathcal{L}$ captures the requirement that the ball should bounce first on the opponents court. See Fig. 3 for an illustration. The precise definitions of these constraint functions and the constraint sets will be introduced in Section 5.

Solutions of (1)–(6) can be found using Pontryagin's minimum principle [26]. The optimal $\mathbf{q}(t)$ is a third degree polynomial for each degree of freedom, with the inequality constraints (2)–(4) imposing generalized *transversality conditions* on the Hamiltonian and the momenta to satisfy at striking time [27,28]. Solving such boundary value problems is hard, especially given real time constraints. In the later sections we will introduce two algorithms that will solve this problem efficiently under additional constraints. These two approaches can be seen as different ways to solve the underlying table tennis task efficiently and they lead to two different play-styles.

When given only constraints at the boundary, the striking time T, the joint position and velocity values at striking time $\mathbf{q}_f$ and $\dot{\mathbf{q}}_f$ fully parameterize this problem. The polynomial coefficients for the striking trajectory

$$\mathbf{q}_{\text{strike}}(t) = \mathbf{a}_3 t^3 + \mathbf{a}_2 t^2 + \dot{\mathbf{q}}_0 t + \mathbf{q}_0, \tag{7}$$

can then be determined in joint-space for each degree of freedom of the robot

$$\begin{aligned} \mathbf{a}_3 &= \frac{2}{T^3}(\mathbf{q}_0 - \mathbf{q}_f) + \frac{1}{T^2}(\dot{\mathbf{q}}_0 + \dot{\mathbf{q}}_f), \\ \mathbf{a}_2 &= \frac{3}{T^2}(\mathbf{q}_f - \mathbf{q}_0) - \frac{1}{T}(\dot{\mathbf{q}}_f + 2\dot{\mathbf{q}}_0). \end{aligned} \tag{8}$$

The notation that is used frequently in the rest of the paper is shown in Table 1 for the reader's convenience.

3.1. Background on ball prediction

For the trajectory generation process, three ball models will be used to determine the table tennis task constraints (2)–(4): ball flight model, ball–table rebound model and ball–racket contact model. Whenever an incoming ball is detected in midair, a flight model will first be used to predict the trajectory $\mathbf{b}(t)$ of the ball center of mass coordinates $\mathbf{b} = (b_x, b_y, b_z)^{\mathrm{T}}$ until impact with a racket, table or ground.

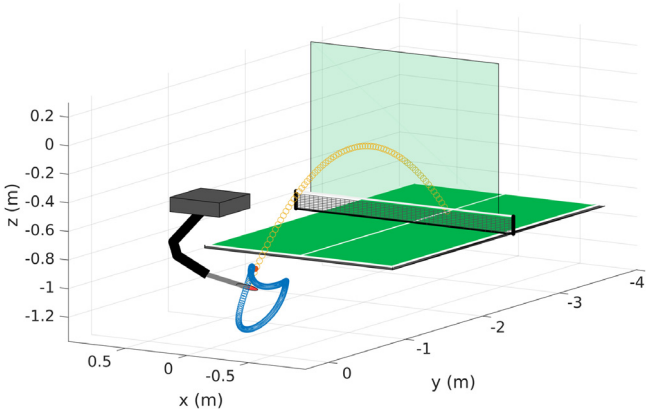


Fig. 3. In table tennis, robot trajectories (blue) can be seen as reactions to predicted ball trajectories. The players are free to decide where, when and how to intercept the ball. However, the resulting outgoing ball trajectories (orange) need to be *feasible*: the ball has to pass above the net and land on the opponent's court. The feasible region above the net is drawn in transparent green. The rules of the game can be captured as constraints for generating robot striking trajectories. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Table of symbols.

Notation	Explanation
T	Hitting time
T_{rest}	Return time
T_{land}	Desired ball landing time after hit
$\mathbf{b}_{\text{goal}}$	Desired ball landing positions
$\mathbf{q}(t)$	Joint trajectory
$\mathbf{b}(t)$	Predicted ball trajectory
$\mathbf{r}(t)$	Racket center position
$\mathbf{v}(t)$	Racket velocity
$\mathbf{n}(t)$	Racket normal
$\mathbf{K}_p$	Kinematics function for racket center position
$\mathbf{K}_n$	Kinematics function for racket normal
$\mathbf{J}(\mathbf{q}_f)$	Jacobian at hitting time
$\mathbf{n}_{\text{des}}(T)$	Desired racket normal at hitting time
$\mathbf{v}_{\text{des}}(T)$	Desired racket velocity at hitting time
$\boldsymbol{\omega}$	Ball spin
$\dot{\mathbf{b}}_{\text{in}}, \dot{\mathbf{b}}_{\text{out}}$	Ball velocity before and after impact
N	Minimum number of balls to start prediction
$\mathbf{q}_0, \dot{\mathbf{q}}_0$	Initial joint positions and velocities
$\mathbf{q}_{\text{cur}}, \dot{\mathbf{q}}_{\text{cur}}$	Joint position and velocity estimates
$\mathbf{q}_f, \dot{\mathbf{q}}_f$	Joint position and velocity at hitting time
$\mathbf{q}_{\text{ext}}$	Joint extreme values of trajectory
$\mathbf{q}_{\max}, \mathbf{q}_{\min}$	Joint angle upper and lower limits
$\mathbf{R}$	Weighting matrix
$\mathbf{q}_{\text{strike}}(t)$	Joint striking trajectory
$\mathbf{q}_{\text{return}}(t)$	Joint returning trajectory
$\Psi_{\text{hit}}, \Psi_{\text{land}}, \Psi_{\text{net}}$	Table tennis task constraints

Flight model. Table tennis balls are very light, a standard ball weighs about 2.7 g, which makes nonlinear effects due to air drag and spin noticeable especially when the ball speed $v = \|\dot{\mathbf{b}}\|_2$ is high. The *flight model* [14]

$$\ddot{\mathbf{b}} = \mathbf{g} - C_D v\, \dot{\mathbf{b}} + C_L \boldsymbol{\omega} \times \dot{\mathbf{b}}, \tag{9}$$

is a nonlinear dynamics model that incorporates air drag and spin effects. The air drag constant C_D and the lift constant C_L as well as gravity g, $\mathbf{g} = (0, 0, g)^{\mathrm{T}}$, parameterize this model. The *magnus effect* due to spin (angular velocity) $\boldsymbol{\omega}$, for example, acts as an additional downward force for an incoming ball if the angular

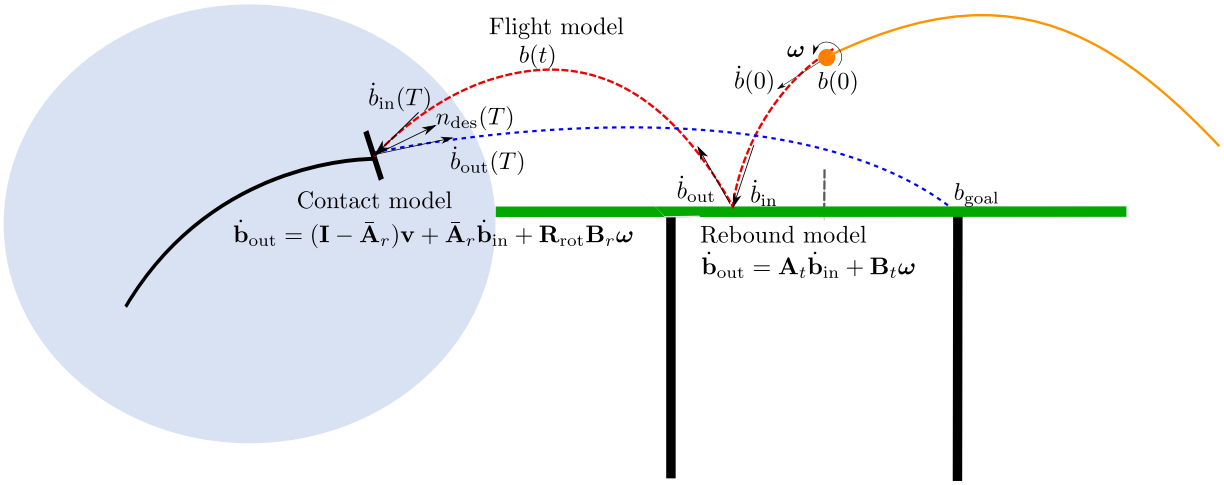


Fig. 4. Ball prediction schema for table tennis. After estimating the initial ball position, velocity and spin, the future path of the ball can be predicted using the flight model, the rebound model and the racket–ball contact model. The trajectory generation framework uses these models to compute desired striking trajectories.

velocities are in the negative x-direction (topspin). It is assumed that spin stays constant throughout the ball motion.

Rebound model. Formally, rebound is a discrete event which reflects the ball velocity when the ball hits the table. The incoming velocities $\dot{\mathbf{b}}_{\text{in}}$ at bouncing time are transformed to outgoing velocities $\dot{\mathbf{b}}_{\text{out}}$. The following nonlinear *rebound model* [14] for a standard ball with radius $r_B = 2$ cm,

$$\dot{\mathbf{b}}_{\text{out}} = \mathbf{A}_t\dot{\mathbf{b}}_{\text{in}} + \mathbf{B}_t\boldsymbol{\omega}, \tag{10}$$

is parameterized by the dynamic coefficient of friction μ_t and the coefficient of restitution ϵ_t of the table

$$\mathbf{A}_t = \begin{bmatrix} 1-\alpha & 0 & 0 \\ 0 & 1-\alpha & 0 \\ 0 & 0 & -\epsilon_t \end{bmatrix},\ \mathbf{B}_t = \begin{bmatrix} 0 & \alpha r_B & 0 \\ -\alpha r_B & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{11}$$

where the nonlinearity comes from the sliding parameter

$$\alpha = \mu_t(1+\epsilon_t)\frac{\dot{b}_z}{\|\dot{\mathbf{b}}_T\|}, \tag{12}$$

$\dot{\mathbf{b}}_T$ is the *tangent velocity* at contact

$$\dot{\mathbf{b}}_T = (\dot{b}_x - r_B\omega_y, \dot{b}_y + r_B\omega_x, 0)^{\mathrm{T}}, \tag{13}$$

for $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^{\mathrm{T}}$. This model suggests, for example, that some amount of topspin is transferred at sliding impact to linear velocity in the y-direction. See Fig. 4 for a table tennis schema.

Racket contact model. We assume the following linear racket contact model holds for a standard racket with radius $r_R \approx 7.6$ cm,

$$\mathbf{o} = \mathbf{A}_r\mathbf{i} + \mathbf{B}_r\boldsymbol{\omega}, \tag{14}$$

between the outgoing ball velocity $\mathbf{o}$ and the incoming ball velocity $\mathbf{i}$, similar to (10) but in the moving racket frame. The outgoing ball Cartesian velocities are hence found by multiplying $\mathbf{o}$ with the racket rotation matrix and adding the racket velocities, i.e., $\dot{\mathbf{b}}_{\text{out}}(t) = \mathbf{R}_{\text{rot}}\mathbf{o}(t) + \mathbf{v}(t)$, where the rotation matrix $\mathbf{R}_{\text{rot}}(\mathbf{q}(t))$ of the racket is given by the kinematics function. The impact model is parameterized by the constants κ and ϵ_r

$$\mathbf{A}_r = \begin{bmatrix} 1-\kappa & 0 & 0 \\ 0 & 1-\kappa & 0 \\ 0 & 0 & -\epsilon_r \end{bmatrix},\ \mathbf{B}_r = \begin{bmatrix} 0 & \kappa r_R & 0 \\ -\kappa r_R & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{15}$$

Letting $\bar{\mathbf{A}}_r := \mathbf{R}_{\text{rot}}\mathbf{A}_r\mathbf{R}_{\text{rot}}^{\mathrm{T}}$, we get the following relationship between the Cartesian velocities:

$$\dot{\mathbf{b}}_{\text{out}} = (\mathbf{I} - \bar{\mathbf{A}}_r)\mathbf{v} + \bar{\mathbf{A}}_r\dot{\mathbf{b}}_{\text{in}} + \mathbf{R}_{\text{rot}}\mathbf{B}_r\boldsymbol{\omega}. \tag{16}$$

Ball prediction. The models (9)–(16) can be composed together to predict the future ball trajectory given camera observations. We use an Extended Kalman Filter (EKF) to estimate the ball state from observations [29]. The ball state for the filter is the ball positions and velocities, since we assume that the ball spin is constant throughout motion. The ball spin can be seen as a parameter of the prediction functions.

EKF instantiated with the models (9)–(16), the initial ball positions $\mathbf{b}_0$ and velocities $\dot{\mathbf{b}}_0$ and spin $\boldsymbol{\omega}$, estimates the evolving ball state and predicts the future ball trajectory at each time instant t: a multivariate normal distribution $p_t(\mathbf{b}, \dot{\mathbf{b}})$ of ball states parameterized by time is generated

$$\left(\mathbf{b}(t)^{\intercal}, \dot{\mathbf{b}}(t)^{\intercal}\right)^{\intercal} \sim p_t(\mathbf{b}, \dot{\mathbf{b}}) = \mathcal{N}(\boldsymbol{\mu}(t), \boldsymbol{\Sigma}(t)), \tag{17}$$

where $\boldsymbol{\mu}(t) = (\mathbf{b}(t)^{\intercal}, \dot{\mathbf{b}}(t)^{\intercal})^{\intercal}$ is the mean ball position and velocity predictions. The covariance matrix $\boldsymbol{\Sigma}(t)$ is updated along with the mean estimate $\boldsymbol{\mu}(t)$ using the EKF predict and update equations. The covariance matrices are used to reject outliers and hence make Kalman Filtering more robust to ball detection errors.

4. The focused player

A higher-level strategy in table tennis could, based on a perceived state of the opponent, command to return an incoming ball to a desired location. A reliable trajectory generation algorithm for that purpose should be flexible and easily find safe joint movements. The optimal control based approach penalizing sum of squared accelerations, in this regard, leads to a flexible optimization problem where it is easy to find good hitting postures, while satisfying additional safety constraints.

