Towards Table Tennis with a Quadrotor Autonomous Learning Robot and Onboard Vision*

Rui Silva[1] and Francisco S. Melo[1] and Manuela Veloso[2]

***Abstract*— Robot table tennis is a challenging domain in both robotics, artificial intelligence and machine learning. In terms of robotics, it requires fast and reliable perception and control; in terms of artificial intelligence, it requires fast decision making to determine the best motion to hit the ball; in terms of machine learning, it requires the ability to accurately estimate where and when the ball will be so that it can be hit. The use of sophisticated perception (relying, for example, in multi-camera vision systems) and state-of-the-art robot manipulators significantly alleviates concerns with perception and control, leaving room for the exploration of novel approaches that focus on estimating where, when and how to hit the ball. In this paper, we move away from the hardware setup commonly used in this domain—typically relying on robotic manipulators combined with an array of multiple fixed cameras—and give the first steps towards having autonomous aerial table tennis robotic players. Specifically, we focus on the task of hitting a ping pong ball thrown at a commercial drone, equipped with a light cardboard racket and an onboard camera. We adopt a general framework for learning complex robot tasks and show that, in spite of the perceptual and actuation limitations of our system, the overall approach enables the quadrotor system to successfully respond to balls served by a human user.**

I. Introduction

In this paper, we address the problem of intercepting a target object moving in 3D space using an autonomous drone. We are interested in a particular instantiation of this general problem where the moving target is a table tennis ball thrown at the drone by a human user. The drone is a commercial quadrotor (the Parrot AR Drone), and the only information about the target and the drone itself is provided by basic sensing available onboard (including an onboard camera).

Our domain is inspired by robot table tennis, an appealing domain for robot research, since a robot that is able to play table tennis with some level of competence must successfully address issues of

- *(P) Perception*, *i.e.*, reliably *detect* the position and velocity of the tennis ball as it moves towards the robot;
- *(E) Estimation/decision*, *i.e.*, reliably *predict* the position in which the robot will hit the ball (hitting point) and *select* the best motion to hit it (hitting motion);
- *(C) Control*, *i.e.*, reliably *execute* the hitting motion.

*This work was partially supported by national funds through FCT—Fundação para a Ciência e a Tecnologia, with reference UID/CEC/50021/2013, and a scholarship with the same reference.

[1]R. Silva and F.S. Melo are with INESC-ID and with Instituto Superior Técnico, University of Lisbon, Porto Salvo, Portugal. E-mail: rui.teixeira.silva@ist.utl.pt, fmelo@inesc-id.pt

[2]M. Veloso is with the School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA. E-mail: veloso@cs.cmu.edu

All above issues (perception, estimation and control) must be addressed in *real-time*, to ensure that the robot reaches the desired hitting point before the ball does.

Our work is related with recent work on drone applications, including cooperative load carrying [1] and acrobatic object manipulation [2], [3]. In terms of the three aspects discussed above, all aforementioned works rely on motion capture systems or similarly sophisticated perception modules, providing real-time robust perception. Estimation relies on analytical dynamical models. Control is computed from trajectories generated on-the-fly. Unlike these works, our perception relies solely on onboard sensing. Additionally, we adopt a data-driven/model-free approach to estimate the hitting point. Finally, we adopt a learning approach to construct on-the-fly the hitting motion from a library of parameterizable hitting motions.

Another related area of research is robot soccer, where the robots must approach/intercept the moving ball based only on visual information collected from the onboard cameras [4], [5]. There are, however, important differences in terms of the three dimensions identified above. In terms of perception, the existence of teammates and opponents renders perception more challenging—since the ball may often be occluded or follow an unpredictable trajectory, also rendering estimation quite challenging. On the other hand, both robots and the ball essentially move in a 2D world, which is also quite distinct from the 3D motion featured in our table tennis setting.

Finally, there is a significant volume of work on robot table tennis. Early approaches to robot table tennis already exhibited some level of success [6]–[10], in spite of the limited hardware and computational power. Most approaches focused on the design of real-time robust perception and control (issues (P) and (C) above), and relied heavily on human knowledge in the form of explicit models of the task. Powerful hardware and software shifted the focus of robot table tennis research to issues of *generalization* and *adaptation*. In terms of the issues previously identified, machine learning techniques have been used mainly to address the problem of hitting point estimation and hitting motion selection, corresponding to issue (E) above.

In this paper, we address the problem of robotic table tennis using a quadrotor drone (the Parrot AR Drone 2.0, see Fig. 1a) equipped with a light cardboard racket (see Fig. 1b) and an onboard camera. The robot is controlled via wireless from an off-board commodity computer that processes the camera feed and sends back the control commands to the drone. The purpose of the racket is not to bounce back the ball but merely to intercept it. The main distinction

(a) The Parrot AR Drone 2.0. (b) Cardboard racket used. (c) Assembled set in flight. (d) Simulator screenshot.

