Adaptive Robot Systems in Highly Dynamic Environments: A Table Tennis Robot

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

M.Sc. Jonas Tebbe

aus Nordwalde

Tübingen

2022

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	30.06.2022
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. rer. nat. Andreas Zell
2. Berichterstatter:	Prof. Dr. rer. nat. Andreas Schilling

Abstract

Background: Robotic table tennis systems offer an ideal platform for pushing camera-based robotic manipulation systems to the limit. The unique challenge arises from the fast-paced play and the wide variation in spin and speed between strokes. The range of scenarios under which existing table tennis robots are able to operate is, however, limited, requiring slow play with low rotational velocity of the ball (spin).

Research Goal: We aim to develop a table tennis robot system with learning capabilities able to handle spin against a human opponent.

Methods: The robot system presented in this thesis consists of six components: ball position detection, ball spin detection, ball trajectory prediction, stroke parameter suggestion, robot trajectory generation, and robot control.

For ball detection, the camera images pass through a conventional image processing pipeline. The ball's 3D positions are determined using iterative triangulation and these are then used to estimate the current ball state (position and velocity).

We propose three methods for estimating the spin. The first two methods estimate spin by analyzing the movement of the logo printed on the ball on high-resolution images using either conventional computer vision or convolutional neural networks. The final approach involves analyzing the trajectory of the ball using Magnus force fitting. Once the ball's position, velocity, and spin are known, the future trajectory is predicted by forward-solving a physical ball model involving gravitational, drag, and Magnus forces.

With the predicted ball state at hitting time as state input, we train a reinforcement learning algorithm to suggest the racket state at hitting time (stroke parameter). We use the Reflexxes library to generate a robot trajectory to achieve the suggested racket state.

Results: Quantitative evaluation showed that all system components achieve results as good as or better than comparable robots. Regarding the research goal of this thesis, the robot was able to

- maintain stable counter-hitting rallies of up to 60 balls with a human player,
- return balls with different spin types (topspin and backspin) in the same rally,
- learn multiple table tennis drills in just 200 strokes or fewer.

Conclusion: Our spin detection system and reinforcement learning-based stroke parameter suggestion introduce significant algorithmic novelties. In contrast to previous work, our robot succeeds in more difficult spin scenarios and drills.

Key words: Robotic table tennis, robot vision, reinforcement learning

Kurzfassung

Hintergrund: Tischtennis bietet ideale Bedingungen, um Kamera-basierte Roboterarme am Limit zu testen. Die besondere Herausforderung liegt in der hohen Geschwindigkeit des Spiels und in der großen Varianz von Spin und Tempo jedes einzelnen Schlages. Die bisherige Forschung mit Tischtennisrobotern beschränkt sich jedoch auf einfache Szenarien, d.h. auf langsame Bälle mit einer geringen Rotation.

Forschungsziel: Es soll ein lernfähiger Tischtennisroboter entwickelt werden, der mit dem Spin menschlicher Gegner umgehen kann.

Methoden: Das vorgestellte Robotersystem besteht aus sechs Komponenten: Ballpositionserkennung, Ballspinerkennung, Balltrajektorienvorhersage, Schlagparameterbestimmung, Robotertrajektorienplanung und Robotersteuerung.

Zuerst wird der Ball mit traditioneller Bildverarbeitung in den Kamerabildern lokalisiert. Mit iterativer Triangulation wird dann seine 3D-Position berechnet. Aus der Kurve der Ballpositionen wird die aktuelle Position und Geschwindigkeit des Balles ermittelt.

Für die Spinerkennung werden drei Methoden präsentiert: Die ersten beiden verfolgen die Bewegung des aufgedruckten Ball-Logos auf hochauflösenden Bildern durch Computer Vision bzw. Convolutional Neural Networks. Im dritten Ansatz wird die Flugbahn des Balls unter Berücksichtigung der Magnus-Kraft analysiert.

Anhand der Position, der Geschwindigkeit und des Spins des Balls wird die zukünftige Flugbahn berechnet. Dafür wird die physikalische Diffenzialgleichung mit Gravitationskraft, Luftwiderstandskraft und Magnus-Kraft schrittweise gelöst.

Mit dem berechneten Zustand des Balls am Schlagpunkt haben wir einen Reinforcement-Learning-Algorithmus trainiert, der bestimmt, mit welchen Schlagparametern der Ball zu treffen ist. Eine passende Robotertrajektorie wird von der Reflexxes-Bibliothek generiert.

Ergebnisse: In der quantitativen Auswertung erzielen die einzelnen Komponenten mindestens so gute Ergebnisse wie vergleichbare Tischtennisroboter. Im Hinblick auf das Forschungsziel konnte der Roboter

- ein Konterspiel mit einem Menschen führen, mit bis zu 60 Rückschlägen,
- unterschiedlichen Spin (Über- und Unterschnitt) retournieren
- und mehrere Tischtennisübungen innerhalb von 200 Schlägen erlernen.

Schlußfolgerung: Bedeutende algorithmische Neuerungen führen wir in der Spinerkennung und beim Reinforcement Learning von Schlagparametern ein. Dadurch meistert der Roboter anspruchsvollere Spin- und Übungsszenarien als in vergleichbaren Arbeiten.

Schlagwörter: Tischtennisroboter, Robot Vision, Reinforcement Learning

Contents

Publications

Parts of this thesis have been published elsewhere. This section lists all publications that form part of this thesis and outlines my contribution to each of them. The publications themselves can be found in the appendix. This thesis focuses on the papers 1. - 3.

1. Tebbe, J., Gao, Y., Sastre-Rienitz, M., and Zell, A. (2018). A Table Tennis Robot System using an industrial KUKA Robot Arm. In *Pattern Recognition. GCPR 2018*, pages 33–45, Stuttgart, Germany

   Ball detection and prediction were designed and implemented by me in part using code from a previous Bachelor thesis by Eduard Hez. Robot control was implemented by Marc Sastre-Rienietz under my supervision. Calibration and triangulation algorithms were designed by Yapeng Gao. The main demonstration experiment was planned, recorded and evaluated by me. Half of the manuscript was written by me and the rest by Marc Sastre-Rienietz and Yapeng Gao. The research process was supervised by Andreas Zell. I consider my contribution to this work to be 50% of the total work. The manuscript is found in Appendix A.

2. Tebbe, J., Klamt, L., Gao, Y., and Zell, A. (2020). Spin Detection in Robotic Table Tennis. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9694–9700, Paris, France

   The conventional image processing pipeline was implemented by me, partly using code from a previous Bachelor thesis by Katharina Emde. The CNN method was implemented by Lukas Klamt under my supervision. The Magnus force fitting was designed and programmed by me. The accuracy of all algorithms was analysed by me. The demonstration on the robot was planned, recorded and evaluated by me. The manuscript was mainly written by myself with contributions from Yapeng Gao. The research process was supervised by Andreas Zell. I consider my contribution to this work to be 70% of the total work. The manuscript is found in Appendix B.