4.1. Racket constraints

For a table tennis player that wants to guarantee the return of the incoming ball to a desired location at a desired landing time, the optimal control problem introduced in (1) can be solved efficiently under additional racket constraints

$$\mathbf{K}_p(\mathbf{q}(T)) = \mathbf{b}(T), \tag{18}$$

$$\mathbf{K}_n(\mathbf{q}(T)) = \mathbf{n}_{\text{des}}(T), \tag{19}$$

$$\mathbf{J}(\mathbf{q}(T))\dot{\mathbf{q}}(T) = \mathbf{v}_{\text{des}}(T). \tag{20}$$

The racket center position $\mathbf{r}(T)$ and the racket normal $\mathbf{n}(T)$ at hitting time T are computed using the kinematics functions $\mathbf{K}_p(\cdot)$

and $\mathbf{K}_n(\cdot)$, respectively. The Jacobian $\mathbf{J}(\cdot) \in \mathbb{R}^{3\times n}$ [30] at hitting time transforms the joint velocities in (20) to racket velocities $\mathbf{v}(T)$. To maximize the probability of hitting the ball, the desired racket center is set at hitting time equal to the mean ball position estimate in (18), i.e., $\mathbf{r}(T) = \mathbf{b}(T)$. To return the ball to the opponent's court, the constraints on the racket normal $\mathbf{n}(T)$ and velocity at hitting time $\mathbf{v}(T)$ are imposed in (19) and (20), respectively.

The imposed racket constraints (18)–(20) can satisfy and hence effectively replace the table tennis task constraints (2)–(4) for suitable $\mathbf{n}_{\text{des}}(T)$, $\mathbf{v}_{\text{des}}(T)$, if the future path of the incoming ball is predicted before the optimization for calculating the hitting movement takes place.

Calculating desired racket parameters. After predicting the future ball path $\mathbf{b}(t)$ at a discrete set of time instants $t \in (0, T_{\text{pred}})$, the next step is to compute desired racket velocities $\mathbf{v}_{\text{des}}(t)$ and desired racket normals on this path $\mathbf{n}_{\text{des}}(t)$. These desired racket parameters will give the incoming ball during the impact, a desired outgoing ball velocity according to (16). They are calculated by first specifying a desired landing point $\mathbf{b}_{\text{goal}}$ and a desired duration of flight after strike T_{land}. A desired ball outgoing velocity is then found by solving the boundary value problem for the flight model (9) with the boundary values

$$\begin{aligned} \mathbf{b}_{\text{out}}(0) &= \mathbf{b}(t), \\ \mathbf{b}_{\text{out}}(T_{\text{land}}) &= \mathbf{b}_{\text{goal}}, \end{aligned} \tag{21}$$

for each t. Spin $\boldsymbol{\omega}$ is assumed to be constant throughout. Afterwards, $\mathbf{v}_{\text{des}}(t)$ and $\mathbf{n}_{\text{des}}(t)$ are calculated by inverting the racket contact model (16) given the outgoing ball velocities $\dot{\mathbf{b}}_{\text{out}}$ at impact.

In practice, (21) can be solved very fast for each t with a gradient-based optimizer. The desired outgoing ball velocities for the sequence of boundary value problems in (21) can be initialized with the previous solutions. The closed form solution of the ballistic flight model (i.e., zero drag and spin) can be used to initialize the process.

4.2. Nonlinear constrained optimization

We briefly show here that the solution $\mathbf{q}(t)$ to the optimal control problem posed in (1) under additional racket constraints (18)–(20) is a third order polynomial for each degree of freedom, $i = 1, \ldots, n$.

Derivation from minimum principle. Using the minimum principle for unconstrained inputs $\mathbf{u}(t) = \ddot{\mathbf{q}}(t) \in \mathbb{R}^n$, the Hamiltonian

$$\mathcal{H}(\mathbf{u}, \dot{\mathbf{q}}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{u}^{\mathrm{T}}\mathbf{u}(t) + \boldsymbol{\lambda}^{\mathrm{T}}\dot{\mathbf{q}} + \boldsymbol{\mu}^{\mathrm{T}}\mathbf{u} \tag{22}$$

for the momenta $[\boldsymbol{\lambda}(t), \boldsymbol{\mu}(t)] \in \mathbb{R}^{2n}$ is minimized at

$$\mathbf{u}^*(t) = -\frac{1}{2}\boldsymbol{\mu}^*(t). \tag{23}$$

Costate equation for the momenta gives

$$\begin{aligned} \dot{\boldsymbol{\lambda}}^*(t) &= \mathbf{0}, \\ \dot{\boldsymbol{\mu}}^*(t) &= -\boldsymbol{\lambda}^*(t), \end{aligned} \tag{24}$$

or in other terms, $\boldsymbol{\lambda}^* = 12\mathbf{a}_3$, $\boldsymbol{\mu}^* = -2(6\mathbf{a}_3 t + 2\mathbf{a}_2)$, for some constant vectors $\mathbf{a}_3, \mathbf{a}_2 \in \mathbb{R}^n$. Plugging it into (23) we get

$$\ddot{\mathbf{q}}^*(t) = 6\mathbf{a}_3 t + 2\mathbf{a}_2, \tag{25}$$

which shows that the optimal accelerations are linear functions of time. The joint positions $\mathbf{q}(t)$ are then third order polynomials as in (7) with $2n$ coefficients to be determined using $\mathbf{n}_{\text{des}}(T)$, $\mathbf{b}(T)$, $\mathbf{v}_{\text{des}}(T)$ and free final time T as another variable. The transversality condition resulting from the boundary constraints $\boldsymbol{\Psi}(\mathbf{q}(T), \dot{\mathbf{q}}(T), T) = \mathbf{0}$ can be written as

$$\begin{bmatrix} -\mathcal{H}(T) \\ \boldsymbol{\lambda}(T) \\ \boldsymbol{\mu}(T) \end{bmatrix} = D\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\nu} = \begin{bmatrix} D_T\boldsymbol{\Psi} & D_{\mathbf{q}}\boldsymbol{\Psi} & D_{\dot{\mathbf{q}}}\boldsymbol{\Psi} \end{bmatrix}^{\mathrm{T}}\boldsymbol{\nu}, \tag{26}$$

$$\boldsymbol{\Psi} = \begin{bmatrix} \mathbf{K}_p(\mathbf{q}(T)) - \mathbf{b}(T) \\ \mathbf{K}_n(\mathbf{q}(T)) - \mathbf{n}_{\text{des}}(T) \\ \mathbf{J}(\mathbf{q}(T))\dot{\mathbf{q}}(T) - \mathbf{v}_{\text{des}}(T) \end{bmatrix}, \tag{27}$$

for some $\boldsymbol{\nu} \in \mathbb{R}^9$. The necessary condition (26) supplies the additional $2n - 8$ equations to determine all the variables. A nonlinear equation solver can be used for this purpose. Alternatively, the first order optimality conditions of the augmented cost function

$$\bar{J}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T, \boldsymbol{\nu}) = \boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\nu} + J(\mathbf{q}_f, \dot{\mathbf{q}}_f, T), \tag{28}$$

$$\begin{aligned} J(\mathbf{q}_f, \dot{\mathbf{q}}_f, T) &= \int_0^T \ddot{\mathbf{q}}^*(t)^{\mathrm{T}}\ddot{\mathbf{q}}^*(t)\,\mathrm{d}t, \\ &= \int_0^T (6\mathbf{a}_3 t + 2\mathbf{a}_2)^{\mathrm{T}}(6\mathbf{a}_3 t + 2\mathbf{a}_2)\,\mathrm{d}t, \\ &= 3T^3\mathbf{a}_3^{\mathrm{T}}\mathbf{a}_3 + 3T^2\mathbf{a}_3^{\mathrm{T}}\mathbf{a}_2 + T\mathbf{a}_2^{\mathrm{T}}\mathbf{a}_2, \end{aligned} \tag{29}$$

directly satisfy (26) and the boundary equality constraints.

Parameter optimization. The optimal trajectories are third order polynomials in joint-space for each degree of freedom of the robot, where the coefficients of the polynomials can be parameterized in terms of final joint positions $\mathbf{q}_f$, final joint velocities $\dot{\mathbf{q}}_f$ and hitting time T. That is, along with the hitting time T as a free parameter, the optimization problem is $2n + 1$ dimensional with nonlinear equality constraints. The integrand in (1) can be rewritten in terms of these free parameters and integrated over time as in (29) to form the following optimization problem

$$\min_{\mathbf{q}_f, \dot{\mathbf{q}}_f, T} 3T^3\mathbf{a}_3^{\mathrm{T}}\mathbf{a}_3 + 3T^2\mathbf{a}_3^{\mathrm{T}}\mathbf{a}_2 + T\mathbf{a}_2^{\mathrm{T}}\mathbf{a}_2 \tag{30}$$

$$\text{s.t. } \mathbf{K}_p(\mathbf{q}_f) = \mathbf{b}(T), \tag{31}$$

$$\mathbf{K}_n(\mathbf{q}_f) = \mathbf{n}_{\text{des}}(T), \tag{32}$$

$$\mathbf{J}(\mathbf{q}_f)\dot{\mathbf{q}}_f = \mathbf{v}_{\text{des}}(T), \tag{33}$$

$$\mathbf{q}_{\min} \le \mathbf{q}_f \le \mathbf{q}_{\max}, \tag{34}$$

$$\mathbf{q}_{\min} \le \mathbf{q}_{\text{ext}} \le \mathbf{q}_{\max}. \tag{35}$$

The returning trajectories that bring the robot from striking joint positions $\mathbf{q}_f$ to the fixed rest position $\mathbf{q}_0$ in joint space are also taken as third order polynomials

$$\mathbf{q}_{\text{return}}(t) = \tilde{\mathbf{a}}_3 t^3 + \tilde{\mathbf{a}}_2 t^2 + \dot{\mathbf{q}}_f t + \mathbf{q}_f, \tag{36}$$

for a fixed return time T_{rest}, $0 \le t \le T_{\text{rest}}$. The coefficients $\tilde{\mathbf{a}}_3$, $\tilde{\mathbf{a}}_2$ of (36) are as in (8) but with $\mathbf{q}_0$, $\mathbf{q}_f$ and $\dot{\mathbf{q}}_0$, $\dot{\mathbf{q}}_f$ reversed

$$\begin{aligned} \tilde{\mathbf{a}}_3 &= \frac{2}{T_{\text{rest}}^3}(\mathbf{q}_f - \mathbf{q}_0) + \frac{1}{T_{\text{rest}}^2}(\dot{\mathbf{q}}_f + \dot{\mathbf{q}}_0), \\ \tilde{\mathbf{a}}_2 &= \frac{3}{T_{\text{rest}}^2}(\mathbf{q}_0 - \mathbf{q}_f) - \frac{1}{T_{\text{rest}}}(\dot{\mathbf{q}}_0 + 2\dot{\mathbf{q}}_f). \end{aligned} \tag{37}$$

The optimization variables $\mathbf{q}_f$, $\dot{\mathbf{q}}_f$ fully parameterize the returning polynomials as well as the striking polynomials.

Joint limit satisfaction. Inspired by the simplicity of the Minimum Principle based solution, the same parameterization can be extended to the more realistic scenario where joint limits are included additionally as inequality constraints in the optimization. When optimizing (30) the final joint positions $\mathbf{q}_f$ are enforced in (34) to respect the joint limits for each component. However, the whole trajectory, both the striking and returning segments, needs to respect the joint limits at all times. Third order polynomials can

Algorithm 1 *Focused Player* (FP)

Require: $\mathbf{q}_0, \mathbf{b}_{\text{goal}}, T_{\text{land}}, T_{\text{pred}}, T_{\text{rest}}, N, \mathbf{R}$
1: Move to initial posture $\mathbf{q}_0$, $\dot{\mathbf{q}}_0 = \mathbf{0}$.
2: **loop**
3: Query vision sys. for new observation $\mathbf{b}_{\text{obs}}$.
4: Observe current state $\mathbf{q}_{\text{cur}}, \dot{\mathbf{q}}_{\text{cur}}$.
5: **if** N new ball observations $\mathbf{b}_{\text{obs}}$ **then**
6: Initialize EKF.
7: **end if**
8: **if** EKF is initialized **and** valid obs. $\mathbf{b}_{\text{obs}}$ **then**
9: Estimate position $\mathbf{b}$ and vel. $\dot{\mathbf{b}}$ with EKF.
10: Predict $\mathbf{b}(t)$ till horizon T_{pred}.
11: Compute $\mathbf{v}_{\text{des}}(t)$, $\mathbf{n}_{\text{des}}(t)$ using racket model and boundary values $\mathbf{b}_{\text{goal}}, T_{\text{land}}$.
12: Compute param. $\mathbf{q}_f, \dot{\mathbf{q}}_f, T$ from $\mathbf{q}_{\text{cur}}, \dot{\mathbf{q}}_{\text{cur}}$ using desired resting posture $\mathbf{q}_0$, time to return T_{rest}, weighting matrix $\mathbf{R}$, and task constraints $\mathbf{b}(t)$, $\mathbf{v}_{\text{des}}(t)$, $\mathbf{n}_{\text{des}}(t)$.
13: Update strike and return trajectories $\mathbf{q}_{\text{des}}(t) = \{\mathbf{q}_{\text{strike}}(t), \mathbf{q}_{\text{return}}(t)\}$.
14: **end if**
15: Track $\mathbf{q}_{\text{des}}(t)$ with Inv. Dyn. $\mathbf{fi} = \mathbf{f}(\mathbf{q}_{\text{des}}, \dot{\mathbf{q}}_{\text{des}}, \ddot{\mathbf{q}}_{\text{des}})$.
16: **end loop**

each have at most 2 extrema $\mathbf{q}_{\text{ext}}$ in the interior of their domains, corresponding to the conditions

$$\dot{\mathbf{q}}_{\text{strike}}(t) = 3\mathbf{a}_3 t^2 + 2\mathbf{a}_2 t + \dot{\mathbf{q}}_0 = 0, \tag{38}$$

$$\dot{\mathbf{q}}_{\text{return}}(t) = 3\tilde{\mathbf{a}}_3 t^2 + 2\tilde{\mathbf{a}}_2 t + \dot{\mathbf{q}}_f = 0. \tag{39}$$

Therefore, checking the joint extrema candidates $\mathbf{q}_{\text{ext}}$ in (35) at times

$$\upsilon_j^{1,2} = \frac{-a_{2,j} \pm \sqrt{a_{2,j}^2 - 3a_{3,j}\dot{q}_{0,j}}}{3a_{3,j}}, \qquad \upsilon_j^{3,4} = \frac{-\tilde{a}_{2,j} \pm \sqrt{\tilde{a}_{2,j}^2 - 3\tilde{a}_{3,j}\dot{q}_{f,j}}}{3\tilde{a}_{3,j}}, \tag{40}$$

for each $j = 1, \ldots, n$ makes sure that the joint limits are satisfied both for the striking trajectory (at times $\upsilon_j^{1,2}$) and for the returning trajectory (at times $\upsilon_j^{3,4}$). These candidate times are fully parameterized by the optimization variables since the coefficients $\mathbf{a}_3, \mathbf{a}_2$ (8) and $\tilde{\mathbf{a}}_3, \tilde{\mathbf{a}}_2$ (36) appearing in (40) can be determined whenever $\mathbf{q}_f, \dot{\mathbf{q}}_f, T$ are computed. The values $\boldsymbol{\upsilon}^{1,2}$ are clamped to the interval $[0,\ T_{\text{pred}}]$ and $\boldsymbol{\upsilon}^{3,4}$ to $[0,\ T_{\text{rest}}]$ if they are imaginary or outside their corresponding intervals.