Fig. 1. Setup used in this paper (real and simulated).

between our work and existing work on drone applications and robot table tennis is, perhaps, in terms of perception. This difference translates in two key challenges. First, ball tracking must rely exclusively on the onboard camera and not on external motion caption systems or arrays of external fixed cameras. Second, robot positioning relies only on proprioception and, as such, is used minimally.

We adopt a general framework for complex robot tasks that combines two powerful learning paradigms (imitation learning and reinforcement learning), allowing initial knowledge to be fed into the system from demonstration and allowing the system to improve its performance by trial and error [11], [12]. We adapt this framework to the constraints imposed by our robotic setup (namely, the use of a single onboard camera as the only sensor, and the much slower response of the robot) and construct a library of hitting motions that are combined to enable the robot to successfully intercept tennis balls thrown at it by a human user. Our results show, both in simulation and in experiments with the real quadrocopter, that the overall approach is able to accommodate the perceptual and actuation limitations of the robot, while requiring minimum domain knowledge to be explicitly provided to the system.

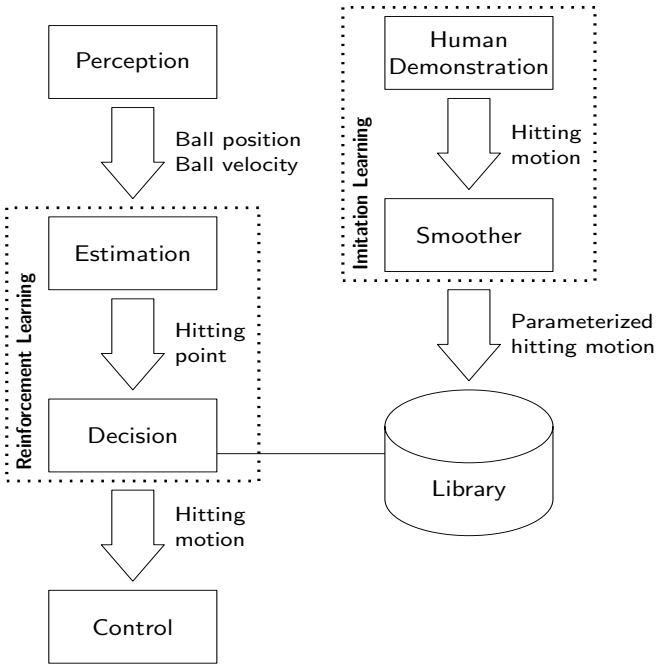


Fig. 2. Overview of the framework used in the paper.

II. Overview of the Approach

In this section we describe the robotic platform used, and provide an overview of the approach used.

A. The Robot

We use a commercial quadrotor drone, the Parrot AR Drone 2.0 (see Fig. 1a). Among other sensors, the drone is equipped with an HD camera (resolution of 720px at 30 fps), a gyroscope, an accelerometer, a magnetometer, an ultrasound altimeter and flight stabilizers. The drone offers the capability of remote programming, by creating a dedicated Wi-Fi to which other devices can connect. There is also a ROS driver for the robot and a model for Gazebo. Using this model, we created a ROS-based realistic simulator for the robot table tennis domain (Fig. 1d) that is used both for allowing human users to provide demonstration data and for initial trial and error learning to take place.

We attached a light card-board racket with the dimensions of an official ITTF racket to the drone (Figs. 1b and 1c). The material selected offers the advantage of being very light and, as such, does not interfere significantly with the flight of the drone. It also provides a convenient "target surface" to intercept the tennis ball, diminishing the possibility of the latter contacting with the rotors. We note, however, that the restitution coefficient of the racket is not sufficient for the ball to bounce. While this is sufficient for our purposes, a different racket would be necessary for the system to be "tennis-table-ready".

B. The Approach

We adopt the general framework described in Fig. 2. This general framework has several appealing features as a general framework for learning complex robotic tasks. First, the use of RL enables the robot to *adapt* to the task at hand, improving its performance while requiring little domain knowledge to be explicitly encoded thereinto. Second, the combination of RL with imitation learning enables human users to provide (implicit) domain knowledge upon which the RL algorithm can then improve. Thirdly, such domain knowledge can be provided by experts on the task at hand,

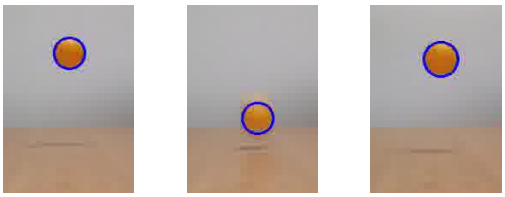


Fig. 3. Real-time detection of a bouncing ball.

even if they have little expertise in robotics [13].

Different instantiations of the above framework that combine RL with imitation learning have been used in different domains. For example, Abbeel et al. use a similar approach to "program" an RC helicopter to perform complex maneuvers [11]. Imitation learning relies on *inverse reinforcement learning*, while the RL module uses differential dynamic programming [11]. Muelling et al. adopt this same framework in a robot table tennis system that is capable of returning balls with a 94% success rate [12].