3. Tebbe, J., Krauch, L., Gao, Y., and Zell, A. (2021). Sample-efficient Reinforcement Learning in Robotic Table Tennis. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4171–4178, Xian, China

   The reinforcement learning algorithm was programmed by me based on a first implementation by Lukas Krauch. All experiments were planned, recorded and evaluated by me. The manuscript was mainly written by me with contributions from Yapeng Gao. The research process was supervised by Andreas Zell. I consider my contribution to this work to be 70% of the total work. The manuscript is found in Appendix C.

4. *Conference-Paper*: Gao, Y., Tebbe, J., Krismer, J., and Zell, A. (2019a). Markerless racket pose detection and stroke classification based on stereo vision for table tennis robots. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 189–196
   Journal-Paper: Gao, Y., Tebbe, J., and Zell, A. (2019b). Real-time 6d racket pose estimation and classification for table tennis robots. *International Journal of Robotic Computing*, **1**(1), 23–39

   The detection algorithm was programmed by Yapeng Gao. I helped on planning and recording the experiments. The manuscript was mainly written by Yapeng Gao with contributions from me. The part on comparing the main method to stroke classification using convolutional pose machines was done by Julian Krismer and me. The research process was supervised by Andreas Zell. I consider my contribution to this work to be 20% of the total work. The manuscript is found in Appendix D.

5. Gao, Y., Tebbe, J., and Zell, A. (2021). Robust stroke recognition via vision andimu in robotic table tennis. In *International Conference on Artificial Neural Networks*, pages 379–390, Cham. Springer, Springer International Publishing

   The stroke recognition networks were developed by Yapeng Gao. I helped on planning and recording the experiments. The manuscript was mainly written by Yapeng Gao with contributions from me. The research process was supervised by Andreas Zell. I consider my contribution to this work to be 10% of the total work. The manuscript is found in Appendix E.

Chapter 1

Introduction

The first large-scale commercial use of robotics was the use of industrial robotic arms in manufacturing. Industrial robots of this type require factories and the manufacturing process to be adapted accordingly. Although these robots are able to perform specific tasks much faster, more safely, and in particular more cheaply than human workers, materials and tools have to be perfectly prepared.

Today, robotic arms are being used for increasingly dynamic scenarios in combination with camera systems for tasks such as picking up unorganized components on a conveyor belt. In future, humans and machines will need to be able to cooperate. This will require robots that are no longer strictly separated in safety cages, but are able to operate in the direct vicinity of human workers. Successful cooperation will require efficient data processing and rapid reactions.

A broad range of basic research on combining robotic arms and computer vision has been undertaken. Since in many cases these systems do not have strict real-time requirements, the movement can be planned ahead of execution. Practical application such as manufacturing usually have real-time requirements to some degree. To explore the limits of what such systems are capable of requires more demanding applications. Robotic table tennis provides a good research model for evaluating capabilities in a complex environment.

1.1 Robotic Table Tennis

Robotic table tennis presents dynamic robot systems with a number of challenges:

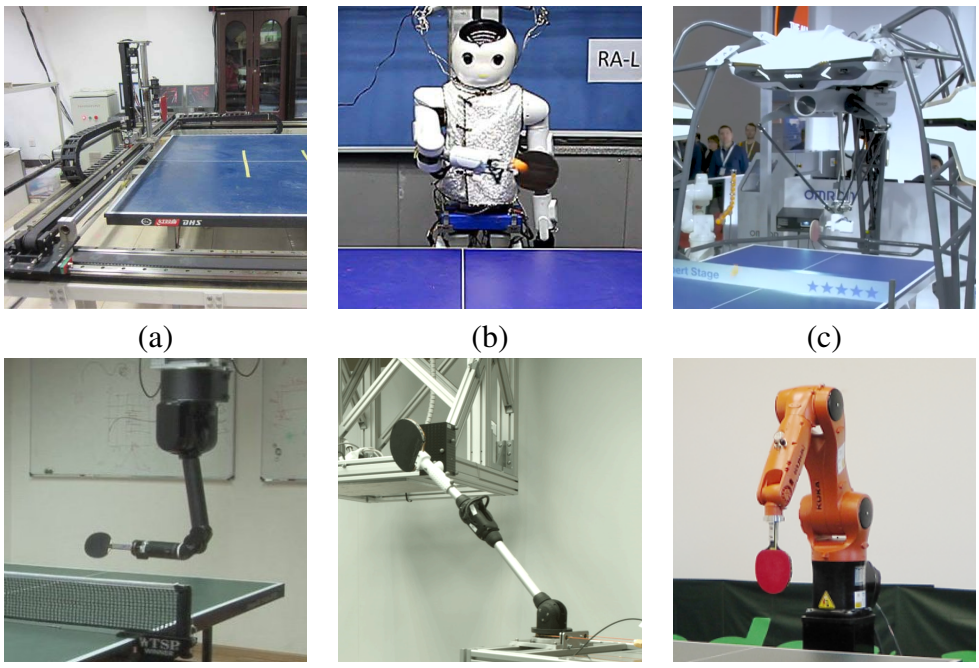
- **Rapid reaction time.** From the ball hitting the racket to its arrival at the opposite

side of the table takes between just 0.1 and 1 second. During this interval, the software has to analyze the trajectory of the ball and plan and execute a robotic return stroke.

- **Noisy measurements.** The robot system has to deal with a number of sources of noise in ball measurements, trajectory prediction, and arm movement.
- **Hidden variables.** Ball rotation (spin) is difficult to measure but essential to know. Different types of strokes produce different types of spin. Even for a human player, accurately estimating spin requires a lot of experience.
- **Dynamic motion planning.** The accuracy with which it is possible to predict the ball's future trajectory is heavily dependant on the number of measurements of the ball's position. For a successful stroke, the predicted trajectory needs to be updated regularly and the movement of the robot arm readjusted each time.
- **Human contact.** The robot directly reacts to the actions of a human table tennis player. In the case of cooperative play, consideration needs to be given to the fact that human strokes constantly vary and that human players may be impatient for the robot to adapt. In the case of competitive play, anticipating human strokes and strategies is crucial.

Ever since Billingsley initiated a robot table tennis competition in 1983 (Billingsley, 1983), robotic table tennis has been a popular tool for research in image processing and robot control. Various types of manipulators have been used. Huang *et al.* (2011) used a 5-DOF robot with three linear axes plus a pan-tilt unit. Xiong *et al.* (2012) developed two human-like robots, Wu & Kong. Both robots have 30 DOF in total, with two 7-DOF arms, two 6-DOF legs, and 4-DOF for head and hip. Omron frequently showcases its Delta robot at trade shows, with a table tennis racket attached after two additional swivel joints (Asai *et al.*, 2019). Büchler *et al.* (2020) have designed a completely new pneumatic robot arm able to attain very high end effector speeds. Particularly popular are industrial 6 or 7-axis articulated arm robots in which all joints are rotational and which are relatively similar to the human arm (Muelling *et al.*, 2013; Lin *et al.*, 2020; Gao *et al.*, 2020). Our system also employs this type of robot, the Agilus KR6 R900 sixx made by KUKA.