Online trajectory generation. Using a constrained nonlinear optimizer, the algorithm can be run online whenever there are enough ball samples $N = 12$ available to estimate the incoming ball state and spin reliably. After computing an initial striking trajectory and starting to move, the trajectories can be corrected online whenever new ball samples are available.

The full trajectory generation framework and the resulting table tennis player *Focused Player* (FP) is summarized in Algorithm 1. After bringing the robot to a desired initial posture $\mathbf{q}_0$, the vision system is queried (line 3) for new reliable ball observations. The Extended Kalman Filter (EKF) is initialized (line 6) using the first $N = 12$ ball positions. EKF then updates the ball state whenever new ball observations $\mathbf{b}_{\text{obs}}$ are available. The ball state is used to predict every $dt = 2$ ms, a discrete set of ball positions and velocities along the future ball path $\mathbf{b}(t)$, up to a horizon of $T_{\text{pred}} = 1.0$ s. Desired racket parameters are then computed (line 11) for each $t = dt, \ldots, T_{\text{pred}}$, before the optimization for the robot joint movements is launched. With an optimized implementation, the optimization (line 12) takes on average 25 ms to find the trajectory parameters for the Barrett WAM. The optimized implementation thus makes it possible to implement the approach online in the robot table tennis setup. See Section 6 for the implementation details.

The desired landing position $\mathbf{b}_{\text{goal}}$ and T_{land} are important free parameters of the algorithm, that can possibly be set by a higher-level strategy. For instance, a fast playing robot would prefer to set T_{land} rather low, and given an opponent state, a robot that wants to score a point could profit from adapting the desired landing position as well. We discuss in the Experiment section the effects of changing these parameters for the overall returning accuracy of the Barrett WAM.

The optimization takes place online (line 12) whenever new reliable ball observations and robot joint sensor recordings $\mathbf{q}_{\text{cur}}$ are available. The desired robot movement can be updated to accommodate for modeling and control errors. Feasible striking and return trajectories are then formed or updated (line 13), which are executed with an existing inverse dynamics controller. In actual table tennis experiments, we apply high gain PD-control in addition to inverse dynamics (computed torque). See Section 6 for more details of how the algorithm runs online in actual robot table tennis experiments.

For simplicity we have not introduced a weighting matrix in (30). We include in Algorithm 1 an arbitrary positive definite weighting matrix $\mathbf{R}$, which can be used to emphasize for each degree of freedom the difficulty of accelerating that particular joint.

5. The defensive player

The optimal control problem introduced in (1) can be solved directly without the additional Cartesian constraints considered in the previous section. As opposed to fixing a desired landing point and a desired landing time to satisfy the requirements of a higher-level strategy, there can be times during table tennis where it is much more important to safely return the ball. A *defensive* table tennis player could relax the previously imposed racket constraints (18)–(20) by requiring only that the task constraints (2)–(4) are satisfied. See Fig. 5 for an illustration.

5.1. Table tennis task constraints

The indoor environment that is modeled contains a standard ping pong table with coordinates

$$\mathcal{T} = \{(x, y, z_T) \in \mathbb{R}^3 |\ -\tfrac{w_T}{2} \le x \le \tfrac{w_T}{2}, \quad y_{\text{edge}} - l_T \le y \le y_{\text{edge}}\}, \tag{41}$$

where the origin is placed at the robot base. The table with width $w_T = 152$ cm and length $l_T = 276$ cm is approximately at $z_T = -0.89$ cm height and placed $|y_{\text{edge}}| = 115$ cm away from the robot base, see Fig. 1. The racket and the table tennis ball have a radius of $r_R \approx 7.6$ cm and $r_B = 2$ cm, respectively. The condition for successful landing can be put succinctly as follows: the ball after the hit has to pass over the net, below the wall and land on the opponents court. See Fig. 3 for an illustration.

Hitting constraint. All possible impacts of the racket with the ball at time T are captured by the *hitting set* $\mathcal{H}$

$$\mathcal{H} = \{(T,\ \mathbf{r}(T),\ \mathbf{n}(T)) \in \mathbb{R}^7 \mid T \ge 0, \quad 0 \le \mathbf{n}(T)^{\mathrm{T}}(\mathbf{b}(T) - \mathbf{r}(T)) \le r_B, \quad \|\mathbf{P}_n^{\perp}(T)(\mathbf{b}(T) - \mathbf{r}(T))\| \le r_R\}, \tag{42}$$

where $\mathbf{P}_n^{\perp}(t) = \mathbf{I} - \mathbf{n}(t)\mathbf{n}^{\mathrm{T}}(t)$ is the projection matrix onto the racket plane. The hitting function

$$\boldsymbol{\Psi}_{\text{hit}}\big(\mathbf{q}(T), T\big) = \begin{pmatrix} T \\ \mathbf{K}_p(\mathbf{q}(T)) \\ \mathbf{K}_n(\mathbf{q}(T)) \end{pmatrix} \tag{43}$$

enforces the kinematic constraints for hitting when $\boldsymbol{\Psi}_{\text{hit}}\big(\mathbf{q}(T), T\big) \in \mathcal{H}$.

Net constraint. When crossing the net at time t, the ball should be above the net height z_{net} and below the wall z_{wall}, that is, $(t,\ \mathbf{b}(t))$ should belong to the set

$$\begin{aligned} \mathcal{N} = \{(T_{\text{net}},\ \mathbf{b}(T_{\text{net}})) \in\ & \mathbb{R}^4 \mid T_{\text{net}} > 0, \\ & b_y(T_{\text{net}}) = y_{\text{net}} \coloneqq y_{\text{edge}} - \tfrac{l_T}{2}, \\ & z_{\text{net}} \le\ b_z(T_{\text{net}}) \le z_{\text{wall}}\}. \end{aligned} \tag{44}$$

The net hitting time T_{net} is calculated by using the ball prediction functions,

$$T_{\text{net}}(\mathbf{q}(T), \dot{\mathbf{q}}(T), T) = \{t \mid b_y(t) = y_{\text{net}}\}. \tag{45}$$

The net function $\boldsymbol{\Psi}_{\text{net}}$ that predicts the future ball position on the vertical net plane

$$\boldsymbol{\Psi}_{\text{net}}\big(\mathbf{q}(T), \dot{\mathbf{q}}(T), T\big) = \begin{pmatrix} T_{\text{net}} \\ b_x(T_{\text{net}}) \\ y_{\text{net}} \\ b_z(T_{\text{net}}) \end{pmatrix} \tag{46}$$

is then the composition of a ball–racket contact model with the ball flight model.

Landing constraint. The desired condition for landing afterwards in the opponents court will then be

$$\begin{aligned} \mathcal{L} = \{(T_{\text{land}}, \mathbf{b}(T_{\text{land}})) \in \mathbb{R}^4 \mid & T_{\text{land}} > T_{\text{net}}, \\ & b_z(T_{\text{land}}) = z_T + r_B, \\ & -\tfrac{w_T}{2} \le b_x(T_{\text{land}}) \le \tfrac{w_T}{2}, \\ & y_{\text{net}} - \tfrac{l_T}{2} \le\ b_y(T_{\text{land}}) \le y_{\text{net}}\}. \end{aligned} \tag{47}$$

The landing time T_{land} at which the ball hits the horizontal table plane, is found using the ball prediction functions

$$T_{\text{land}}(\mathbf{q}(T), \dot{\mathbf{q}}(T), T) = \{t > T_{\text{net}} \mid b_z(t) = z_T + r_B\}. \tag{48}$$

The landing function $\boldsymbol{\Psi}_{\text{land}}$ that predicts the future ball position on the horizontal table plane,

$$\boldsymbol{\Psi}_{\text{land}}\big(\mathbf{q}(T), \dot{\mathbf{q}}(T), T\big) = \begin{pmatrix} T_{\text{land}} \\ b_x(T_{\text{land}}) \\ b_y(T_{\text{land}}) \\ z_T + r_B \end{pmatrix}, \tag{49}$$

is, as before, the composition of a ball–racket contact model with the ball flight model.

5.2. Nonlinear constrained optimization

We briefly show here that the solution $\mathbf{q}(t)$ to the original optimal control problem posed in (1)–(6), with additional penalties for landing and hitting, is a third order polynomial for each degree of freedom, $i = 1, \ldots, n$. The penalties for landing and hitting can be grouped together as ϕ_{pen}, where

$$\begin{aligned} \phi_{\text{pen}} &= \alpha_{\text{hit}}\phi_{\text{hit}}(\mathbf{q}_f, T) + \alpha_{\text{land}}\phi_{\text{land}}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T), \\ \phi_{\text{hit}} &= (\mathbf{b}(T) - \mathbf{r}(T))^{\mathrm{T}}\mathbf{P}_n^{\perp}(T)(\mathbf{b}(T) - \mathbf{r}(T)), \\ \phi_{\text{land}} &= (\mathbf{b}(T_{\text{land}}) - \mathbf{b}_{\text{goal}})^{\mathrm{T}}(\mathbf{b}(T_{\text{land}}) - \mathbf{b}_{\text{goal}}). \end{aligned} \tag{50}$$

with tunable weights α_{hit} and α_{land}.

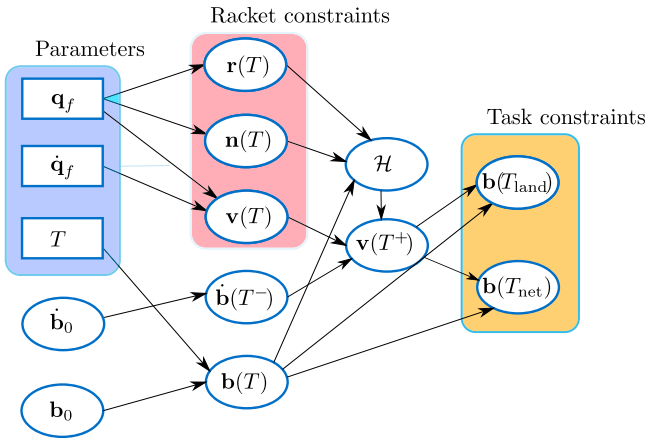


Fig. 5. Graphical representation of table tennis interactions. The hybrid system for the table tennis ball is described by the flight dynamics, governed by a set of differential equations, as well as a discrete hitting event $\mathcal{H}$ that changes the ball velocity from $\dot{\mathbf{b}}(T^-)$ to $\dot{\mathbf{b}}(T^+)$ at the hitting time T. The control variables for the reduced optimization problem are located in the light blue rectangle. Racket constraints that are enforced by *Focused Player* to land the ball to a fixed location are indicated in the red rectangle. *Defensive Player* on the other hand, directly enforces the task (landing and net) constraints, located in the orange rectangle, without additional constraints. By additionally checking for the hitting condition $\mathcal{H}$ in the optimization, this problem can be cast as a (standard) continuous optimal control problem, where the decision variables $\mathbf{q}_f, \dot{\mathbf{q}}_f$ and T continuously affect the outgoing ball velocity, the ball net and landing positions, through the repeated application of the flight model (9) and the contact model (14).

Derivation from minimum principle. The same derivation in Section 4.2 applies for the Hamiltonian and the momenta. Instead of the boundary equality constraints we get the more general inequality constraints at striking time

$$\begin{aligned} -\mathcal{H}(T) &= \frac{\partial \Phi}{\partial T}, \\ \boldsymbol{\lambda}^*(T) &= \frac{\partial \Phi}{\partial \mathbf{q}(T)}, \\ \boldsymbol{\mu}^*(T) &= \frac{\partial \Phi}{\partial \dot{\mathbf{q}}(T)}, \end{aligned} \tag{51}$$

where the generalized boundary cost is

$$\Phi(\mathbf{q}, \dot{\mathbf{q}}, T, \boldsymbol{\nu}) = \phi_{\text{pen}} + \boldsymbol{\nu}^{\mathrm{T}}\boldsymbol{\Psi}_{\text{strike}} \tag{52}$$

for some Lagrange multipliers $\boldsymbol{\nu} \in \mathbb{R}^{13}$ and $\boldsymbol{\Psi}_{\text{strike}} \le \mathbf{0}$ representing the hitting, landing and net inequality constraints (2)–(4). The conditions (51) along with primal feasibility, complementary slackness and dual feasibility conditions

$$\begin{aligned} &\boldsymbol{\Psi}_{\text{strike}}(\mathbf{q}(T), \dot{\mathbf{q}}(T), T) \le \mathbf{0}, \\ &\boldsymbol{\Psi}_{\text{strike}}^{\mathrm{T}}\boldsymbol{\nu} = 0, \\ &\boldsymbol{\nu} \ge \mathbf{0}, \end{aligned} \tag{53}$$

respectively, supply the additional equations to determine all the variables. The first order optimality conditions of the augmented cost function

$$\tilde{J}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T, \boldsymbol{\nu}) = \Phi(\mathbf{q}_f, \dot{\mathbf{q}}_f, T, \boldsymbol{\nu}) + J(\mathbf{q}_f, \dot{\mathbf{q}}_f, T), \tag{54}$$

directly satisfy (51) and the associated boundary inequality constraints.