Our instantiation of the framework in Fig. 2 builds on that of Muelling et al. [12]. We perform hitting point estimation using a variation of regularized kernel regression, dubbed *cost-regularized kernel regression* (CrKR), in which samples are weighted according to their relevance for the task at hand [14].[1] Once the hitting point is estimated, the decision module determines the hitting motion from a set of parameterized hitting motions learned from human demonstrations. These "motion primitives" are combined into one single motion that depends on the situation (ball position and velocity, robot pose, etc.) and on their estimated success. The weights are adjusted using reinforcement learning.

III. Tracking a Tennis Ball with a Moving Camera

As discussed before, perception in our system relies essentially on a single onboard camera. In particular, the camera feed is used to estimate the 3D ball position and velocity which, in turn, are used to determine the hitting point.

In order to recover the 3D position and velocity of the ball from the single camera image, we adopt a simple approach that ensures a relatively robust estimate in real-time. Related approaches have been used in other works requiring fast and accurate tracking, most notably in the Robocup competition [15], [16]. In our approach, we use simple HSV color segmentation to identify the table tennis ball in each image captured by the onboard camera. We fit the parameters of a circular boundary using regularized least-squares, obtaining an estimate of the ball position in the image [17].[2] Figure 3 illustrates the real-time ball detection in a sequence of images.

We now use the diameter of the ball in the image, h_{image}, and its position in the image, (x, y), to estimate its 3D position and velocity. Both the position (X, Y, Z) and the velocity $(\dot{X}, \dot{Y}, \dot{Z})$ of the ball are computed with respect to the robot's reference frame (see Fig. 6 for an illustration). The selected reference frame matches that of the camera, which avoids transformations between coordinate frames and, as will soon become apparent, facilitates the integration of estimation, decision and control.

To estimate the 3D position of the ball from h_{image}, and its position in the image, (x, y), we adopt a simple pinhole model for the camera. The distance d between the ball and the focal-point of the camera can be computed from the projective relation

$$d = \frac{f}{m} \cdot \frac{h_{\text{world}}}{h_{\text{image}}}, \tag{1}$$

where f is the focal length of the camera, m is the pixel size, and h_{world} represents the diameter of the ball in the world. The parameters f and m are easily obtained from the camera calibration and table tennis balls have a standard diameter of 4cm, thus implying $h_{\text{world}} = 4\text{cm}$. Once d is computed, and given the image coordinates x and y of the ball, we can solve for X, Y and Z by solving the following system of equations

$$x = \frac{f}{m_x}\frac{X}{Z} + \frac{p_x}{m_x}, \tag{2a}$$

$$y = \frac{f}{m_y}\frac{Y}{Z} + \frac{p_y}{m_y}, \tag{2b}$$

$$d = \sqrt{X^2 + Y^2 + Z^2}. \tag{2c}$$

In (2), m_x and m_y represent the horizontal and vertical pixel size, and p_x and p_y represent the coordinates of the image center. All these parameters are also estimated from camera calibration. Finally, to estimate the ball velocity in consecutive frames, we compute the ratio between the distance traversed by the ball between the two frames, and the time elapsed.

To evaluate our ball pose estimation approach, we use the simulator to track the ball along a complete trajectory, measuring the average error in both pose and velocity against the simulator ground-truth. The results are summarized in Table I and Fig. 4.

TABLE I

Average estimation error per trajectory. The position estimate errors are expressed in cm, while the velocity estimation errors are expressed in cm/sec. Results are averages over 20 random independent trajectories.

	X	Y	Z	$\dot{X}$	$\dot{Y}$	$\dot{Z}$
Error	0.77	4.96	24.15	17.07	119.0	478.0

Observing the results in Table I, we note that the error in the Z coordinate (depth) is significantly superior to that in X and Y (by one order of magnitude or more). This difference in the order of magnitude of the error can be explained by the fact that, as the distance between the ball and the robot increases, so does the impact of a single pixel in the radius

[1] In the specific case of table tennis, the samples are weighted by how successful the corresponding hitting motion was.

[2] In our system, we used the OpenCV implementation of the aforementioned method.

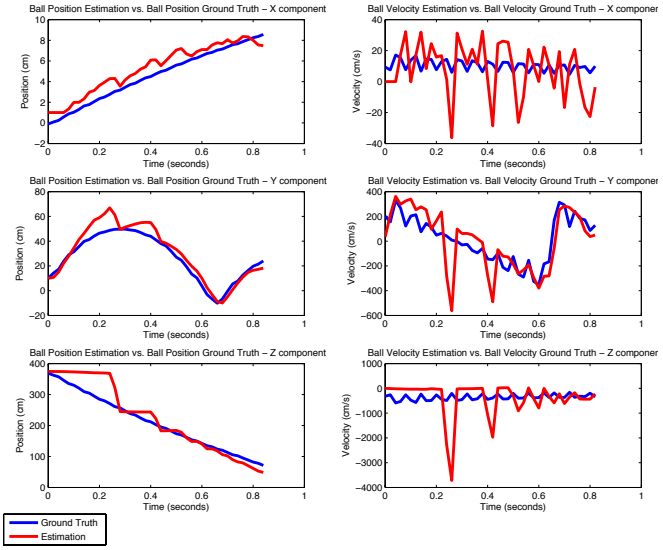


Fig. 4. Estimation error in both ball position and velocity for one trajectory.