[1] `https://industrial.omron.eu/en/news-events/news/hannover-messe-may-2019`



(a) (b) (c)

(d) (e) (f)

Figure 1.1: A range of robots have been used for robotic table tennis. (a) Linear robot [Reprinted from Huang *et al.* (2013), © 2013 IEEE] (b) Humanoid robot Wu [Reprinted from Zhao *et al.* (2016), © 2016 IEEE] (c) Omron Delta robot [Taken from press release video [1], © 2019 Omron] (d) Barrett WAM robot arm [Reprinted from Muelling *et al.* (2013), © 2013 SAGE Publications] (e) Pneumatic robot arm [Reprinted from Büchler *et al.* (2020), unpublished] (f) Our KUKA Agilus robot.

A number of different approaches to developing table tennis robots are reported in the literature, each with a different research focus. Many published papers are based on simulations. Different methods for finding optimal policies have been tested, including reinforcement learning with sparse rewards (Peters *et al.*, 2010), simplified one-step environments (Zhu *et al.*, 2018) and evolutionary search for a CNN-based policy (Gao *et al.*, 2020).

Others mimic successful return patterns from human experts. Mahjourian *et al.* (2018) captured human strokes in a virtual reality environment and used self-play to optimize a policy based on these strokes. Muelling *et al.* (2013) recorded stroke movements by physically controlling the robot (kinesthetic teaching). Human demonstrations were then combined using dynamic movement primitives to produce a complete trajectory of the robot arm.

Research to date has produced some highly successful methods, though a number of limitations remain. There is still much work to be done before we will be able to build a robot able to compete with a human. Current research robots are only able to play table tennis in very limited scenarios. Most robots fail to take account of the spin of the ball or do so in theory only, while in demonstrations the spin varies very little.

1.2 Contributions

Our research is focused on removing some of these limitations. To this end, there are a number of topics which are addressed throughout this thesis. The first is playing against a human opponent. Previous research has mainly been tested against ball-throwing machines, which deliver a roughly similar ball every time. In games against humans, testing has been limited to scenarios where the human player plays slow balls. In our paper

- Tebbe, J., Gao, Y., Sastre-Rienitz, M., and Zell, A. (2018). A Table Tennis Robot System using an industrial KUKA Robot Arm. In *Pattern Recognition. GCPR 2018*, pages 33–45, Stuttgart, Germany

we presented a fast-reacting robot system which used the KUKA Agilus KR6 R900 sixx industrial robot arm. Our system achieved greater robustness and responsiveness than systems described in other research papers. In a fast counter-attack game against a human player, the robot achieved continuous rallies of up to 50 strokes.

The second topic is spin estimation. In table tennis, the spin of the ball has a number of effects on the game. Through the Magnus effect it affects the trajectory of the ball through the air. This is important for hitting the ball accurately, where a few centimetres are crucial. The spin also exerts an effect during the impact with both table and racket. The largest effect is exerted during the contact between ball and racket. The ball spins into the rubber of the racket and bounces out again. A small change in spin generally results in a large difference in the bounce angle.

Accurate spin detection is therefore essential for successful returns. In our publication

- Tebbe, J., Klamt, L., Gao, Y., and Zell, A. (2020). Spin Detection in Robotic Table Tennis. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9694–9700, Paris, France

we developed and compared three very different methods. In contrast to previous research, we were able to show that our system is sufficiently robust to be able to return both balls with topspin and balls with backspin from a human player within a single rally.

The majority of existing table tennis robots are unable to undergo further adaptation after development. The algorithms do not improve during a game or do so only with great difficulty. Our paper

- Tebbe, J., Krauch, L., Gao, Y., and Zell, A. (2021). Sample-efficient Reinforcement Learning in Robotic Table Tennis. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4171–4178, Xian, China

goes one step further. Our objective was, starting from zero with random stroke parameters, to teach the robot a successful table tennis stroke. Our method resulted in a very successful return stroke being learned in multiple scenarios, with an error of less than 30 cm to a landing target on the table tennis table in less than 200 balls against a human training partner.

Chapter 2

Research Objective

The goal of the project associated with this thesis was the development of an adaptive table tennis robot capable of playing real table tennis, in particular capable of dealing with spin (e.g. topspin vs backspin).

The first step was to develop and build a basic robot system. Required components were an algorithm to track the table tennis ball, a method for predicting future ball trajectory and an algorithm for controlling the robot. The robotic platform for the project is an industrial robot arm with six rotary joints, specifically the KUKA Agilus KR6 R900 sixx. Five cameras were employed to sense the environment, four PointGrey Chameleon3 (1.3MP at 149 fps) and one PointGrey Grasshopper3 (2.3 MP at 163fps).

This is not the first such project, and several table tennis robots have been developed in a number of countries. Until now, however, these robots (with the exception of one industrial project by Omron) have been capable of playing against a human in very simple scenarios only. Balls are served by a ball throwing machine or by a human with very little spin and speed. In real table tennis, however, the spin of the ball is crucial. Placement and speed variation also play a significant role. The next steps would therefore involve developing the basic system to enable it to deal with these more challenging situations. In particular the robot should be able to play competently against both a ball throwing machine and a human opponent, against low and high spinning balls, and against variable speed and placement. The robot should also learn to adapt, i.e. use advanced machine learning techniques to improve over time.

How do we measure the outcome? Qualitative evaluation, even by non-expert observers, is simple. Does the robot hit the ball and does the ball come back to the original side of the table? Is this behavior repeatable, even in different placement, speed and spin scenarios? The system as a whole can be assessed quantitatively by means of the

success rate and, if we specify a target on the table, by calculating the error between the ball's landing point and the target. Additionally, it is also possible to analyze each component separately. Ball detection can be evaluated by measuring the error for known ball positions (e.g. a ball mounted on the robot's end effector). Trajectory prediction can be evaluated by comparing the prediction arrived at using only a part of the trajectory against the recorded ground truth for a set of recorded trajectories.

Chapter 3

Algorithmic Updates

This chapter presents algorithmic descriptions that were not included in the published papers, because they were developed after publication.

3.1 Ball Detection

We start with ball detection. In our paper Tebbe *et al.* (2018) we used a combination of frame difference and color thresholding to extract the pixels of the ball. A new version of this method uses the difference on one channel followed by stricter filtering on contour shapes in the final binary image. This version delivers faster, more accurate results.

Fig. 3.1 depicts the image processing pipeline for both the version described in the above paper and the updated version. The approach used in the paper combines frame difference with color thresholding. The images are converted to HSV color space for more accurate color thresholding. This conversion takes up a large portion of the execution time. The newer approach computes the frame difference for the red channel of the RGB images only. The red channel was chosen because our balls are orange.

The new method significantly shortens average execution time. The time for the whole image (1280×1024) is reduced from 2.2 ms to 0.9 ms. During play, we restrict the region of interest, as the expected next position can be deduced from previous ball positions. We restrict our evaluation to a 160×160 pixel square centered on the expected position. This reduces the execution time from about 0.8 ms to 0.15 ms.