Parameter optimization. With the same parameterization as in *Focused Player* (FP), the cost functional is extended with an additional penalty term $\phi_{\text{pen}}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T)$ while enforcing the more general inequality constraints

$$\min_{\mathbf{q}_f, \dot{\mathbf{q}}_f, T} 3T^3 \mathbf{a}_3^{\mathrm{T}} \mathbf{a}_3 + 3T^2 \mathbf{a}_3^{\mathrm{T}} \mathbf{a}_2 + T \mathbf{a}_2^{\mathrm{T}} \mathbf{a}_2 + \phi_{\text{pen}} \tag{55}$$

$$\text{s.t. } \Psi_{\text{strike}}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T) \leq \mathbf{0}, \tag{56}$$

$$\mathbf{q}_{\min} \leq \mathbf{q}_f \leq \mathbf{q}_{\max}, \tag{57}$$

$$\mathbf{q}_{\min} \leq \mathbf{q}_{\text{ext}} \leq \mathbf{q}_{\max}. \tag{58}$$

The components $\phi_{\text{land}}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T)$ and $\phi_{\text{hit}}(\mathbf{q}_f, T)$ of ϕ_{pen} impose additional penalties on the hitting joint positions and velocities. Unlike the FP, the joint extrema $\mathbf{q}_{\text{ext}}$ are only checked for the striking trajectory, as the returning trajectory is the result of an additional optimization.

Resting state optimization. For the DP, we additionally consider a resting posture optimization to find a more *defensive* posture for the robot. By finding a joint resting state $\mathbf{q}_0$ that minimizes both the distance from the hitting state $\mathbf{q}_f$ and the squared Frobenius norm of the Jacobian at the resting state

$$\min_{\mathbf{q}_0, t} (\mathbf{q}_0 - \mathbf{q}_f)^{\mathrm{T}} (\mathbf{q}_0 - \mathbf{q}_f) + \|\mathbf{J}(\mathbf{q}_0)\|_F^2 \tag{59}$$

$$\text{s.t. } 0 \leq t \leq T_{\text{pred}}, \tag{60}$$

$$\mathbf{K}_p(\mathbf{q}_0) = \mathbf{b}(t), \tag{61}$$

$$\mathbf{q}_{\min} \leq \mathbf{q}_0 \leq \mathbf{q}_{\max}, \tag{62}$$

$$\mathbf{q}_{\min} \leq \mathbf{q}_{\text{ext}} \leq \mathbf{q}_{\max}, \tag{63}$$

such that the Cartesian resting state intersects the ball path for some t, $0 \leq t \leq T_{\text{pred}}$, we can minimize the amount of movement necessary to return the next incoming ball. The feasibility of the third order polynomials that goes from hitting state $\mathbf{q}_f$, $\dot{\mathbf{q}}_f$ to $\mathbf{q}_0$, $\dot{\mathbf{q}}_0 = \mathbf{0}$ is ensured by including the joint extrema candidates throughout the returning trajectory in (63). Including the Frobenius norm of the Jacobian in the cost function makes sure that the striking trajectories will be easy to generate (i.e., have low accelerations) for the next predicted ball trajectories near the last ball trajectory.

Online trajectory generation. The resulting table tennis player is summarized in pseudocode format in Algorithm 2. As in Algorithm 1, the algorithm can be run online whenever there are enough ball samples $N = 12$ available to estimate the incoming ball reliably. The ball state is used, as before, to predict every $dt = 2$ ms, a discrete set of ball positions and velocities along the future ball path $\mathbf{b}(t)$, up to a horizon of $T_{\text{pred}} = 1.0$ s. The optimization for the striking trajectory (line 11) is then launched, which takes on average 25 ms to find a local optimum for the Barrett WAM. Good initialization and an optimized implementation make it possible to implement the approach online in our robot table tennis setup. See Section 6 for the implementation details.

After computing an initial striking trajectory and starting to move, the trajectories can be corrected online (line 11–13) whenever there are new ball samples available. Compared to *Focused Player*, the gained flexibility due to relaxed constraints is increased with the addition of the resting posture optimization (line 12) that reduces the accelerations of the next hitting movements for similar incoming balls.

Similar to Algorithm 1, we include in Algorithm 2 an arbitrary positive definite weighting matrix $\mathbf{R}$, which together with the task weights α_{hit} and α_{land}, adjusts the weight of a particular degree of freedom's accelerations in calculating the total cost (55).

Algorithm 2 *Defensive Player* (DP)

Require: $\mathbf{q}_0, \mathbf{R}, N, T_{\text{pred}}, \alpha_{\text{hit}}, \alpha_{\text{land}}$
1: Wait at initial posture $\mathbf{q}_0$.
2: **loop**
3: Query vision sys. for new observation $\mathbf{b}_{\text{obs}}$.
4: Observe current state $\mathbf{q}_{\text{cur}}, \dot{\mathbf{q}}_{\text{cur}}$.
5: **if** N new ball observations $\mathbf{b}_{\text{obs}}$ **then**
6: Initialize EKF.
7: **end if**
8: **if** EKF is initialized **and** valid obs. $\mathbf{b}_{\text{obs}}$ **then**
9: Estimate position $\mathbf{b}$ and vel. $\dot{\mathbf{b}}$ with EKF.
10: Predict $\mathbf{b}(t)$ till horizon T_{pred}.
11: Compute $\mathbf{q}_f, \dot{\mathbf{q}}_f, T$ from $\mathbf{q}_{\text{cur}}, \dot{\mathbf{q}}_{\text{cur}}$ using $\mathbf{b}(t)$ and the weights $\mathbf{R}, \alpha_{\text{hit}}, \alpha_{\text{land}}$.
12: Update $\mathbf{q}_0$ using $\mathbf{q}_f, \mathbf{b}(t)$
13: Update strike and return trajectories $\mathbf{q}_{\text{des}}(t) = \{\mathbf{q}_{\text{strike}}(t), \mathbf{q}_{\text{return}}(t)\}$.
14: **end if**
15: Track $\mathbf{q}_{\text{des}}(t)$ with Inv. Dyn. $\mathbf{fi} = \mathbf{f}(\mathbf{q}_{\text{des}}, \dot{\mathbf{q}}_{\text{des}}, \ddot{\mathbf{q}}_{\text{des}})$.
16: **end loop**

6. Experiments & evaluations

In this section, we will evaluate the performance of the online trajectory generation algorithms in simulation and in real robot table tennis experiments. We first start by comparing the ball returning performance of *Focused Player* (FP) in simulation against the virtual hitting plane (VHP) method.

6.1. Simulation studies

In simulation the performance of the new table tennis players can be extensively evaluated without robot control or ball prediction errors. We will make controlled experiments to first show that the player FP outperforms the VHP based player, and can generate striking trajectories more robustly.

Virtual hitting plane method. The VHP method that we implement is a close variant of Mülling et al. [3]. In this approach, the specification of the VHP (see Fig. 6) fixes the hitting time T as well as the hitting point $\mathbf{b}(T)$ for the racket trajectory. The remaining parameters, the desired racket velocity $\mathbf{v}_{\text{des}}(T)$ and the desired racket normal at hitting time $\mathbf{n}_{\text{des}}(T)$ are calculated by inverting the models (9) and (14) at the hitting time T.

To run inverse kinematics (IK) on the desired hitting point, one needs to additionally specify a desired racket slide [30]. An easy and convenient way to generate a desired racket slide at hitting time is to rotate the initial racket slide until the initial racket normal aligns with the final desired racket normal. This procedure determines the full orientation of the final robot posture at hitting time.

After specifying the orientation of the robot at hitting time, Jacobian pseudoinverse based IK can be run to determine the final joint positions. IK typically takes less than 2 ms to converge to $\mathbf{q}_f$. Final joint velocities are then found by using the Jacobian at hitting time $\mathbf{J}(\mathbf{q}_f)$) and the desired racket velocities

$$\dot{\mathbf{q}}_f = \mathbf{J}^{\dagger}(\mathbf{q}_f) \mathbf{v}_{\text{des}}(T). \tag{64}$$

The computed parameters $\mathbf{q}_f, \dot{\mathbf{q}}_f$ along with the fixed hitting time T fully determine a third degree polynomial in joint space for each degree of freedom of the robot $i = 1, \ldots, n$. The joint limitations are then checked and the procedure is repeated if the accelerations are too high. When the ball is coming close to the robot's initial posture $\mathbf{q}_0$, this complicated IK procedure results in feasible trajectories if the VHP is chosen appropriately. However, it is rather inflexible and can easily fail to find good configurations.

Table 2
Results comparing FP and VHP.

	Returns	Not valid	Infeasible	Miss/Out
FP	151	24	19	6
VHP	125	24	45	6

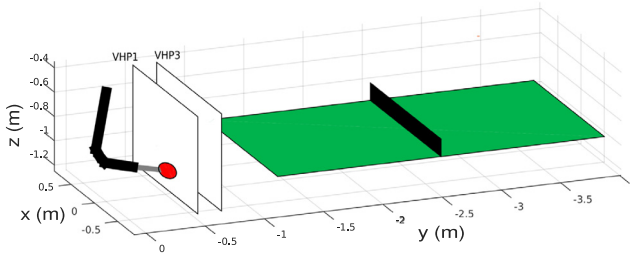
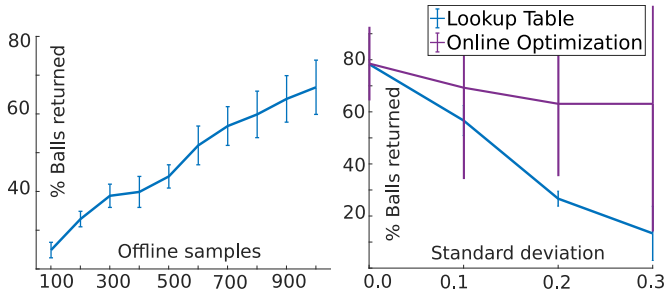


Fig. 6. For simulating the performance of the virtual hitting plane (VHP) based method in a fair way, the results are averaged over four different VHP locations. The first and third plane locations are shown in the figure. Out of 50 balls each, the VHP at $y = -0.7, y = -0.6, y = -0.5, y = -0.4$ return 31, 37, 28, 29 balls respectively.



(a) Lookup table performance vs. sample size.

(b) Performance vs. std. dev. of initial ball distribution mean.

Fig. 7. As an alternative to computing the trajectory parameters online, Koc et al. [4] proposed a lookup table to generate trajectories. Performance of the trajectory generation framework using a lookup table is shown in blue. Results are averaged over 5 different runs. As the number of stored lookup table samples increase, the performance approaches that of the online trajectory generation in (a). However, as shown in (b), even the performance of a lookup table with 4000 entries degrades quickly whenever ball position and velocity estimates are not close to the values stored in the lookup table. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Comparison with the VHP Method. To make a fair comparison between the VHP approach and our algorithm FP, in our simulation environment[1] the initial ball state variance is fixed such that most balls end up close to the initial robot posture. This ensures that a fair evaluation between the two algorithms can be given. Both methods filter the incoming stream of ball position estimates using the same Extended Kalman Filter (EKF) and equally start moving whenever $N \approx 12$ balls are detected.

Evaluations are summarized in Table 2. A total of 200 balls are launched towards the robot in single-ball solo trials from varying initial positions and velocities, $\mathbf{b}_{\text{init}} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{init}}, \sigma^2_{\text{init}}\mathbf{I})$. The initial ball mean positions are fixed on the left corner of the opponent's court and the initial covariance matrix is diagonal with a standard deviation of $\sigma_{\text{init}} = 0.1$. Some balls are illegal, for example they might not bounce on the robot's court. Such balls are detected with our ball prediction models and they are not considered for strike generation. They are marked as *Not Valid* in Table 2.

Comparing with the VHP method, it can be seen that FP is able to return more balls to the other side, with 26 more balls returned to the opponent's court. One of the main reasons for this increase in performance is the fixed location of the VHP. Depending on the incoming ball velocity, trajectories generated using a fixed VHP can result in joint limit violations or infeasible solutions. A second reason is the explicit incorporation of joint limits both for the striking trajectory and the returning trajectory in the optimization problem. Both cases are included as *Infeasible* in Table 2. See Fig. 6 for an illustration. Out of 50 balls each, the VHP methods with the planes fixed at $y = -0.7$, $y = -0.6$, $y = -0.5$, $y = -0.4$ locations return 31, 37, 28, 29 balls respectively. For this particular ball distribution, the plane at $y = -0.6$ seems to be the most robust option. In terms of landing point accuracy, both methods achieve a roughly isotropic Gaussian distribution with variance $\sigma^2 = 0.15m$, with varying desired landing points $\mathbf{b}_{\text{goal}}$ as the mean.

Lookup table. A naive implementation of FP in MATLAB using Sequential Quadratic Programming (SQP), takes about two seconds on our system on average. [4] proposed a lookup table as a remedy to replace online optimization. Whenever a strike computed offline is successful in returning the ball in simulation, ball positions, velocities at the start of the movement and the optimized parameters $\mathbf{q}_f, \dot{\mathbf{q}}_f, T$ can be stored in a lookup table. One can then at runtime simply lookup the optimized parameters that have the closest stored ball position and velocity estimates to new ball estimates. The performance of this lookup table based approach is evaluated in Fig. 7. The same initial ball distribution with the same $\boldsymbol{\mu}_{\text{init}}, \sigma_{\text{init}}$ values is used as before. As the number of lookup table samples increase, the percentage of incoming balls returned successfully approaches that of the online trajectory generation.