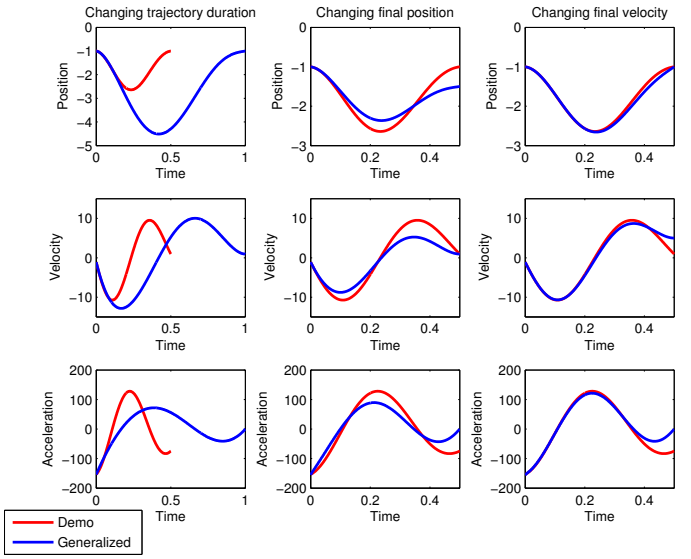


Fig. 5. Example of a DMP. By modifying the dynamics of the canonical system and changing the value of $\mathbf{s}_{\text{ref}}$ it is possible to execute a similar trajectory but in a different time-frame or reaching a different end-point.

of the ball in the image in the distance estimate. This impact is also observed in the plots in Fig. 4: the steps observed in the Z-coordinate estimate arise precisely from an increase in the number of pixels of the ball diameter in the image.

Additionally, we note also that the noise in the position estimates is greatly amplified in the velocity estimates, mainly due to the small time between frames.

Nevertheless, our approach is able to provide relatively accurate position estimates in real-time, while requiring no dynamic model of the ball or the robot. More robust estimates could be obtained by using, for example, an Extended Kalman Filter. However, this would require a dynamic model of the ball in the robot's coordinate frame that could take into account the motion of the latter, and it is not clear that the computational burden associated with such approach would prove advantageous in terms of performance.

IV. Intercepting a Tennis Ball with a Quadrotor

This section describes the imitation and reinforcement learning modules of our architecture.

A. Imitation Learning

In order for the robot to be able to successfully hit the table tennis ball, we initially build a *library of "hitting motions"* from which the robot can determine the best hitting motion to take for any given ball. Such hitting motions are obtained from a human expert piloting the robot, and parameterized in the form of *dynamic motor primitives* [12]. The use of DMP instead of other alternative representations is due to the fact that they can be efficiently computed and reparameterized to obtain similarly-shaped trajectories to any desired goal-point.

1) Dynamic Motor Primitives: A dynamic motor primitive (DMP) is a *dynamical system* with the general form

$$\dot{z}(t) = h(z(t)), \tag{3a}$$

$$\dot{\mathbf{s}}(t) = f(z(t), \mathbf{s}(t), \mathbf{s}_{\text{ref}}, \boldsymbol{\theta}). \tag{3b}$$

Equation (3a) is known as the *canonical system* and (3b) as the *transformed system*. We refer to z as the *phase variable* and to $\mathbf{s}$ as the state of the transformed system;[3] $\mathbf{s}_{\text{ref}}$ is the *reference state* which roughly corresponds to the "goal state" for the trajectory, and $\boldsymbol{\theta}$ are the DMP parameters.

Given some observed trajectory that we wish to model, $\{\mathbf{s}_{\text{demo}}(t), t = t_i, \ldots, t_f\}$, the parameter vector $\boldsymbol{\theta}$ is selected so that the state $\mathbf{s}(t)$ of the transformed system follows $\mathbf{s}_{\text{demo}}(t)$ as closely as possible. Once $\boldsymbol{\theta}$ is determined, it is possible to "drive" the DMP to execute a similar motion but to a different ending point by altering $\mathbf{s}_{\text{ref}}$. Additionally, the dynamics of the phase variable z do not depend on the state $\mathbf{s}$ of the DMP, but the dynamics of $\mathbf{s}$ do depend on z. Hence, it is also possible to "control" the pace at which the DMP trajectory is executed by modifying the dynamics of z. Figure 5 depicts an example of a DMP and how it can easily be modified to reach a different final state or take a different amount of time to execute.