As can be seen in the bottom row of Fig. 3.1, the new method produces a final binary image containing larger white regions (blobs), due to the missing color information. More rigorous filtering of the different blobs in the next step compensates for this appar-

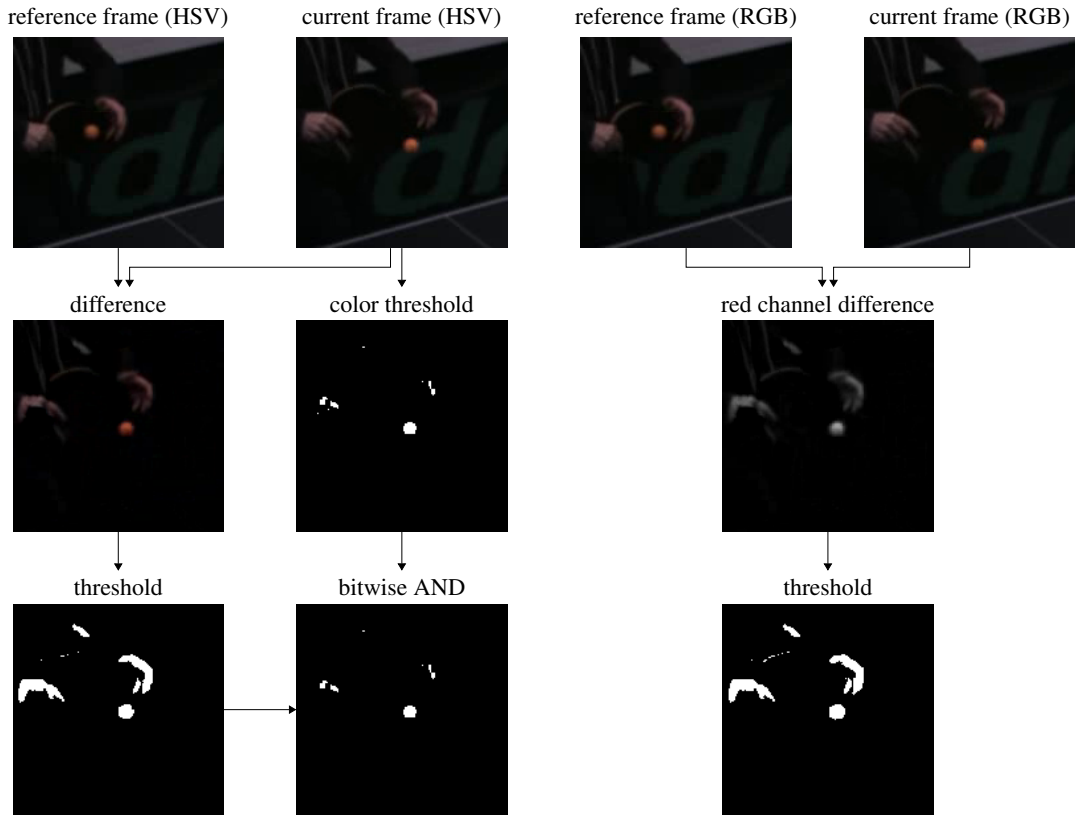


Figure 3.1: Ball detection image pipeline. Note that negative values in the difference image are capped to zero/black. Left: Paper version. Right: New faster version.

ent shortcoming. We first assess a number of shape features of the blobs. The first step is to remove contours which are the wrong size. The median color and other computationally expensive features are then extracted. This enables the most complex part of the analysis to be restricted to a small portion of the image, specifically those elements where a red color change of a specific size is detected. This approach speeds up the process considerably.

The pseudo-code for the new blob filtering method can be found in Algorithm 1. We first calculate multiple shape properties for a blob B. These calculations are computationally inexpensive. We use these shape features to remove blob candidates which are not ball-like. For a feasible blob, we then calculate the median color of the blob both in the original image and in the difference image. In addition, we calculate the minimal B-enclosing circle. For a ball, the center of the circle gives an accurate sub-pixel position estimate. We then calculate the distance between the center and the last found ball position. The calculated values (shape descriptions, color value and distance) are ranked

based on how likely it is that they belong to a detected table tennis ball blob. This ranking was designed by a process of trial and error, based on ball blobs and misdetected blobs. With the large range of properties, it is very easy to design a ranking which assigns low values to shapes associated with something other than the ball, such as the robot. We then simply choose the blob with the highest ranking. If this ranking exceeds a specific threshold, the blob circle center is considered to be the ball position.

Algorithm 1 Blob filtering

Input: array B of white area blobs in the difference image
Output: ball pixels position p or *not found*

for each $b \in B$ **do**
  $R \leftarrow$ bounding rectangle for b
  $a \leftarrow (\text{height of } R)/(\text{width of } R)$ ▷ aspect ratio of R
  $A \leftarrow$ area of b
  $e \leftarrow A/(\text{area of } R)$ ▷ extend value
  $c \leftarrow (4\pi A)/(\text{contour length of } b)^2$ ▷ circularity
  if a, A, e or c not in predefined range **then**
    continue
  end if
  $d \leftarrow$ median color of R in current frame
  $b \leftarrow$ median color of R in difference image
  $C \leftarrow$ minimal b-enclosing circle
  $p \leftarrow$ center of C
  $\Delta p \leftarrow$ distance to previous ball center or 0
  $r \leftarrow 0$, initialization ranking value
  for each property value $v \in \{a, A, e, c, d, b, \Delta p\}$ **do**
    $r' \leftarrow$ rank v for ball-likelihood from $[0, 1]$, worst 0 to best 1
    $r \leftarrow$ r + r'
  end for
  save r and p for b
end for
$b, r, p \leftarrow$ best ranked blob with rank r and circle center p
if $r >$ minimal acceptable rank **then**
  return p
else
  return *not found*
end if

Ball detection accuracy is described in the Section 4.1. The original algorithm experienced problems with the moving robot, leading to many misdetection events. The improvements in filtering have resolved all of these cases, resulting in more robust, more accurate detection.

3.2 Trajectory Prediction

In Tebbe *et al.* (2018) we described a pure curve fitting approach. This method is not able to take into account estimated spin values. The current prediction system instead first estimates the ball state and then applies a physical force model to predict the trajectory.

The above paper also detailed an alternative approach using an extended Kalman filter. The accuracy of the Kalman filter is highly sensitive to the initialization and the model of process. In recent unpublished work, we therefore implemented additional state estimation techniques, firstly fitting the state via polynomials and secondly optimizing the state using Levenberg-Marquardt.

Here we introduce the model and the prediction method. The physical ball model is defined by the following differential equation:

$$a = \dot{v} = \ddot{p} = -k_D \|v\| v + k_M \, \omega \times v - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}. \tag{3.1}$$

The first acceleration term represents the drag force (air resistance) acting in the opposite direction to the flight of the ball. The second component includes the Magnus force generated by the ball's spin. This force is perpendicular to the axis of rotation and the direction of flight. With the last term we include gravitation. The coefficients are

$$k_D = -\frac{1}{2} C_D \rho_a A m^{-1} \tag{3.2}$$

$$k_M = \frac{1}{2} C_M \rho_a A r m^{-1}. \tag{3.3}$$

The following constants are used: the mass of the ball $m = 2.7\text{g}$, the gravitational constant $g = 9.81\text{m/s}^2$, the drag coefficient $C_D = 0.4$, the density of air $\rho_a = 1.29\text{kg/m}^3$, the lift coefficient $C_M = 0.6$, the radius of the ball $r = 0.02\text{m}$, and the cross-sectional area of the ball $A = r^2\pi$.