The simple lookup table approach that is employed here corresponds to k-nearest neighbor interpolation with $k = 1$. Machine learning based methods that regress between lookup table entries using more sophisticated approaches are discussed extensively in, for example, [31].

Online trajectory generation. The lookup table proposed above is based on a fixed initial posture $\mathbf{q}_0$ while the robot is at rest, i.e., $\dot{\mathbf{q}}_0 = \mathbf{0}$. Its performance degrades whenever filtered ball positions $\mathbf{b}_0$ and velocities $\dot{\mathbf{b}}_0$ are not close to the values stored in the lookup table, or when the initial posture is different. See Fig. 7 for the decrease in performance of a lookup table with 4000 entries, as the mean of the initial ball distribution, $\boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{init}}, \sigma^2\mathbf{I})$, is randomized with increasing variances σ^2.

To overcome the shortcomings of a lookup-table based approach, we implemented the optimizations in C++ with an interface to the simulation environment SL [32].[2] SL is also our real-time interface to the robot and runs at a frequency of 500 Hz, terminating any processes that do not finish within 2 milliseconds. It is mainly responsible for running the inverse dynamics and feedback control loop computations. To run the optimization online, a thread separate from the one running the inverse dynamics is launched, whenever there are reliable ball observations available and another thread is not running.

We use the NLopt library [33] to run both optimizations. For the *Focused Player* (FP), we found that among the nonlinear constrained optimization routines, only COBYLA respects the equality constraints given in (18)–(20). The algorithm COBYLA is a simplex method implemented in NLopt that uses direct search with linear approximations [34]. Gradients of the cost function (30) can be easily calculated and fed to a gradient based solver, but this direct

[1] Code for the simulation platform as well as a video showing some example trajectories is available in the GitHub repository: https://github.com/RobotLearning/traj-gen-and-tracking.git.

[2] C++ code for the online optimization run in the real-time simulation platform SL can be found in: https://github.com/RobotLearning/polyoptim.git.

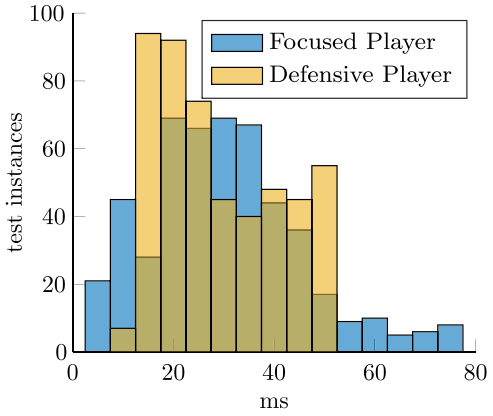
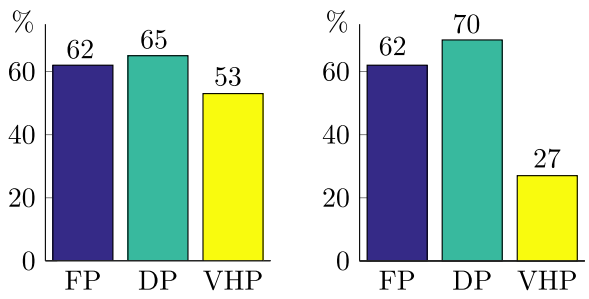


Fig. 8. Histogram of the runtime distributions of the two players, evaluated over 500 random test instances. Both algorithms FP and DP have an average runtime of about 25 ms, but for FP, the distribution is wider. For evaluating DP we have regressed on a lookup table using k-nearest-neighbor (kNN) regression. Without kNN, the runtime distribution for DP concentrates sharply around 50 ms.



(a) Performance under ball observation noise.

(b) Performance under additional ball model mismatch.

Fig. 9. Simulation results comparing the return accuracy of three table tennis players. In (a), ball positions are observed with Gaussian white noise. In (b), there is an additional mismatch due to unknown topspin. Out of 200 balls, 14 and 12 incoming balls did not bounce legally and were not considered for trajectory generation, respectively. The other balls that were not counted as returns were either missed, or did not land legally on the opponent's court.

search routine takes only about 25 ms to converge, i.e., about the same frequency as the incoming ball observations.

Before the optimization for robot striking movements take place, the desired outgoing ball velocities and the corresponding racket parameters $\mathbf{v}_{\text{des}}(t)$, $\mathbf{n}_{\text{des}}(t)$ necessary to enforce the equality constraints (18)–(20) are predicted every 2 ms for $0 \leq t \leq T_{\text{pred}}$. The prediction horizon $T_{\text{pred}} = 1.0$ is more than enough, given the speed of the balls, for the balls to pass the robot workspace. Racket computations take on average 1–1.5 ms for the whole sequence of predicted ball states. The discretization of 2 ms is natural, since the robot control runs at 500 Hz. During the optimization for a continuous T, the corresponding racket variables (racket desired positions, velocities and normals) are interpolated linearly between the discrete predicted values.

For the *Defensive Player* (DP), the Augmented Lagrangian (AUGLAG) method is used to convert the problem to an unconstrained optimization problem, which is then solved with a Quasi-Newton algorithm. In this case, only incoming ball positions and velocities are predicted, again discretized over 2 milliseconds.

Good initialization does not always guarantee faster convergence, but it can help escape bad local minima of the cost functions. The optimization parameters for FP are first initialized to the resting posture, $\mathbf{q}_f = \mathbf{q}_0$, $\dot{\mathbf{q}}_f = \mathbf{0}$ and $T = 0.5$ s. Whenever the robot is already moving and corrections are being computed, the parameters are initialized to their previously computed values. For DP, we initialize by regressing on a lookup table using k-nearest-neighbor (kNN) regression, with $k = 5$, to speedup the optimization process. Fig. 8 shows the runtime distributions of the two algorithms over 500 test instances. In each trial, the ball is launched from different sides with the same distribution as described before, and the robot initial posture is also chosen randomly. Both algorithms FP and DP have approximately an average runtime of 25 ms, but for FP, the distribution is wider. Without regressing on a lookup table, the distribution for DP concentrates sharply around 50 ms.

Online corrections. In order to show the performance increase due to online corrections, we first put Gaussian white noise with $\sigma = 0.02$ m standard deviation on the ball observations and apply Extended Kalman Filter (EKF). As the ball approaches, the robot gets increasingly better estimates of the ball state. Our real-time simulator runs at 500 Hz, while the ball observation is limited to 60 Hz, limiting the frequency of online corrections. The results are summarized in Fig. 9(a), averaged over three different ballgun locations: left, center, and right. The initial pose of the robot is placed opposite accordingly, i.e. on the right side of the table if the ball is coming from the left. Out of 200 balls, 14 incoming balls did not bounce legally and were not considered for trajectory generation. The other balls that were not returned successfully were either missed, or did not land legally on the opponent's court. The online optimization is started whenever there are $N = 12$ ball samples available. This is enough to ensure that the estimated ball velocities will not cause robot movements that are far off from the ball. The solver then continues at a rate of 25 Hz until the ball appears behind the racket center, i.e., $b_y > r_y$. Any ball that suddenly appears on the opponent's court causes the Kalman Filter to reset, reinitialized with that ball observation as the initial mean and with a high variance.

The players that generate trajectories only once are able to return only very few balls (10 on average) in this mode of evaluation. Correcting with the VHP method improves the returning performance significantly. However, balls that are not feasible for the robot in the Virtual Hitting Plane intersection cannot be returned at all with this player. *Focused Player* (FP) and *Defensive Player* (DP) on the other hand, can find and generate feasible movements more flexibly. FP and VHP methods both return the balls with a roughly isotropic Gaussian distribution around the center of the table, with a variance of $\sigma^2_{\text{land}} = 0.3m$, while the ball landing positions $\mathbf{b}_{\text{land}}$ of DP follow roughly a uniform distribution.

As an additional challenge, we also consider the model mismatch case where there is a very high topspin on the ball, around 3000 revolutions per minute (rpm). EKF assumes a nonspinning model for the ball, i.e., $C_L = 0$. The solver is run with an increased rate of 50 Hz to be able to return the balls. The players that generate trajectories only once are not able to return any balls in this aggressive mode of evaluation. The results are summarized in Fig. 9(b). 12 incoming balls out of 200 did not bounce legally and were not considered for trajectory generation. As in the previous experiment, FP and DP correct the trajectories more easily and overall return more balls. The advantage of DP over FP in this setting is due to the more flexible returning criterion, as the resting state optimization was not applied. In terms of landing point accuracy, both algorithms have similar ball landing distributions as before, but the means have an offset of $0.3m$ closer to the other side of the table. The offset is due to the fact that the racket computations for FP and the landing point calculations for DP in this case do not assume a spin model for the ball.

6.2. Real robot table tennis

In this section we describe and discuss our experiments on the robotic table tennis setup, see Fig. 1.

Description of the setup. Our robot is a seven degree of freedom Barrett WAM arm that can easily reach 10g m/s^2 accelerations. It is torque-controlled and cable-driven. A standard size racket (7.6 cm radius) is attached to the end-effector. The vision system tracks the balls at a rate of 60 Hz and consists of four cameras on the corners on the ceiling. See [35] for platform details. The table and the tennis balls are standard sized, the balls have a radius of 2 cm, the table geometry is approximately 276 × 152 × 76 cm.

In the robot experiments, we use a ball-launcher (see Fig. 1) to throw balls to the robot, approximately once every 2–3 s. The balls generally come with a high variance, especially the velocities are quite unpredictable even without oscillating the ball-launcher. The robot base is at a distance of 115 cm to the end of the table and 95 cm above the table. Robot base is centered with respect to the table in the x direction, see Fig. 1.

Estimation and outlier detection. The ball detection algorithm detects the center of mass of the orange balls from each image separately and fuses them together to form two ball position estimates. These are then filtered with an Extended Kalman Filter (EKF) to estimate ball position and velocity.

After the ball-launcher shoots a ball, $N = 12$ ball observations are used to initialize the Kalman Filter state and launch the trajectory generation process, see Algorithms 1 and 2. Balls that suddenly appear on the opponent's court after disappearing for more than 0.5 s from the cameras cause the Kalman Filter to reset. The filter state and the ball spin (assumed constant throughout motion) are then estimated together with a truncated Newton's method[3] using $N = 12$ ball samples. We have experimentally confirmed the value of N to be a good compromise between ball estimation accuracy (which requires waiting) and moving early (which can reduce the accelerations).

Online correction of computed trajectories. Since the ball is moving at fast speeds, our online trajectory generation algorithms need to be on the order of tens of milliseconds, in order to reliably intercept the incoming ball. The optimizers take on average 25 ms to converge, and they can be re-run in the real-time platform whenever there are new reliable ball observations $\mathbf{b}_{\text{obs}}$. Before launching the trajectory optimizers, the path of the ball is predicted each time for $T_{\text{pred}} = 1.0$ s and the algorithms are initialized with current joint state estimates $\mathbf{q}_{\text{cur}}, \dot{\mathbf{q}}_{\text{cur}}$. The optimizations are performed in the same way as described in the previous sections. The only difference is that during the optimization process, the ball is approaching the robot, hence we subtract the optimization time T_{run} from the computed desired hitting time T before generating the hitting trajectory, i.e., $T \leftarrow T - T_{\text{run}}$.

During the correction process, we make sure that the updates are always incremental and feasible. For completeness, we list here our software checks. We make sure that:

1. At least $N = 12$ reliable ball observations are available. This typically happens before the balls pass the net.
2. The new ball estimate is not too far off from the previous estimates.
3. Our previous optimization thread has terminated before another one is launched.
4. The resulting Cartesian trajectory intersects with the ball and all the task constraints are satisfied.
5. The corrections are never excessive, i.e., the acceleration and joint limits (34) and (35) are always respected.
6. The ball estimate appears to be in front of the robot, i.e., $b_y < r_y$.

[3] The optimization is launched on another thread using the *TNEWTON* algorithm in NLopt [33].

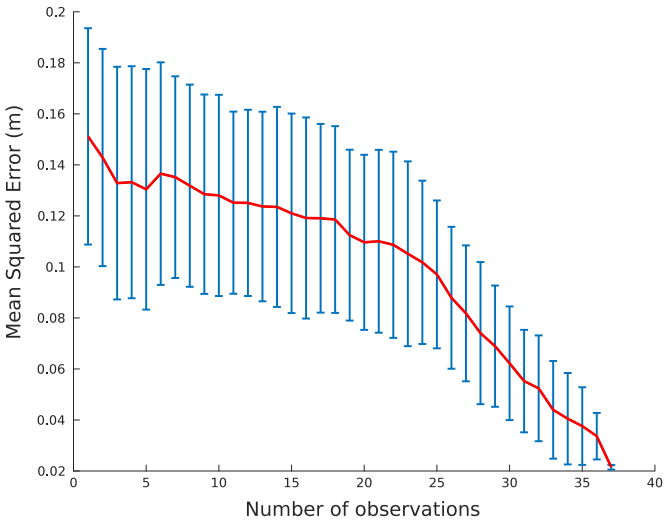


Fig. 10. Mean squared prediction error (red curve) is reduced as more balls are observed. The ball observations are used until contact with racket occurs and the results are averaged over 100 different real ball trials. Correcting for ball prediction error is critical for a robust table tennis performance, as the balls typically come with a high spin. Balls seem to lose some spin after rebound and the prediction error decreases faster. In this case this phenomenon can be observed after about 25 ball observations, where the change in the average slope of the red curve can be seen.

If any of these conditions are violated, then the trajectories are not updated, and the previous striking trajectory is followed without interruption. The balls come with a high variance in position and especially in velocity. Typical incoming ball velocities imparted by the ball-launcher are around 4–6 m/s range in the y-direction, which implies that in practice there can be a maximum of 10 ball corrections till the ball passes the robot. The ball-launcher gives in addition a lot of *topspin* to the ball. This makes the corrections provided by the repeated optimization critical, as the ball models (9)–(14) are unable to capture some of the aerodynamic effects due to spin. Fig. 10 shows the decrease in mean squared prediction error as more ball observations are acquired.