In our implementation, the transformed system represents a body pulled by a "moving target". The state $\mathbf{g}(t)$ of the moving target is $\mathbf{g}(t) = [g(t), \dot{g}(t)]$ and the canonical and transformed systems are given by

$$\begin{aligned} \tau\dot{z}(t) &= -\alpha_z z(t) \\ \tau^2\ddot{s}(t) &= \alpha_s(\beta_s(g(t) - s(t)) + \tau(\dot{g}(t) - \dot{s}(t))) \\ &\quad + \tau^2\ddot{g}(t) + \eta u(t), \end{aligned} \tag{4}$$

where $u(t)$ is a *reference signal*, τ is a temporal scaling factor, and η, α_z, α_s and β_s are tunable constants.[4] By

[3] We represent the state of the transformed system as a vector $\mathbf{s}(t)$ since it usually comprises a "position" component, $s(t)$, and a "velocity" component, $\dot{s}(t)$. In other words, we are typically interested in situations where $\mathbf{s}(t) = [s(t), \dot{s}(t)]^\top$ for some quantity of interest, $s(t)$.

[4] We refer to [18] for additional discussion on these constants.

construction, $g(t)$ is described as a 5th order polynomial

$$g(t) = \sum_{i=0}^{5} b_i t^i, \tag{5}$$

whose coefficients $b_i, i = 0, \ldots, 5$, are computed to match the initial and final states of the desired trajectory.

2) Hitting Motions as DMPs: In the robot table tennis domain, DMPs are used to represent the trajectories of the robot. For ease of notation, we represent the robot state at time t as a vector $\mathbf{S}(t) = [X(t), Y(t), Z(t), \dot{X}(t), \dot{Y}(t), \dot{Z}(t)]^\top$. We write S_i to refer to any one of the coordinates X, Y or Z and $\dot{S}_i$ to refer to its temporal derivative. Each state coordinate S_i is represented using a transformed system, with all three transformed systems sharing the same canonical system. At runtime, by selecting the desired final position and velocity for each coordinate, we are able to send the robot to "any" position of interest (namely, to the hitting point) within "any" time-frame, by setting the phase variable and the reference state adequately and executing the DMP—*i.e.*, simulating the corresponding dynamical system and using the observed values for $\dot{X}$, $\dot{Y}$ and $\dot{Z}$ to directly control the robot.[5]

To construct a DMP from a trajectory performed by a human expert pilot, we track the state $\mathbf{S}$ of the robot throughout the complete hitting motion. Given the observed trajectory $\{\mathbf{S}_{\text{demo}}(t), t = t_i, \ldots, t_f\}$, we use (5) to compute, for each coordinate S_i, the corresponding "moving target" trajectory, $\{g_i(t), t = t_i, \ldots, t_f\}$. From $\{g_i(t), t = t_i, \ldots, t_f\}$ we compute the associated reference signal, $\{u_i(t), t = t_i, \ldots, t_f\}$ according to (4). Finally, once the computation of $\{u_i(t), t = t_i, \ldots, t_f\}$ is complete, we can pair the reference signal and the phase variable values to build a dataset $\mathcal{D}_i = \{(z(t), u_i(t)), t = t_i, \ldots, t_f\}$ and use, for example, locally weighted regression to estimate the parameters $\boldsymbol{\theta}_i$ that minimize the weighted squared error of the estimator $\hat{u}_i(z) = \boldsymbol{\theta}_i^\top \boldsymbol{\psi}(z) \cdot z$ with respect to the data in $\mathcal{D}_i$. This process is repeated for each coordinate S_i.

B. Hitting Point Estimation

The trajectories provided by the human expert, corresponding to successful hitting motions, are used not only to build the DMP library but also a *dataset* from which the hitting point estimator can be derived. Both data collection and DMP construction are conducted using our simulator.

When a ball is launched towards the robot, the latter keeps track of the position/velocity of the ball at each frame. As soon as the ball crosses the 2.5m "plane", its position $\mathbf{s}_{\text{ball}}$ and velocity $\dot{\mathbf{s}}_{\text{ball}}$ are recorded (Fig. 6). The instant t_i at which such crossing takes place is also recorded as the initial instant of the trajectory.

From the instant t_i onwards, and throughout the execution of the hitting motion by the human expert, the trajectory of the drone is tracked using proprioception. The trajectory is considered complete at the instant t_f when the robot hits the ball, and the position and velocity of the robot at that instant, $\mathbf{s}_{\text{robot}}$ and $\dot{\mathbf{s}}_{\text{robot}}$, is recorded as the hitting point. A new datapoint $(\mathbf{s}, \boldsymbol{\delta})$ is added to the dataset, where $\mathbf{s} = (\mathbf{s}_{\text{ball}}, \dot{\mathbf{s}}_{\text{ball}})$ and $\boldsymbol{\delta} = (\mathbf{s}_{\text{robot}}, \dot{\mathbf{s}}_{\text{robot}}, t_f - t_i)$.

[5]The robot is controlled by providing it velocity commands.