Prediction Given estimates of the ball's position p, velocity v and spin ω, we wish to predict the ball's future trajectory. The physical ball model from Eq. (3.1) defines a second order ordinary differential equation (ODE). To make a prediction for the ball curve in mid-air, we solve the ODE given an initial ball state. This is called an initial value problem (IVP). We first reformulate the equation to give:

$$\dot{p} = v \tag{3.4}$$

$$\dot{v} = f_v(v) = -k_D \|v\| v + k_M \, \omega \times v - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{3.5}$$

Defining the state as $s = [p, v]^T$, we get a first order ODE:

$$\dot{s} = \begin{bmatrix} \dot{p} \\ v \end{bmatrix} = \begin{bmatrix} v \\ f_v(v) \end{bmatrix} = f_s(s). \tag{3.6}$$

We can now integrate the differential equation stepwise using an IVP solver. The naive approach is the Euler method as used in Tebbe *et al.* (2018). A prediction of one time step Δt is

$$s_{n+1} = s_n + \Delta t * f(s_n) \tag{3.7}$$

The accumulated error compared to a perfect solution is in the range $\mathcal{O}(\Delta t)$. More accurate solvers, such as classical fourth order Runge Kutta ($\mathcal{O}(\Delta t^4)$) are available. In our case, this improves accuracy by only $0.1mm$, which is much less than the error resulting from state uncertainty. (See Table 4.6 in the results chapter.)

At the bounce point, the speed v and angular velocity ω are transformed by the following linear model:

$$\begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} := \begin{bmatrix} \alpha_x & 0 & 0 & 0 & \beta_x & 0 \\ 0 & \alpha_y & 0 & \beta_y & 0 & 0 \\ 0 & 0 & -\alpha_z & 0 & 0 & 0 \\ 0 & \gamma_x & 0 & \delta_x & 0 & 0 \\ \gamma_y & 0 & 0 & 0 & \delta_y & 0 \\ 0 & 0 & 0 & 0 & 0 & \delta_z \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3.8}$$

A derivation of the model can be found in Nakashima *et al.* (2010). The values used in our experiments are $\alpha_x = \alpha_y = 0.75$, $\alpha_z = -0.97$, $\beta_x = \beta_y = 0.0015$, $\gamma_x = -26, \gamma_y = 25, \delta_x = 0.53$, $\delta_y = 0.6$, and $\delta_z = 0.9$, taken from Zhang *et al.* (2014).

After the bounce we continue solving the ODE until the ball reaches a point 100 mm before the end of the table ($p_x = 100$). This is the point at which we aim to have the robot hit the ball.

Next, we will discuss the state estimation. We start by briefly recapping the EKF introduced in Tebbe *et al.* (2018) and then explain our new approaches.

EKF The extended Kalman filter (EKF) uses a series of possibly noisy measurements to iteratively update an estimate of an unknown variable in a non-linear system. We define the state of the ball as $s = [p, v]$, which is estimated by the EKF. For the algorithm we need the state transition function and an observation model. Using the ODE model from Eq. (3.6) we define the state transition as $f(s_n) = \Delta t * f_s(s_n)$. The observation $h([p_n, v_n]) = p_n$ is just the projection to the first positional part of the state. The pseudocode is shown in Algorithm 2.

Algorithm 2 Extended Kalman filter

Input: array B of all ball positions and times T
Input: spin vector s
Input: physical prediction function $f(t, (pos, vel), spin) \rightarrow (pos, vel)$
Output: ball position and velocity at time t_n

initialize process covariance matrix Q
initialize measurement covariance matrix R
initialize state vector x
initialize state covariance matrix P
for each $p \in B$ and corresponding $t \in T$ **do**
    $x \leftarrow f(\Delta t, x, s)$
    $A \leftarrow$ jacobian $f'(\Delta t, x, s)$
    $P \leftarrow APA^T + Q$
    $H \leftarrow (I_3, 0_3)$ ▷ concatenation of 3×3 identity and zero matrix
    $K \leftarrow PH(HPH^T + R)^{-1}$
    $x \leftarrow x + K(p - Hx)$
    $P \leftarrow (I - KH)P$
end for
return $(p, v) = x$

Polynomial State Fitting In this approach a second-degree polynomial is fitted to each axis of the last 10 ball positions. This is similar to the curve fitting approach in Tebbe *et al.* (2018), but instead of using the fitted curve for prediction, the position and velocity are obtained by evaluating the polynomial function and its derivative at the current time. The trajectory is then predicted using our physical model. The process is detailed in Algorithm 3. For large time values, fitting via linear least squares leads to a numerical error. Normalizing the data to mean 0 and variance 1 fixes this problem. One advantage over the other approaches is that this approach is more robust and there are no hyperparameters requiring tuning.

Algorithm 3 Polynomial state fitting

Input: array B of the last n ball positions and times T
Output: ball position and velocity at time t_n

for each $a \in \{x, y, z\}$ **do**
    $B_a \leftarrow a$-axis values in B normalized by mean and variance
    $T' \leftarrow$ times points in T normalized by mean and variance
    $P_a \leftarrow$ polynomial of degree 2 fitted to B_a with times T
    $P_a \leftarrow$ rescaled P_a to remove normalization
end for
$p \leftarrow (P_x(t_n), P_y(t_n), P_z(t_n))$
$v \leftarrow (P'_x(t_n), P'_y(t_n), P'_z(t_n))$
return p, v

State Optimization The third method involves using the physical ball model to estimate the past trajectory for a specific position p and velocity v. We want to find p and v minimizing the error between estimated trajectory and the measured ball positions. An extremely fast optimization technique is the Levenberg-Marquardt (LM) algorithm. This is a more robust, regularized variant of the Gauss-Newton method for solving non-linear least squares problems. In proximity to a minimum it converges with quadratic speed. Otherwise, it still exhibits linear convergence. LM requires the error function to be in least squares form. We therefore sum squared errors over all measured positions and axes:

$$E(p,v) = \sum_{i=1}^{n} \|p_i - f(-t_i, p, v, \omega))\|_2 \tag{3.9}$$

Because the input dimension is low (6), the Jacobian matrix of the error function can still be efficiently calculated numerically by finite differences. The pseudocode is shown in Algorithm 4.

Algorithm 4 State Optimization

Input: vector B of the last n ball positions and times T
Input: spin vector s
Input: physical prediction function $f(t, pos, vel, spin) \rightarrow pos$
Output: ball position and velocity at time t_n

1: $p \leftarrow$ last ball position b_n
2: $v \leftarrow (B_n - B_{n-1})/(t_n - t_{n-1})$
3: $E(p,v) \leftarrow \sum_{i=1}^{n-1} \|B_i - f(-t_i, p, v, s)\|^2$ ▷ error function
4: $p, v \leftarrow \arg\min_{p,v} E(p,v)$ ▷ using Levenberg-Marquardt
5: **return** p, v

A comparison of the three state estimation methods is given in Section 4.3.

Chapter 4

Results and Discussion

In this section, we evaluate the performance of our table tennis robot quantitatively and qualitatively. We first examine the individual robot software components. Additionally, we look at potential next steps to improve upon current limitations. We then assess the robot system as a whole. Finally, we look at potential next steps to improve upon the current robot setup.