Discussion of results. We compare and evaluate the performance of the two players FP and DP in the robot table tennis setup, see Figs. 11–13. Results are averaged over 200 trials where the ball-launcher is fixed at different positions or is oscillating, and the robot is placed at three different initial postures. For FP, we also consider the variation in performance due to selecting different desired landing positions and landing times. Overall, FP is able to return about 40–60% of the balls to the opponent's court. Setting the desired landing position on the right side of the table, with a desired landing time of $T_{\text{land}} = 0.4$ s, leads to the best performance (∼60%) in our table tennis setup. The ball landing distribution follows roughly an isotropic Gaussian distribution with variance $\sigma^2_{\text{land}} = 0.2m$ and a mean offset of $\|\mu_{\text{land}} - \mathbf{b}_{\text{goal}}\| = 0.25m$, see Fig. 12b. Increasing the time and setting the desired landing position closer to the center of the opponent's court makes the player less robust, decreasing the accuracy down to 40−50% and increasing the variance of the landing locations, see Fig. 12a. We believe this decrease in the performance is due to inaccuracies in the racket model.

The *Defensive Player* (DP) is able to return about 80−90% of the balls, the performance varying depending on the incoming balls and the ballgun settings. The gain in accuracy is due to the increased flexibility of the algorithm, as well as the additional resting posture optimization which simplifies the task significantly. The algorithm finds counterintuitive resting postures that lead to

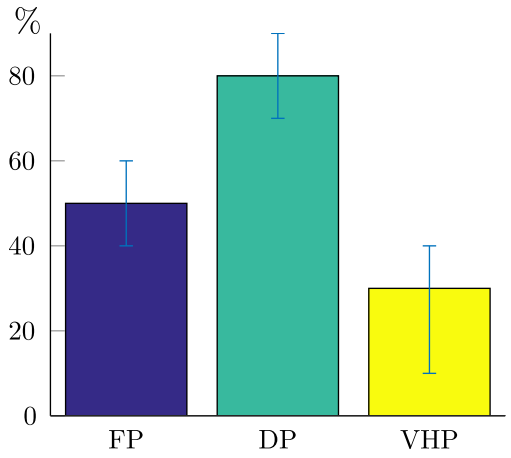


Fig. 11. Summary of real robot table tennis experiment results comparing three table tennis players. Bar plot values show the successful return % averaged over different starting postures and initial ball positions. The error bars indicate the standard deviation over a total of 200 trial runs.

smaller movements with less control error, see Fig. 14 for four consecutive trials of DP. The ball landing distribution in this case is roughly uniform but shifted to the left side of the table, with an offset mean of about $0.15m$, see Fig. 12c. The duration of the returning trajectory $T_{\text{rest}} = 1.0$ s for all players. The weighting matrix $\mathbf{R}$ is set to identity and the weights for hitting and landing penalties are both set to ten: $\alpha_{\text{hit}} = 10$, $\alpha_{\text{land}} = 10$.

VHP can return about 10–40% of the balls. The best setting for the hitting plane location depends strongly on the ballgun settings, which affect the distribution of the incoming ball. In our experiments the hitting plane at $y = 30$ cm in front of the robot lead to the best performance (∼40%). However, the accuracy can drop down significantly (to 10%) if the ballgun is oscillating, or the initial ball velocities are not appropriate for the particular VHP setting. For all three algorithms, without applying any corrections, the robot is able to hit most balls but cannot return most balls successfully to the other side (only 5% of the balls are returned). Applying the corrections about three times, and at least once after rebound, increases the performance to the indicated values, see Fig. 11. This indicates that the rebound model chosen might not be accurate with high topspins that the ballgun imparts to the ball (around 3000 rpm). Two example trials are shown in Fig. 13. The deviation from the desired hitting point, shown as an orange dot, was for the first example within three cm of the racket center, resulting in a hit. The deviations in the reference velocities are higher and lead to approximately 10 cm/s difference in Cartesian space. The successful strike and the landing on the opponent's court can be seen in the upper right figure. In the first example, player FP tries to return the ball to the right side of the opponent's court, with a desired landing time of $T_{\text{land}} = 0.4$ s. The blue dots are the ball observations acquired from cameras 3 and 4, which are located on the corners of the ceiling on the robot side. In the second example (screenshots 3–6), the player DP also returns the ball successfully, but unlike the other player, DP does not bring the robot back to the same initial posture. Control errors on the joint positions and velocities for this example are shown in Fig. 15. After the desired trajectories are calculated, high gain PD-control is applied along with an inverse dynamics controller (computed-torque). The inverse dynamics model is not very precise, but the feedback with high gains compensates for it well, especially in the shoulders and the elbow.

7. Conclusion

In this paper we have presented two new algorithms (FP and DP) for generating table tennis striking trajectories that extend previous work in table tennis strike movement generation.

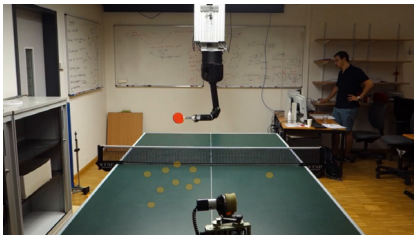
7.1. Summary of the contributions

The two table tennis players use an optimal-control based approach for generating striking trajectories. The striking and return trajectories are third order polynomials that intercept the ball at the optimized hitting point at the optimized hitting time. Unlike previous approaches, our optimization based framework respects the joint limits, while leading to efficient movements with low accelerations. Furthermore, by varying the hitting time T the problem of finding feasible joint trajectories is simplified. Further constraints can be easily imposed on the system, and we have considered, for instance, racket constraints for FP and an additional resting posture optimization for DP.

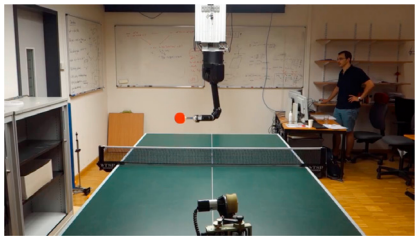
The optimizations can be run online in the robotic setup shown in Fig. 1 and given new joint position and ball position measurements, the trajectories can be updated. Correcting for new ball positions, by repeating optimization, makes our table tennis players more robust to execution errors and inaccuracies in ball estimation & prediction. We show the performance of our two table tennis players in the real robot platform and compare with previous approaches.

7.2. Outlook & future work

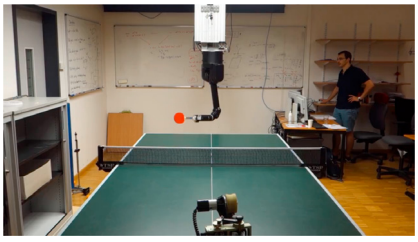
The two players *Focused Player* and *Defensive Player* can generate trajectories more flexibly than before and lead to two different play-styles which could potentially be utilized by a higher-level strategy. We believe that this is a promising direction, where a higher level learning algorithm could switch between different



(a) FP with the desired ball landing location at the center of the opponent's court.



(b) FP with the desired ball landing location at the right side.



(c) DP with a more flexible returning criterion.

Fig. 12. Overall, FP is able to return about 40–60% of the balls to the opponent's court. Setting the desired landing position on the right side of the table, with a desired landing time of $T_{\text{land}} = 0.4$ s, leads to the best performance (∼60%) in our table tennis setup. Some example landing locations are indicated in orange in (b). Setting the desired landing position closer to the center of the opponent's court decreases the accuracy down to 40–50%, increasing also the variance of the landing locations, as shown in (a). DP in (c) with a landing accuracy of 80% has the highest variance in terms of the ball landing locations, as its returning criterion considers the whole opponent's court.

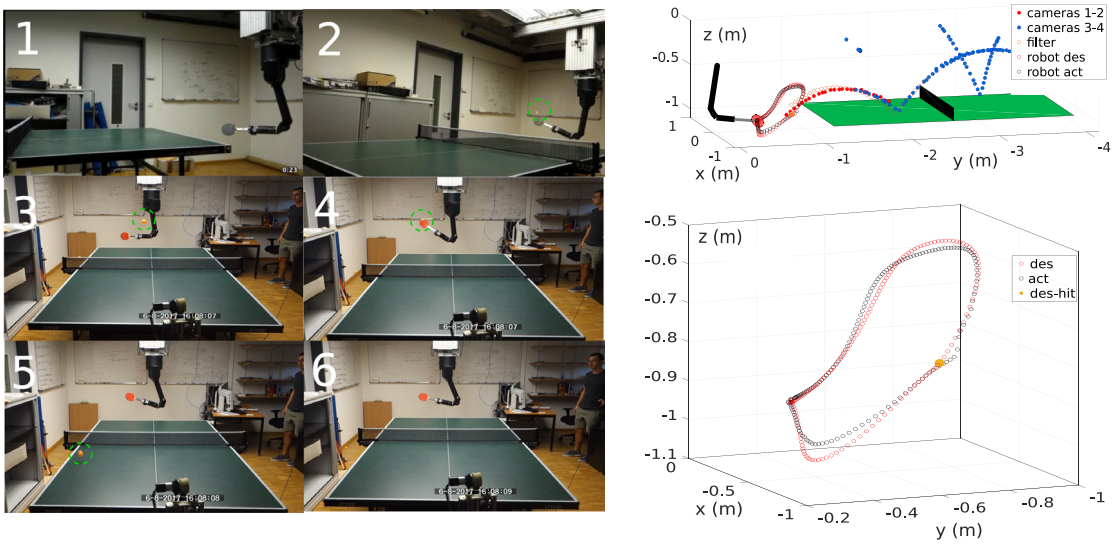


Fig. 13. Two example table tennis trials recorded in the table tennis setup are shown on the left hand side. The top two screenshots show the *Focused Player* (FP) in action, and the bottom four the *Defensive Player* (DP). Unlike FP, DP does not bring the robot back to the same initial posture (screenshots 3 vs. 6). Successful strike and the valid landing on the opponent's court for DP can be seen in the screenshots 4—5. Balls are highlighted with green dashed circles for visibility. The plot in the upper right figure shows the recordings from the cameras and the robot sensors, corresponding to the hitting movement in screenshots 1 and 2. The blue dots are the ball observations coming from cameras 3 and 4. The desired Cartesian trajectory is drawn in red, and the actual trajectory, in black. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

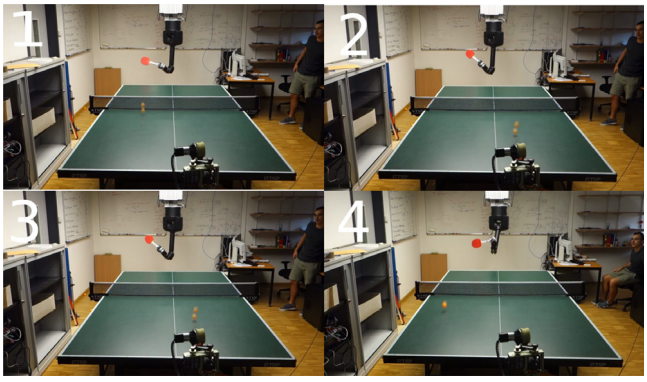


Fig. 14. Four consecutive lands are shown for the *Defensive Player* (DP). In each trial, the arm goes back to a different resting posture.

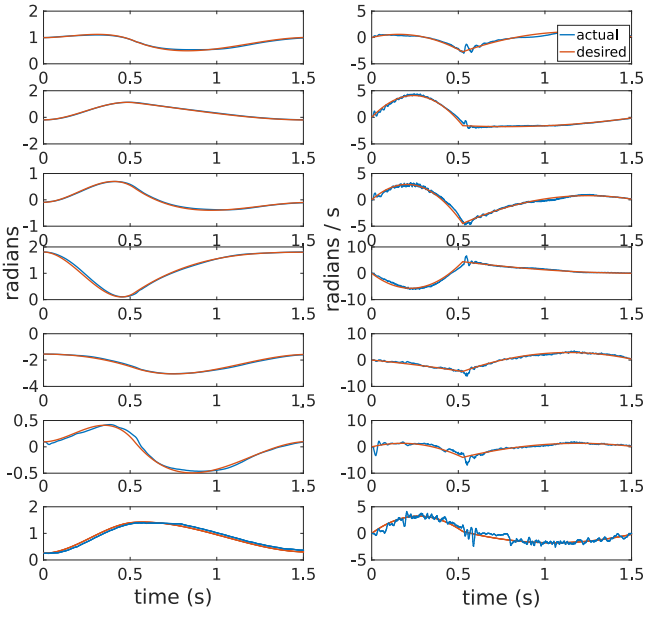


Fig. 15. Tracking errors are shown for each joint. The desired joint positions and velocities are tracked with a PD controller. The deviation from the desired hitting point, shown as an orange dot in Fig. 13, was for this example within three centimeters of the racket center, resulting in a hit.

trajectory generation schemes. The weights and the additional parameterization for the two algorithms can be explored based on feedback on the robot's performance. Reinforcement learning [36] with rewards based on observed ball landing positions, provides a suitable framework to tune the proposed algorithms' performance online.