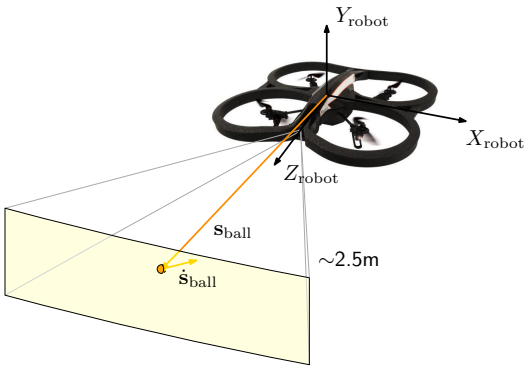


Fig. 6. The instant the ball-robot distance drops below 2.5m is used as the initial instant of the corresponding DMP. Additionally, the position and velocity of the ball at that instant are used to estimate the hitting point.

Given the dataset $\{(\mathbf{s}_n, \boldsymbol{\delta}_n), n = 1, \ldots, N\}$ constructed above, we use a variation of *regularized kernel regression* [19] to build an estimator $\hat{\boldsymbol{\delta}}(\mathbf{s})$ with ith component

$$\hat{\delta}_i(\mathbf{s}) = \mathbf{k}(\mathbf{s})^\top (\mathbf{K} + \lambda \mathbf{C})^{-1} \Delta_i. \tag{6}$$

We use a Gaussian kernel $k(\mathbf{s}, \mathbf{s}')$ and write $\mathbf{k}(\mathbf{s})$ to denote the vector with nth component $k(\mathbf{s}, \mathbf{s}_n)$. Likewise, we write $\mathbf{K}$ to denote the matrix with mn element given by $k(\mathbf{s}_m, \mathbf{s}_n)$. Finally, Δ_i is a N-dimensional column-vector in which the nth component corresponds to the ith element of $\boldsymbol{\delta}_n$.

To describe in greater detail the particular variation of regularized kernel regression used, we recall that each sample point in our dataset is derived from a single hitting motion. The approach used, known as *Cost-regularized Kernel Regression*, or CrKR, uses the success (or lack thereof) of the hitting motion to weight the different samples in the dataset [14]. In particular, we associate with each sample point $(\mathbf{s}_n, \boldsymbol{\delta}_n)$ a cost c_n that is inversely proportional to how far the ball was hit into the table. Then, in (6), $\mathbf{C}$ is a diagonal matrix whose nth nonzero entry takes the value c_n.

C. Hitting Motion Selection

The robot is provided with a library of hitting motions obtained from demonstration by a human expert and encoded as DMPs. Let Π represent such library, and let $\pi_k, k = 1, \ldots, K$, denote the individual DMPs in Π. Roughly speaking, each DMP π_k maps a hitting point $\boldsymbol{\delta}$ into a particular trajectory. The components $\mathbf{s}_{\text{robot}}$ and $\dot{\mathbf{s}}_{\text{robot}}$ of $\boldsymbol{\delta}$ provide the reference state for the DMP, while the component $t_f - t_i$ is used to adjust the phase variable of the DMP.

At runtime, after detecting that the ball has crossed the 2.5m "plane", the robot determines the state $\mathbf{s}$ of the ball and estimates the corresponding hitting point, $\hat{\boldsymbol{\delta}}(\mathbf{s})$. It then computes the hitting motion as a *mixture* of the DMPs in Π. The weight assigned to each DMP π_k generally depends

both on $\mathbf{s}$ and $\hat{\boldsymbol{\delta}}(\mathbf{s})$. The resulting hitting motion, $\hat{\pi}$, is

$$\hat{\pi}(\boldsymbol{\delta}) = \sum_{k=1}^{K} \frac{\gamma_k \pi_k(\boldsymbol{\delta})}{\sum_{k'=1}^{K} \gamma_{k'}},$$

where γ_k is the weight assigned to the DMP π_k. The motion $\hat{\pi}$ is executed and evaluated according to its success in sending the ball towards the opponent's court.

The evaluation of each successful hitting motion is provided in the form of a *reward* r that is used to evaluate the individual success of each motor primitive and adjust the corresponding weight. In particular, given a pair $(\mathbf{s}, \boldsymbol{\delta})$, the *value* $V^{\pi_k}(\mathbf{s}, \boldsymbol{\delta})$ of the DMP π_k is defined as the expected reward received when π_k is executed at $(\mathbf{s}, \boldsymbol{\delta})$, with weight

$$\gamma_k(\mathbf{s}, \boldsymbol{\delta}) = \exp\{V^{\pi_k}(\mathbf{s}, \boldsymbol{\delta})\}.$$

Everytime the robot executes a successful hitting motion, the value of V^{π_k} is updated to reflect the most recent value of r observed. We represent V^{π_k} in the form

$$V^{\pi_k}(\mathbf{s}, \boldsymbol{\delta}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{s}, \boldsymbol{\delta})$$

and compute the parameter vector $\mathbf{w}$ using standard Bayesian linear regression [19].

D. Control

We conclude our description of our adopted framework by discussing the control module. The AR Drone 2.0 includes flight stabilizers and automatically compensates for gravity. As such, the control module needs only to provide, at each time instant t, a command $(\dot{X}, \dot{Y}, \dot{Z})$ corresponding to the desired velocity for the robot at t. Therefore, once a trajectory is encoded in the form of a DMP, the execution of the latter directly provides the control commands necessary to execute that same trajectory in the robot.