4.1 Ball Detection

The camera setup uses PointGrey Chameleon3 cameras recording at a frame rate of 150 fps. To achieve real-time ball detection, execution speed is crucial. We therefore keep image processing as simple as possible. First, the difference on the red channel between the current frame and a previous frame is calculated. The difference image is then converted into a binary image using thresholding. For each white region on the binary image, the likelihood that that region corresponds to the ball is evaluated. This evaluation makes use of the following properties, weighted accordingly: aspect ratio, area, perimeter, accuracy of the minimum enclosing circle, distance from the previous ball, median color on the current frame and on the difference image. The center of the most likely shape is treated as the ball position on the camera image. An iterative triangulation method (Manuscript 1, Section 3.1) is applied to the ball's pixel positions on the different cameras. This gives us the coordinates of the 3D position in the coordinate system used by the reference camera. These coordinates are then converted into the coordinate system defined for the table tennis table. A sequence of positions can then be used for the trajectory prediction algorithm. Section 3.1 covers the ball detection in depth.

Table 4.1: Average results per image on a human-labeled dataset of balls.

Method	Precision	Recall	Ball center error	Execution time (whole image)
rgb difference (paper)	99.5%	86.9%	1.32 px	0.8 ms (2.2 ms)
red diff + filtering (updated)	100%	92.8%	1.53 px	0.15 ms (0.9 ms)
shallow CNN	100.0%	95.6%	0.99 px	3.26 ms

Performance per camera image on a human labeled dataset of approximately 2500 images is shown in Table 4.1. The ball center error is defined as the distance (in pixels) between the detected and labeled ball centers. This error is calculated using correctly detected balls (true positives) only. The precision value is a measure of false positives. False positives are especially problematic because the scene contains other moving objects of similar color. The ball is not the only orange object, as the robot, a player's skin and the red side of the racket can also appear orange under some lighting conditions. More rigorous filtering based on shape, size and color features reduces the number of misdetections during testing to zero. The recall value enables us to determine whether the algorithm has failed to detect balls. A recall value in excess of 90% shows that most, but not all, balls are detected.

The performance is in the range of other robots. Ji *et al.* (2018) showed multiple methods with up to 90% accuracy and processing times starting from 15 ms. Qiao (2021) got 89% accuracy. Zhang *et al.* (2010) detected 97.8% of the balls at 10 ms processing time. Li *et al.* (2012) recognized 97-99% of 1000 balls at 15 ms latency. The time reported did, however, also include image capturing and transmission. Our results are slightly worse than the last two but we emphasized recording difficult balls in the dataset, e.g. balls with net or robot contact, to improve our algorithm on these cases.

Failure cases Fig. 4.1 shows four examples of balls that were not detected. If the ball is next to a moving object, such as the racket in the first example or the robot in the second, the ball can appear to merge with that object in the binarized difference image. This results in a shape which is no longer circular. A similar problem arises when the ball is directly in front of a very bright area of background, such as the lines marking the edge of the table. In these cases, the color difference between the white line and the bright orange ball is too small. As a result, the blob in the binary image omits the

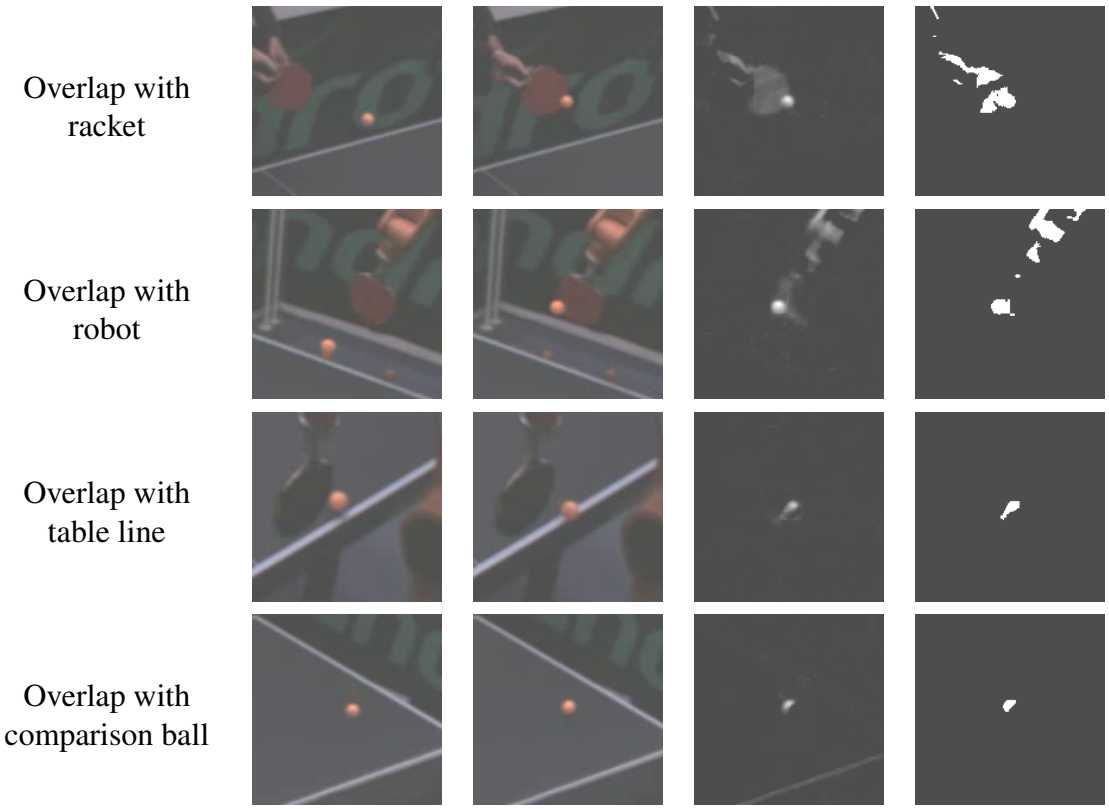


Figure 4.1: Four examples of false negatives, where the ball was not detected. Images are brightened to increase visiblity. In the top row, the ball is in front of a moving racket. In the second row, part of the robot appears to merge with the ball. In the third row, the ball is passing in front of the white line marking the edge of the table. The third row shows a very slow ball after hitting the net. (a): the comparison frame $((n-7)^{th})$. (b): the current frame (n^{th}). (c): subtraction of the red channel (a) from (b). (d): thresholding of (c).

part of the ball in front of the line, resulting in a non-circular shape. The final example shows a very slow ball. Because the balls in the current and comparison frames overlap, only a small part of the ball appears on the binary image. Such very slow balls are rare in normal play, but they do occur (as in this example) when the ball loses most of its velocity after hitting the net. One third of our false negatives were due to net contact. Most of the other cases occurred during the stroke, when the ball was next to a racket, human hand or the robot.