Finally, the cost functionals that we have introduced consider the accelerations as the quantity to be minimized. Whenever the cancellation in feedback linearization is imperfect due to inaccurate robot dynamics models, execution errors will prevent the robot from achieving the desired trajectories or the minimal accelerations. A more robust and adaptive way to include execution errors in trajectory generation will be considered in future work.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7-ICT-2013-10) under Grant agreement 610878 (3rdHand). This article is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

Appendix A. Parameter estimation

In order to reach successful performance in real robot table tennis, accurate ball prediction models are needed. For this purpose, we collect data from a human table tennis demonstration recording and estimate the parameters of the ball models. In these sessions, we record the ball position observations from the cameras as well as the robot joint angles. A ball-launcher is used to launch balls with high topspin. The noisy dataset of human table tennis demonstrations

$$\mathcal{D} = \{\mathbf{t}_i, \mathbf{b}_i, \mathbf{q}_i\}_{i=1}^{N}, \tag{A.1}$$

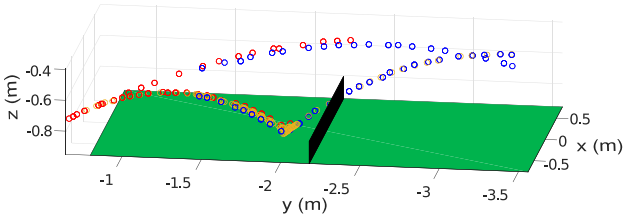


Fig. A.16. Using Extended Kalman Smoothing (EKS) to estimate the parameters of the rebound model from actual noisy ball data during a demonstration recording. Ball observations are acquired from two different sets of cameras on opposite sides of the table, shown as red and blue circles respectively. They are then smoothened with the EKS, shown in yellow, to obtain velocity estimates before rebound and just after rebound. Nonlinear least squares is then used to estimate the rebound model parameters μ_t and ϵ_t. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

consists of $N = 90$ trials. Each trial i contains M_i ball position and K_i joint position recordings sorted by time, i.e.,

$$\mathbf{b}_i = \{\mathbf{b}_{ij}\}_{j=1}^{M_i}, \mathbf{q}_i = \{\mathbf{q}_{ij}\}_{j=1}^{K_i}, \tag{A.2}$$

sorted by $\mathbf{t}_i = \{t_{ij}\}_{j=1}^{K_i}$. Typically $K_i > M_i$, e.g., for the Barrett WAM, we record the robot joint values with a frequency of 500 Hz, whereas our vision system outputs ball observations at around 60 Hz.

Whenever the future path of the ball is predicted with the ball models, the accuracy of the predictions using the rebound model (10) and the racket contact model (14) clearly depend on that of the flight model (9). Hence we first start by estimating the parameters of the flight model using nonlinear least squares (NLS). We collect the time stamps and the ball positions detected by the vision system before rebound. The rebound index for each trial i is estimated as

$$\begin{aligned} j_b(i) &= \arg\min_j b_{z_{ij}}, \\ \text{s.t. } &-\tfrac{w_T}{2} \le b_{x_{ij}} \le \tfrac{w_T}{2}, \\ &y_{\text{edge}} - l_T \le b_{y_{ij}} \le y_{\text{edge}}. \end{aligned} \tag{A.3}$$

We then use all of the observations until rebound, $\{(t_{ij}, \mathbf{b}_{ij}\}_{j=1}^{j_b(i)}\}$, to estimate the parameters g, C_D and C_L. This procedure requires to estimate as well the trial specific topspin parameter, i.e., $\boldsymbol{\omega} = (0, 0, \omega_z)^{\mathrm{T}}$ for some topspin value w_z, separately for each trial $i = 1 \ldots, N$. We use the flight model with the estimated parameters to smoothen the ball path before rebound and after rebound. Using an Extended Kalman Smoother [29], the ball velocities are calculated before and after rebound, $\dot{\mathbf{b}}_{\text{in}}, \dot{\mathbf{b}}_{\text{out}}$ respectively. NLS is again used to estimate the rebound parameters μ_t and ϵ_t. See Fig. A.16 for an illustration.

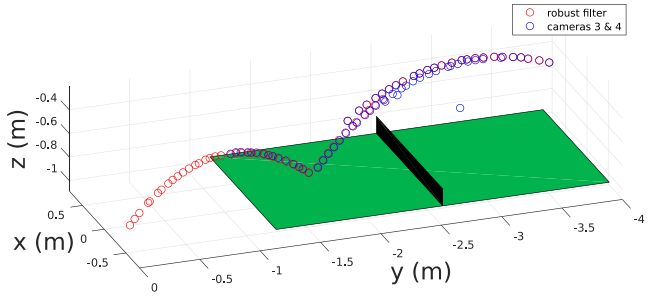
In order to estimate the parameters, i.e., κ and ϵ_r, of the ball–racket interaction model (14) we first use the dataset to estimate the striking times T_i. By considering the minimum distance between the ball samples and the demonstrated Cartesian robot trajectories

$$\begin{aligned} j_h(i) &= \arg\min_j \|\mathbf{b}_{ij} - \mathbf{K}_p(\mathbf{q}_{ij})\|_2, \\ T_i &\approx t_{ij_h(i)}, \end{aligned} \tag{A.4}$$

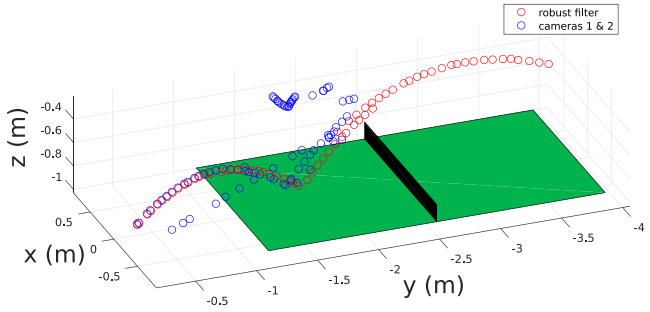
for each trial $i = 1, \ldots, N$ we can roughly estimate the striking times T_i. As in estimating the rebound model parameters, the Extended Kalman Smoother is then used to smoothen the ball demonstrations before and after the striking time separately. This procedure results in estimating the incoming and outgoing ball velocities, $\dot{\mathbf{b}}_{\text{in}}(T_i)$, $\dot{\mathbf{b}}_{\text{out}}(T_i)$ as well as the required racket quantities $\mathbf{v}(T_i)$, $\mathbf{n}(T_i)$ at striking times T_i. Linear Least Squares is then used with regularizer $\lambda = 0.001$ to estimate the coefficients κ and ϵ_r. See Table A.3 for the estimated values of all the ball model parameters.

Table A.3
Ball model parameter estimates.

Parameter	Description	Estimate
C_D	Air drag coefficient	0.141 1/m
C_L	Lift coefficient	0.001 1/rad
g	Gravity	-9.802 m/s^2
μ_t	Coeff. of friction of table	0.102
ϵ_t	Coef. of restitution of table	0.883
κ	Coeff. of friction of racket	0.020
ϵ_r	Coeff. of restitution of racket	0.788



(a) Filtering the noisy and corrupted data acquired from cameras 3 and 4 on the robot side.



(b) Filtering the noisy and corrupted data acquired from cameras 1 and 2 opposite to the robot side.

Fig. B.17. Kalman Filtering with simultaneous outlier detection. The ball detection algorithm sometimes outputs outliers, typically more as the ball approaches the racket. Such corrupted data can be identified and discarded using the covariance of the Kalman Filter.

An alternative approach would be to estimate all the model parameters together with a smoothing Expectation–Maximization (EM) [37] algorithm, yielding additionally covariance estimates for noisy ball observations.

Appendix B. Robust Kalman Filtering

The ball detection algorithm sometimes outputs outliers, possibly meters away from the actual ball. This typically happens more as the ball approaches the racket and new ball observations become more valuable. In order to prevent the outliers from ruining the estimation and the overall performance, we have implemented a robust EKF that does not perform measurement updates, if the ball observations lie more than 2 standard deviations away from the predicted state. See Fig. B.17 for actual table tennis ball data. We adjust the covariance estimates $\Sigma(t)$ accordingly to make this procedure work in practice, e.g., covariances are initialized with a large Σ_{init} value and the noise covariances $\mathbf{W}(t) \approx 10^{-3}\mathbf{I}$ are adjusted to make sure that $\Sigma(t)$ decreases suitably over time.

Appendix C. Derivations

We will give a self-contained derivation of the fact that the solutions to the parametric optimization problem (55) are also locally optimal solutions to (1). The same holds for the solutions of (30) under the additional racket constraints (18)–(20). First, we start by deriving the fixed final-time version of the Minimum Principle (MP) from the Hamilton Jacobi Bellman equation (HJB). Sufficient continuity of the introduced variables are assumed throughout to simplify the presentation. The solutions of the following fixed final-time optimal control problem

$$\begin{aligned} &\min_{\mathbf{u}\in\mathcal{U}} \int_0^T l(\mathbf{x},\mathbf{u})\,\mathrm{d}t + \phi(\mathbf{x}(T)), \\ &\text{s.t. } \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}), \end{aligned} \tag{C.1}$$

from arbitrary initial conditions are given by the solutions of the HJB partial differential equation (PDE)

$$\min_{\boldsymbol{\pi}(\mathbf{x},t)} \left(l(\mathbf{x},\boldsymbol{\pi}) + \frac{\partial V(\mathbf{x},t)}{\partial \mathbf{x}}^{\mathrm{T}} \mathbf{f}(\mathbf{x},\boldsymbol{\pi}(\mathbf{x},t)) \right) + \frac{\partial V(\mathbf{x},t)}{\partial t} = 0, \tag{C.2}$$

$$V(\mathbf{x},T) = \phi(\mathbf{x}(T)). \tag{C.3}$$

For every state $\mathbf{x}$, $V(\mathbf{x},t)$ is the optimal cost to (C.1) for following the optimal policy $\boldsymbol{\pi}(\mathbf{x},t)$, also called the *value function*. MP [26] can be seen in the smooth case as a solution technique[4] that reduces (C.2) to a family of ordinary differential equations (ODE). Introducing the *momenta* $\mathbf{p}(t) = \frac{\partial V}{\partial \mathbf{x}}$ and the *Hamiltonian* $\mathcal{H}(\mathbf{x},\mathbf{u},\mathbf{p}) = l(\mathbf{x},\mathbf{u}) + \mathbf{p}(t)^{\mathrm{T}}\mathbf{f}(\mathbf{x},\mathbf{u})$, the HJB equation (C.2) becomes

$$\min_{\mathbf{u}(t)} \mathcal{H}(\mathbf{x}(t),\mathbf{u}(t),\mathbf{p}(t)) + \tfrac{\partial V}{\partial t} = 0. \tag{C.4}$$

Given an initial condition $\mathbf{x}(0) = \mathbf{x}_0 \in \mathbb{R}^n$, solving (C.4) is reduced to solving a Boundary Value Problem (BVP)

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t),\mathbf{u}^*(t)), \tag{C.5}$$

$$\dot{\mathbf{p}}(t) = \frac{\partial}{\partial t}\frac{\partial V(\mathbf{x},t)}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{x}}\frac{\partial V}{\partial t} = -\frac{\partial \mathcal{H}(\mathbf{x},\mathbf{u}^*,\mathbf{p})}{\partial \mathbf{x}}, \tag{C.6}$$

$$\mathbf{u}^*(t) = \arg\min_{\mathbf{u}} \mathcal{H}(\mathbf{x}(t),\mathbf{u}(t),\mathbf{p}(t)), \tag{C.7}$$

with prescribed boundary values

$$\begin{aligned} &\mathbf{x}(0) = \mathbf{x}_0, \\ &\mathbf{p}(T) = \left.\frac{\partial V(\mathbf{x},T)}{\partial \mathbf{x}}\right|_{\mathbf{x}(T)} = \left.\frac{\partial \phi(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}(T)}. \end{aligned} \tag{C.8}$$

The state equation (C.5) and the *costate* equation (C.6) that describes the evolution of the introduced momenta are $2n$ first-order coupled ODEs that can be solved numerically with a BVP solver.

Free final-time. If the final time T is free, the optimal control problem

$$\begin{aligned} &\min_{\mathbf{u},T} \int_0^T l(\mathbf{x},\mathbf{u})\,\mathrm{d}t + \phi(\mathbf{x}(T),T), \\ &\text{s.t. } \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}), \end{aligned} \tag{C.9}$$

can be transformed to the standard form (C.1) by using the transformation $\tau = Tt$

$$\begin{aligned} &\min_{\mathbf{u},T} \int_0^1 Tl(\mathbf{x},\mathbf{u}) + \phi(\mathbf{x}(T),T)\,\mathrm{d}\tau, \\ &\text{s.t. } \frac{d\mathbf{x}}{d\tau} = T\mathbf{f}(\mathbf{x}(\tau),\mathbf{u}(\tau)), \end{aligned} \tag{C.10}$$

and the associated HJB, similar to (C.4) can be written as

$$\min_{\mathbf{u}(t),T} (T\mathcal{H}(\mathbf{x}(t),\mathbf{u}(t),\mathbf{p}(t)) + \phi(\mathbf{x}(T),T)) + \tfrac{\partial V}{\partial t} = 0. \tag{C.11}$$

In this case, the state and the costate differential equations are preserved

$$\begin{aligned} \frac{d\mathbf{x}}{d\tau} &= \mathbf{f}(\mathbf{x}(\tau),\mathbf{u}(\tau)), \\ \frac{d\mathbf{p}}{d\tau} &= -\frac{\partial \mathcal{H}(\mathbf{x},\mathbf{u}^*,\mathbf{p})}{\partial \mathbf{x}}, \end{aligned} \tag{C.12}$$

while the additional minimization with respect to T in (C.11) yields an additional *time variation* to constrain the Hamiltonian

$$\mathcal{H}(\mathbf{x},\mathbf{u}^*,\mathbf{p}) + \frac{\partial \phi(\mathbf{x},T)}{\partial T} = 0. \tag{C.13}$$

The $2n$ first-order coupled ODEs (C.12) can again be solved numerically with a BVP solver, but with the additional parameterization T of the boundary values (C.8) that is resolved implicitly through (C.13).

Equality constraints. When m equality constraints at final time T are present rather than a final cost term ϕ, the (generalized) HJB equation (C.2) still holds but the boundary term (C.3) constraining the value function is replaced with an equality constraint $\boldsymbol{\Psi}(\mathbf{x}(T),T) = \mathbf{0}$. The value function is undefined for states violating the equality constraint at final time, and zero otherwise [38].