TABLE II

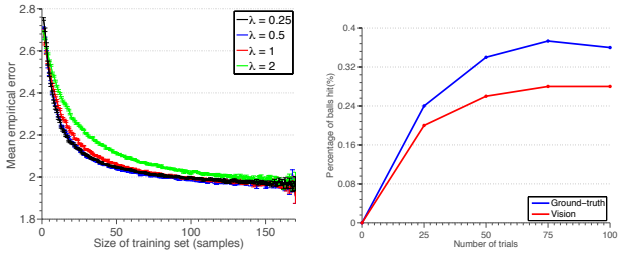
AVERAGE DMP EXECUTION ERROR. THE POSITION ERRORS ARE EXPRESSED IN CM, WHILE THE VELOCITY ESTIMATION ERRORS ARE EXPRESSED IN CM/SEC. RESULTS ARE AVERAGES OVER 10 RANDOM INDEPENDENT RUNS.

	X	Y	Z	$\dot{X}$	$\dot{Y}$	$\dot{Z}$
Error	0.40	3.82	6.77	0.89	12.58	26.88

To evaluate the accuracy of the control approach described, we measured the average DMP execution error by comparing the trajectory described by the robot when executing a DMP and the "original" ideal trajectory. The results are summarized in Table II.

Observing the results in Table II, we note that the error generally remains around 5cm.[6] An error of this order of magnitude is quite manageable, in light of the dimensions of the racket. Although closed-loop control could be deployed using, for example, the approach of [20], it would require accurate sensing with respect to the robot's pose and, much like the perception module, it is not clear that the computational burden associated with such approach would prove advantageous in terms of performance.

[6]The results presented correspond to a trajectory where the displacement in X was small, hence the small error in X.



(a) CrKR performance. (b) Hitting performance

Fig. 7. Overal performance evaluation of the framework for robot table tennis. In (b) the red line corresponds to the results with the vision system, while the blue line corresponds to the results obtained using ground-truth values for $\mathbf{s}_{\text{ball}}$ and $\dot{\mathbf{s}}_{\text{ball}}$.

V. EVALUATION

To evaluate the performance of the overall system, we analyze its ability to predict the hitting point and the ability to hit the ball. We use the robot table tennis simulator to build an initial library of 10 DMPs from an expert human pilot. As described in the previous section, this initial set of DMPs also provides the data used to construct the initial CrKR hitting point estimator.

From this initial condition, balls are randomly fired towards the robot. The robot determines the adequate hitting motion using the mixture of DMPs described in the previous section and executes the resulting trajectory. For every successful hit, the mixture weights are adequately updated, and the observed pair $(\mathbf{s}, \boldsymbol{\delta})$ is added to the CrKR dataset, enabling the CrKR estimator to also improve its estimates as the robot plays more games.[7]

To analyze the performance of CrKR, we used a dataset resulting from a total of 170 successful hits, and measured how the estimation error varied with the number of samples and as a function of the parameter λ (see (6)). The results are depicted in Fig. 7a, and illustrate a quick initial improvement in the estimation accuracy followed by a plateau. This plateau is due mainly to the large errors in the ball velocity estimates, as was observed in Table I.

We analyze the ability of the robot to hit the ball by evaluating the percentage of hits in 100 balls fired at the robot. To better assess the impact of the inaccurate vision in the performance of the system, we compare the results obtained with the vision system with those obtained using ground-truth values for $\mathbf{s}_{\text{ball}}$ and $\dot{\mathbf{s}}_{\text{ball}}$. The results are summarized in Fig. 7b.[8] While the impact of the vision errors is quite evident from the plot, both plots evidence a similar plateau effect to that observed in the CrKR performance. The plateau effect appears sooner when using the vision system, but is also visible when using ground-truth values for $\mathbf{s}_{\text{ball}}$ and

[7]The parameters used in our deployment are provided in the supplementary material.

[8]See also the supplementary video material.

$\dot{\mathbf{s}}_{\mathrm{ball}}$. This may be partly due to the actuation limitations of the robot, since its velocity does not allow it to reach balls passing somewhat far from the robot. It was also observed that several missed balls were "almost-hits". Such almost-hits may be due to small delays in the the computations.

Finally, we ported the setup deployed in the simulator to the real robot and tested its performance against balls fired by a human user. No adjustments were made to the parameters used in simulation. In a first test, we evaluated the individual performance of each of 4 motor primitives against balls fired by a human user. For each motor primitive, the human user was asked to launch a total of 20 balls towards the robot with an approximately constant direction and velocity. Before any learning, each motor primitive averaged a hitting rate of approximately 20%. After the first successful hits, the hitting rate increased to 32.5%.

In a second test, we evaluated the performance of the system with a combined set of 10 DMPs (the same used in the simulation). A human user was asked to again throw the ball towards the robot, slightly changing in each throw the direction and initial velocity of the ball. Before any training, the performance of the system system was around 10%. However, after 80 successful hits, the value increased to approximately 20% hitting rate.