To access the accuracy of our system, we compared our ball positions to positions

obtained using the OptiTrack system, for which the company claims sub-millimeter accuracy. The OptiTrack system uses infrared cameras, so we coated some table tennis balls in infrared-reflective tape. Because these coated balls are a different color, the ball centers were labeled manually for this evaluation. This experiment is therefore solely a measure of triangulation accuracy. In addition we mounted a table tennis ball on the end effector of our robot arm and used the internal calculated position values as ground truth. In both cases we used a small number of balls to estimate a transformation between the different coordinate systems used by the different systems. Results are shown in Table 4.2. The accuracy is comparable to similar setups: 9 mm by Zhang *et al.* (2015), 5 mm by Yu *et al.* (2013), and 19.9 mm by Gomez-Gonzalez *et al.* (2019).

Table 4.2: Average position error between our method and other systems. The first row shows the error against the OptiTrack system for balls coated in reflective markers. The second row evaluates against the robot's internally calculated position for balls mounted on the robot.

Evaluation	2 cameras	4 cameras
Error vs OptiTrack	6.8 mm	6.8 mm
Error vs robot arm	3.2 mm	2.0 mm

Further work Our ball detection system outputs the precise 3D position of a table tennis ball and is robust and fast even at high frame rates (150 fps). The current algorithm is, however, capable of tracking only a single ball. In experiments with our ball throwing machine, the ball sometimes hits the machine or net and bounces back onto the table when the next ball has already been played. The moving balls are then in the field of view of the cameras and the correct ball cannot be identified. In addition, some human players are impatient and serve the ball even when the previous ball is still on the table. In these cases it would be helpful to be able to track several balls at the same time. One impediment to distinguishing multiple balls is the large difference in viewing angle between the individual cameras, which are mounted in the corners of the playing area. As a result, a ball may be visible in different regions in each image or may be visible from one camera only. Triangulation tests can help in deciding which blobs belong to the same ball.

Combining pixel positions of different balls will in most cases introduce large reprojection errors. Tracking the different objects for each camera can help to filter out pos-

sible candidates beforehand. Finally, we need to choose which ball to use for trajectory prediction and robot movement. Selecting by velocity may be sufficient, as unimportant balls will typically be moving much more slowly.

Cameras are another factor, as better models become available over time. In addition to industrial cameras with higher frame rates or resolutions, new event-based cameras have recently been developed. Rather than transferring the whole image, these cameras trigger events reporting changes in brightness per pixel. For our stationary cameras, where the difference between frames is minimal, event-based cameras could be a useful option. These cameras would give us many more positions per second. However, the resolution may be lower, leading to larger measurement errors. Synchronizing these cameras might prove to be more challenging. For a real-time system it might be necessary to switch from a standard image processing pipeline, since image operations usually have execution times in the millisecond range, which would be too slow for frame rates higher than 1000 fps.

4.2 Spin Detection

To estimate the spin of the ball, we tried two fundamentally different approaches (Tebbe *et al.*, 2020). The first approach uses a Point Grey Grasshopper3 industrial high-speed camera mounted on the ceiling above the center of the table. By limiting the region of interest to an area of 400×1920 px, we were able to achieve a frame rate of 400 fps. This is fast enough to allow observation of the brand logo on the ball. To estimate ball rotation from the brand logo position, we developed a conventional image processing pipeline and a convolutional neural network (CNN) approach.

In the image processing pipeline, we first compare the cropped ball image with a reference image of a logo-free ball, obtained by computing the pixel-wise median for a sequence of 50 ball images. By thresholding the difference image, we are able to identify pixels that have a high probability of forming part of the brand logo. The pixel center of the logo is calculated by the median point and corrected for half-visible logos using circular segment fitting. The center of the logo is then projected onto the unit sphere. The movement of the logo on the unit sphere is used to estimate the rotation of the ball.

An alternative approach involves using a CNN to estimate the orientation of the ball in each frame. As the network architecture a ResNet-18 (He *et al.*, 2016) was used.

The network was trained on a dataset of 4656 human-labeled images. Ball orientation was identified by overlaying the ball image and a 3D ball model using the 3D modeling program Blender (Blender Online Community, 2016).

Table 4.3 shows an evaluation on the test data of the dataset. All balls were celluloid balls manufactured by Double Circle, featuring the same circular brand logo (see first image sequence in Fig. 4.2). The vector angle error is the angle between the estimated brand center vector and the ground truth on the unit sphere. The geodesic error is based on the geodesic distance in $SO(3)$ (Huynh, 2009). For two rotations it also includes the rotation around the center point of the logo, which is difficult to estimate for a circular brand logo. The execution times are measured on an Intel Core i7-8700K CPU and an NVIDIA GeForce GTX 1080 Ti GPU, respectively.

Table 4.3: Average results per image on a human-labeled ball orientation dataset.

Method	Geodesic angle error	Vector angle error	Execution time per image
Image processing pipeline	-	5.06°	0.3 ms (CPU)
Convolutional neural network	20.14°	4.23°	3.7 ms (GPU)

A third, fundamentally different, approach analyzes the trajectory of the ball. The rotation of the ball influences the trajectory via the Magnus effect, named after German experimental scientist Heinrich Gustav Magnus, who first came up with a physical explanation for the phenomenon. The Magnus force acts perpendicular to the direction of flight and axis of rotation of the ball. This effect only becomes apparent, however, when the trajectory is observed over a longer period of time. In addition to the Magnus effect, our physical ball flight model also models drag and gravity. To estimate the spin, we first approximate the trajectory using third-degree polynomials to obtain smooth time-parameterized functions for ball position and velocity. We then use these to calculate the forces for a sequence of time points. The Magnus force at each time point can be used to obtain a least squares solution for the spin.

Finally, we performed a comparison between the three spin detection algorithms. No exact ground truth values for spin are available. We therefore evaluated how well the algorithms perform in classifying the spin type. We chose three different magnitudes of three different spin types (topspin, backspin, and sidespin) for a total of 9 settings. For each setting, we recorded 50 trajectories served by a TTMatic 404 ball throwing machine.

Table 4.4: Classification accuracy of all the algorithms.

Spin type	Background subtraction	Bg. sub. + segment fit	CNN	Trajectory fitting
Backspin				
Low	88.0%	94.0%	96.0%	100.0%
Medium	84.0%	92.0%	94.0%	58.0%
High	70.0%	86.0%	80.0%	60.0%
Sidespin				
Low	94.0%	98.0%	98.0%	100.0%
Medium	68.0%	58.0%	74.0%	94.0%
High	60.0%	68.0%	66.0%	100.0%
Topspin				
Low	84.0%	90.0%	88.0%	86.0%
Medium	78.0%	86.0%	88.0%	96.0%
High	90.0%	96.0%	96.0%	100.0%
Total	79.6%	85.3%	86.7%	88.2%

Table 4.5: Spin medians for the trajectory fitting. Spin around the x-axis (long table side) is called corkscrew spin. Rarely, it occurs in serves or in the snake return.

Spin axis / type	x Corkscrew rev/s	y Top/backspin rev/s	z Sidespin rev/s
Backspin			
Low	2.6	15.5	-5.0
Medium	3.2	19.8	-2.5
High	3.1	20.1	-2.4
Sidespin			
Low	2.6	-4.7	-15.2
Medium	9.2	-7.5	-30.1
High	12.4	-9.0	-31.5
Topspin			
Low	0.0	-19.2	-7.2
Medium	-0.5	-25.1	-6.9
High	-5.5	-28.9	-6.1

The TTMatic 404 does not permit ball speed and rotation to be varied independently – balls with more spin also have higher velocities. The spin detection methods are applied for classification as follows. First, the three-dimensional spin value is determined for each trajectory. Then, for each of the spin settings, the element-wise median is calculated from the 50 spin values. Finally, for each trajectory spin value, we evaluated whether the closest median belongs to the correct setting. Results by spin type are shown in Table 4.4.