Equality constraints at the boundaries do not affect the differential equations (C.12), they are still valid for $t \in (0,T)$. The time variation (C.13) has to be adapted with the addition of Lagrange multipliers $\boldsymbol{\nu} \in \mathbb{R}^m$

$$\mathcal{H}(\mathbf{x},\mathbf{u}^*,\mathbf{p}) + \frac{\partial\, \boldsymbol{\Psi}(\mathbf{x}(T),T)^{\mathrm{T}}\boldsymbol{\nu}}{\partial T} = 0. \tag{C.14}$$

This can be combined with the boundary term

$$\mathbf{p}(T) = \left.\frac{\partial\, \boldsymbol{\Psi}(\mathbf{x}(T),T)^{\mathrm{T}}\boldsymbol{\nu}}{\partial \mathbf{x}}\right|_{\mathbf{x}(T)}, \tag{C.15}$$

to form the *transversality conditions*[5]

$$\begin{bmatrix} -\mathcal{H}(T) \\ \mathbf{p}(T) \end{bmatrix} = D\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\nu} = \begin{bmatrix} \frac{\partial\, \boldsymbol{\Psi}(\mathbf{x}(T),T)}{\partial T} & \frac{\partial\, \boldsymbol{\Psi}(\mathbf{x}(T),T)}{\partial \mathbf{x}} \end{bmatrix}^{\mathrm{T}} \boldsymbol{\nu}. \tag{C.16}$$

When solving (C.12) starting from $\mathbf{x}_0$, the additional parameters $\boldsymbol{\nu} \in \mathbb{R}^m$ and T can be found by solving the nonlinear set of Eqs. (C.16) along with $\boldsymbol{\Psi}(\mathbf{x}(T),T) = \mathbf{0}$.

Inequality constraints. When m inequality constraints $\boldsymbol{\Phi}(\mathbf{x}(T),T) \leq \mathbf{0}$ are present at final time T rather than equalities, the equality constraints are replaced with the *Karush–Kuhn–Tucker (KKT) conditions*,

$$\begin{aligned} &\boldsymbol{\Phi}(\mathbf{x}(T),T) \leq \mathbf{0}, \\ &\boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{x}(T),T)\boldsymbol{\nu} = 0, \\ &\boldsymbol{\nu} \geq \mathbf{0}. \end{aligned} \tag{C.17}$$

These are known as primal feasibility, complementary slackness and dual feasibility conditions, respectively. Transversality conditions (C.16) still hold.

Local & global sufficiency. The state and costate equations (C.12) along with the transversality conditions (C.16) are only *necessary* conditions [26]. Necessary solutions can be strengthened with HJB to form sufficient conditions for global optimality. Solutions found with MP are (globally) sufficient if the optimal controls (C.7) satisfy HJB (C.2) for a continuously differentiable $V(\mathbf{x},t) \in \mathcal{C}^1(\mathbb{R}^n,[0,T])$, which is hard to find for free-time control problems. Sufficient conditions for local optimality, on the other hand, are easier to verify, see for example, Chapter 6 of Bryson and Ho [27] or Chapter 5 of Speyer and Jacobson [38].

[4] More generally, the solution technique is called the *method of characteristics* for hyperbolic PDEs.

[5] With slight abuse of notation, the derivatives of the boundary term $\boldsymbol{\Psi}$ are evaluated at *optimized* T and $\mathbf{x}(T)$, which is found by evolving the state equation till T.

Numerical solution by direct optimization. Numerical integration of BVP when free-parameters are present can be very difficult and/or time consuming. An alternative approach suitable for generating minimum-acceleration trajectories is given in this paper. Inserting (C.5)–(C.7) into (1), yields for our particular problem Eq. (25). The integrand of the cost functional, parameterized by the free parameters $\mathbf{a}_3, \mathbf{a}_2, T$ (or equivalently, $\mathbf{q}_f, \dot{\mathbf{q}}_f, T$ using Eq. (8)) of the momenta, is integrated from 0 to T to form the cost function (30) of the Focused Player optimization. The Defensive Player optimization (55) changes the racket equality constraints to ball landing inequalities while adding additional penalties $\phi_{\text{pen}}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T)$ to the cost function.

Local optima of the augmented cost functionals (28) and (54) can be shown to satisfy the remaining necessary conditions for optimality, namely the transversality conditions (C.16) and (C.17). Parameterizing the Hamiltonian $\mathcal{H}$ and the momenta $\mathbf{p}$ at striking time w.r.t. optimization variables $\mathbf{q}_f, \dot{\mathbf{q}}_f, T$, (C.16) and (C.17) are consequences of the necessary first order optimality conditions for Problems (30) and (55), respectively.

References

[1] M. Matsushima, T. Hashimoto, M. Takeuchi, F. Miyazaki, A learning approach to robotic table tennis, IEEE Trans. Robot. 21 (4) (2005) 767–771.

[2] Y. Huang, D. Xu, M. Tan, H. Su, Adding active learning to LWR for Ping-Pong playing robot, IEEE Trans. Control Syst. Technol. 21 (4) (2013) 1489–1494.

[3] K. Mülling, J. Kober, J. Peters, A biomimetic approach to robot table tennis, Adapt. Behav. 19 (5) (2011) 359–376 arXiv:http://adb.sagepub.com/content/19/5/359.full.pdf+html.

[4] O. Koç, G. Maeda, J. Peters, A new trajectory generation framework in robotic table tennis, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2016.

[5] J. Billingsley, Robot ping pong, Pract. Comput. (1983).

[6] R.L. Anderson, A Robot Ping-pong Player: Experiment in Real-time Intelligent Control, MIT Press, Cambridge, MA, USA, 1988.

[7] L. Acosta, J.J. Rodrigo, J.A. Mendez, G.N. Marichal, M. Sigut, Ping-pong player prototype, IEEE Robot. Autom. Mag. 10 (4) (2003) 44–52. http://dx.doi.org/10.1109/MRA.2003.1256297.

[8] H. Li, H. Wu, L. Lou, K. Khnlenz, O. Ravn, Ping-pong robotics with high-speed vision system, in: 2012 12th International Conference on Control Automation Robotics Vision, ICARCV, 2012, pp. 106–111. http://dx.doi.org/10.1109/ICARCV.2012.6485142.

[9] C.H. Lai, T.I.J. Tsay, Self-learning for a humanoid robotic ping-pong player, Adv. Robot. 25 (9–10) (2011) 1183–1208. http://dx.doi.org/10.1163/016918611X574678.

[10] Z. Yu, Y. Liu, Q. Huang, X. Chen, W. Zhang, J. Li, G. Ma, L. Meng, T. Li, W. Zhang, Design of a humanoid ping-pong player robot with redundant joints, in: 2013 IEEE International Conference on Robotics and Biomimetics, ROBIO, 2013, pp. 911–916. http://dx.doi.org/10.1109/ROBIO.2013.6739578.

[11] Y. Sun, R. Xiong, Q. Zhu, J. Wu, J. Chu, Balance motion generation for a humanoid robot playing table tennis, in: 2011 11th IEEE-RAS International Conference on Humanoid Robots, 2011, pp. 19–25. http://dx.doi.org/10.1109/Humanoids.2011.6100826.

[12] R. Xiong, Y. Sun, Q. Zhu, J. Wu, J. Chu, Impedance control and its effects on a humanoid robot playing table tennis, Int. J. Adv. Robot. Syst. 9 (5) (2012) 178. http://dx.doi.org/10.5772/51924.

[13] Y. Huang, B. Schölkopf, J. Peters, Learning optimal striking points for a ping-pong playing robot, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2015, pp. 4587–4592. http://dx.doi.org/10.1109/IROS.2015.7354030.

[14] A. Nakashima, Y. Ogawa, Y. Kobayashi, Y. Hayakawa, Modeling of rebound phenomenon of a rigid ball with friction and elastic effects, in: Proceedings of the 2010 American Control Conference, 2010, pp. 1410–1415. http://dx.doi.org/10.1109/ACC.2010.5530520.

[15] Y. Huang, D. Xu, M. Tan, H. Su, Trajectory prediction of spinning ball for ping-pong player robot, in: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011, pp. 3434–3439. http://dx.doi.org/10.1109/IROS.2011.6095044.

[16] J. Glover, L.P. Kaelbling, Tracking the spin on a ping pong ball with the quaternion bingham filter, in: IEEE Conference on Robotics and Automation, ICRA, 2014.

[17] Y. Zhang, R. Xiong, Y. Zhao, J. Wang, Real-time spin estimation of ping-pong ball using its natural brand, IEEE Trans. Instrum. Meas. 64 (8) (2015) 2280–2290. http://dx.doi.org/10.1109/TIM.2014.2385173.

[18] Y. Zhao, R. Xiong, Y. Zhang, Model based motion state estimation and trajectory prediction of spinning ball for ping-pong robots using expectation-maximization algorithm, J. Intell. Robot. Syst. 87 (3) (2017) 407–423.

[19] M. Ramanantsoa, A. Durey, Towards a stroke construction model, Int. J. Table Tennis Sci. 2 (1994) 97–114.

[20] K. Mülling, J. Kober, O. Kroemer, J. Peters, Learning to select and generalize striking movements in robot table tennis, Int. J. Robot. Res. 32 (3) (2013) 263–279. http://dx.doi.org/10.1177/0278364912472380.

[21] Y. Huang, D. Büchler, O. Koç, B. Schölkopf, J. Peters, Jointly learning trajectory generation and hitting point prediction in robot table tennis, in: 2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids), 2016, pp. 650–655. http://dx.doi.org/10.1109/HUMANOIDS.2016.7803343.

[22] B. Bäuml, O. Birbach, T. Wimböck, U. Frese, A. Dietrich, G. Hirzinger, Catching flying balls with a mobile humanoid: System overview and design considerations, in: 2011 11th IEEE-RAS International Conference on Humanoid Robots, 2011, pp. 513–520.

[23] S. Kim, E. Gribovskaya, A. Billard, Learning motion dynamics to catch a moving object, in: 2010 10th IEEE-RAS International Conference on Humanoid Robots, 2010, pp. 106–111.

[24] J. Nakanishi, A. Radulescu, D.J. Braun, S. Vijayakumar, Spatio-temporal stiffness optimization with switching dynamics, Auton. Robots (2016) 1–19.

[25] D. Liberzon, Calculus of Variations and Optimal Control Theory: A Concise Introduction, Princeton University Press, Princeton, NJ, USA, 2011.

[26] L.S. Pontryagin, V.G. Boltyanskii, R.V. Gamkrelidze, E.F. Mishchenko, The Mathematical Theory of Optimal Processes, Interscience, 1962 (English translation).

[27] A.E. Bryson, Y.-C. Ho, Applied Optimal Control : Optimization, Estimation, and Control, Rev. printing., Hemisphere Pub. Corp.; distributed by Halsted Press Washington, New York, 1975, p. 481.

[28] H. Schättler, U. Ledzewicz, Geometric optimal control : theory, methods and examples, in: Interdisciplinary Applied Mathematics, Springer, New York, Heidelberg, Dordrecht, 2012.

[29] H.W. Sorenson, Kalman Filtering: Theory and Application, IEEE Press, Los Alamitos, CA, 1985.

[30] M.W. Spong, S. Hutchinson, M. Vidyasagar, Robot Modeling and Control, John Wiley & Sons, Hoboken (N.J.), 2006.

[31] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, J. Peters, Trajectory planning for optimal robot catching in real-time, in: 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 3719–3726.

[32] S. Schaal, The SL Simulation and Real-Time Control Software Package, University of Southern California, 2006.

[33] S.G. Johnson, The nlopt nonlinear-optimization package, 2016, http://ab-initio.mit.edu/nlopt.

[34] M.J.D. Powell, A direct search optimization method that models the objective and constraint functions by linear interpolation, in: S. Gomez, J.-P. Hennart (Eds.), Advances in Optimization and Numerical Analysis, Springer Netherlands, Dordrecht, 1994, pp. 51–67.

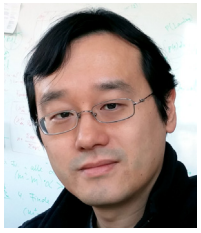
[35] C.H. Lampert, J. Peters, Real-time detection of colored objects in multiple camera streams with off-the-shelf hardware components, J. Real-Time Image Process. 7 (1) (2012) 31–41.

[36] R.S. Sutton, A.G. Barto, Introduction to Reinforcement Learning, first ed., MIT Press, Cambridge, MA, USA, 1998.

[37] R.H. Shumway, D.S. Stoffer, An approach to time series smoothing and forecasting using the em algorithm, J. Time Series Anal. 3 (4) (1982) 253–264.

[38] J.L. Speyer, D.H. Jacobson, Primer on Optimal Control Theory, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2010.

Okan Koç received his B.S. degrees in Electrical Engineering and Mathematics from Boğaziçi University, Turkey in 2009. He received his M.S. degree in 2013 in Applied Mathematics from ETH Zürich. He is currently a Ph.D. candidate in Intelligent Autonomous Systems at Technische Universität Darmstadt and working in the Max Planck Institute for Intelligent Systems, Empirical Inference department in Tüebingen. His research interests are in the areas of optimal control and learning control.



Guilherme Maeda is a research scientist at ATR Computational Neuroscience Laboratories in the Department of Brain Robot Interface, Kyoto, Japan. From 2013 to 2017 he was a team leader at the Intelligent Autonomous Systems group (IAS) in TU Darmstadt. He has a Ph.D. in robotics from the Australian Centre for Field Robotics and a masters in control engineering from the Tokyo Institute of Technology. His research interests are in robotics, specifically in control, learning methods for control, and their applications to physical human–robot interaction and collaboration.

Jan Peters is a full professor (W3) for Intelligent Autonomous Systems at the Computer Science Department of the Technische Universität Darmstadt and at the same time an adjunct senior research scientist at the Max-Planck Institute for Intelligent Systems, where he heads the interdepartmental Robot Learning Group between the departments of Empirical Inference and Autonomous Motion. Jan Peters has received a few awards, most notably, he has received the Dick Volz Best 2007 US Ph.D. Thesis Runner Up Award, the 2012 Robotics: Science & Systems—Early Career Spotlight, the 2013 IEEE Robotics & Automation Society's Early Career Award, and the 2013 INNS Young Investigator Award.