VI. Conclusions and Future Work

In this paper, we addressed the problem of intercepting a target object moving in 3D space using an autonomous drone and onboard camera, with a direct application to robotic table tennis. Such limited perception poses two important challenges for the task at hand: ball tracking must rely exclusively on the onboard camera and not on external motion caption systems or arrays of external fixed cameras; additionally, robot positioning must rely only on proprioception and, as such, is used minimally.

The tests established the success of the approach, having reached hitting rates of 30% and 20% in simulation and with the real quadrotor, respectively. Although the performance of the system as a whole is below that of manipulator-based systems, our experiments suggest that, after learning, most balls missed were due to the relatively slow response of the robot—causing it to reach the hitting point slightly late, in what can be considered a "near-hit". For this reason, we believe that the overall performance of our system is satisfactory, given the sensorial and actuation limitations of the robot system used.

While our results are satisfactory, they also open the door to several additional avenues for future work. On one hand, and although a significant improvement in performance is not expectable, it would be important to investigate the tradeoff between computational effort vs performance involved in including more sophisticated perception and control (*e.g.*, by the use of an EKF for ball tracking a closed loop controller). Additionally, we would like to develop image-based visual servoing (IBVS) to return the quadcopter to an initial position, at the top of the table. This would allow the robot to respond to several balls in a row, something which is currently not possible. We would also like to develop a reachability analysis of the missed balls, in order to better understand the "near-hits". However, this would require us to use additional equipment—a precise multi-camera vision system to track both the quadrotor and the ball. Finally, it would be interesting to explore the conjugation of IBVS with DMPs, by representing trajectories in terms of image features, thus eliminating the need of estimating the ball pose.

References

[1] D. Mellinger, M. Shomin, N. Michael, and V. Kumar, "Cooperative grasping and transport using multiple quadrotors," in *Proc. 10th Int. Symp. Distributed Autonomous Robotic Systems*, no. 545–558, 2013.

[2] M. Müller, S. Lupashin, and R. D'Andrea, "Quadrocopter ball juggling," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2011, pp. 5113–5120.

[3] D. Brescianini, M. Hehn, and R. D'Andrea, "Quadrocopter pole acrobatics," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2013, pp. 3472–3479.

[4] M. Veloso, E. Winner, S. Lenser, J. Bruce, and T. Balch, "Vision-servoed localization and behavior-based planning for an autonomous quadruped legged robot," in *Proc. 5th Int. Conf. Artificial Intelligence Planning Systems*, 2000, pp. 387–394.

[5] S. Lenser, J. Bruce, and M. Veloso, "CMPack: A complete software system for autonomous legged soccer robots," in *Proc. 5th Int. Conf. Autonomous Agents*, 2001, pp. 204–211.

[6] J. Knight and D. Lowery, "Pingpong-playing robot controlled by a microcomputer," *Microprocessors and Microsystems*, vol. 10, no. 6, pp. 332–335, 1986.

[7] J. Hartley, "Toshiba progresses towards sensory control in real-time," *The Industrial Robot*, vol. 14, no. 1, pp. 50–52, 1987.

[8] H. Hashimoto, F. Ozaki, K. Asano, and K. Osuka, "Development of a ping pong robot system using 7 degrees of freedom direct drive," in *Proc. IECON 87: Industrial Applications of Robotics and Machine Vision*, 1987, pp. 608–615.

[9] R. Andersson, *A Robot Ping-Pong Player: Experiment in Real-Time Intelligent Control*. MIT Press, 1988.

[10] H. Fässler, A. Beyer, and T. Wen, "A robot ping pong player: Optimized mechanics, high performance 3D vision, and intelligent sensor control," *Robotersysteme*, vol. 6, no. 3, pp. 161–170, 1990.

[11] P. Abbeel, A. Coates, and A. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[12] K. Mülling, J. Kober, O. Krömer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *Int. J. Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013.

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

[14] J. Kober, A. Wilhelm, E. Oztop, and J. Peters, "Reinforcement learning to adjust parametrized motor primitives to new situations," *Autonomous Robots*, vol. 33, no. 4, pp. 361–379, 2012.

[15] O. Birbach, J. Kurlbaum, T. Laue, and U. Frese, "Tracking of ball trajectories with a free moving camera-inertial sensor," in *RoboCup 2008: Robot Soccer World Cup XII*, 2009, pp. 49–60.

[16] J. Bruce, T. Balch, and M. Veloso, "Fast and inexpensive color image segmentation for interactive robots," in *Proc. 2000 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2000, pp. 2061–2066.

[17] Z. Zhang, "Parameter estimation techniques: A tutorial with application to conic fitting," *Image and Vision Computing*, vol. 15, 1996.

[18] J. Kober, K. Mülling, O. Krömer, C. Lampert, B. Schölkopf, and J. Peters, "Movement templates for learning of hitting and batting," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2010, pp. 853–858.

[19] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[20] D. Mellinger and V. Kumar, "Minimum swap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2011, pp. 2520–2525.