Analyzing overall accuracy for each algorithm shows that adding circular segment fitting improved the background subtraction method. CNN-based and trajectory fitting are slightly better, but the difference between the three algorithms in terms of overall accuracy is modest. As detailed below, each of the methods has lower performance for some spin type. Combining logo observation and Magnus force fitting may improve accuracy and reduce the number of spin detection errors.

Failure cases The logo-based variants are noticeably less accurate for sidespin. One reason for this is that the logo often rotates about itself, so that there is little or no detectable change in position (Fig. 4.2 top). The worst case is the opposite, where the logo is stationary and facing away from the camera (Fig. 4.2 middle). With all other spin types, there are cases where the brand logo is only just visible (Fig. 4.2 bottom). In this case determining the axis of rotation is extremely challenging, as the majority of logo movement is not visible to the camera. In these cases, logo observation is unable to identify the spin and trajectory analysis is less error-prone. Conversely, Magnus force fitting is poor at distinguishing between different backspin variants, where the median values for medium and high backspin were almost identical (Table 4.5). The human eye is, however, able to discern a difference between the trajectories for these two variants, as ball velocity differs. A clearer example taken from two balls played by humans is shown in Fig. 4.3. Visually, the two trajectories are similar before the bounce, as shown by the fact that the topspin values obtained using trajectory fitting are very similar. There is, however, a large difference in trajectory after the bounce, showing that the balls have different spin.

Further work Estimates of spin made using the trajectory method depend on the number of ball positions available and the accuracy of the curvature information. The more ball positions are available and the more accurate the curvature information, the better the spin estimate. The accuracy, especially for the early estimates, could be improved by

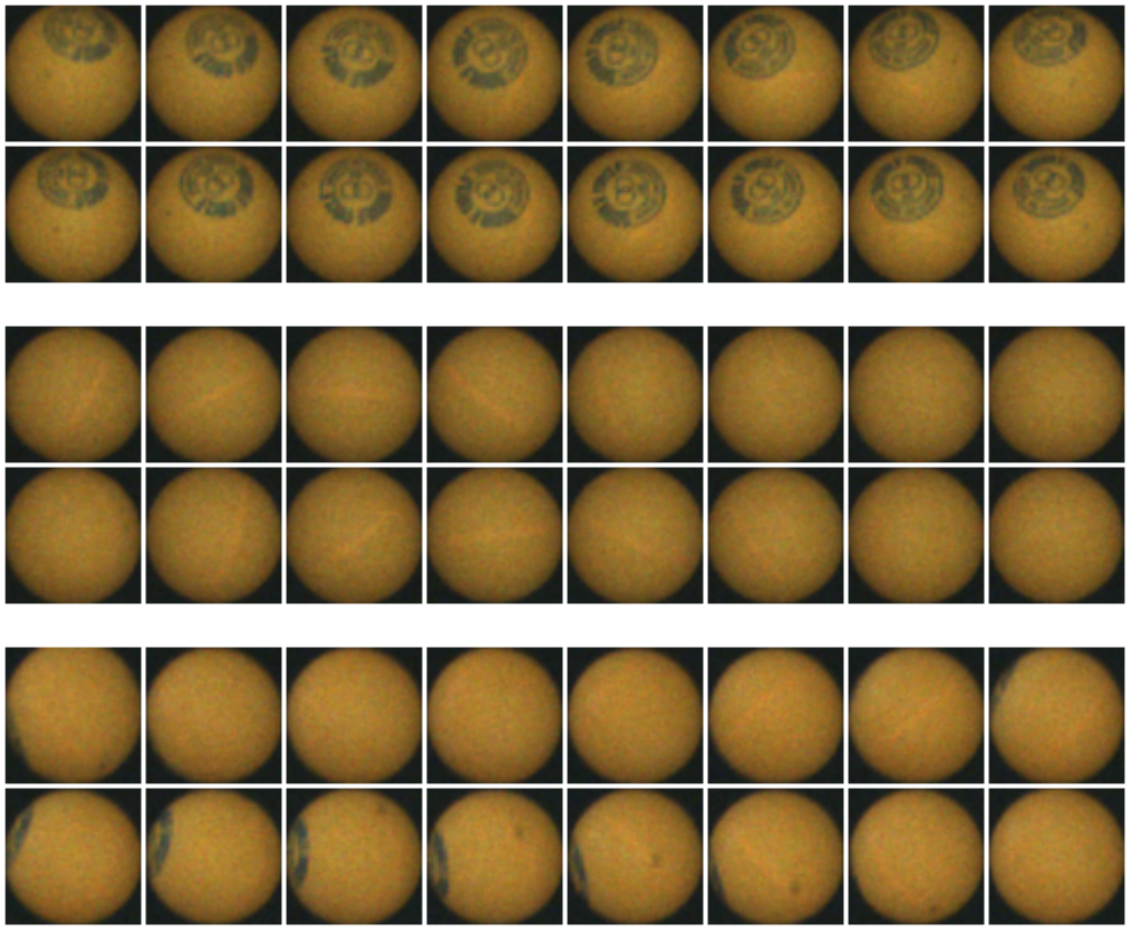


Figure 4.2: Sequences of ball images in which the brand logo rotates about itself (top), is not visible at all (middle) or is barely visible (bottom).

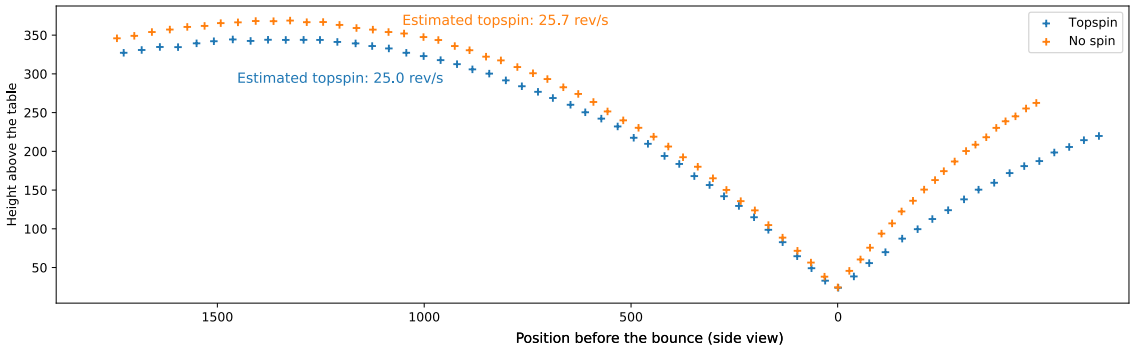


Figure 4.3: Trajectory curves viewed from the side for a ball with heavy topspin and with nearly no spin. Note that both balls have very similar movement along the axis, not shown in the plot.

using faster cameras, as suggested in the section on future work on ball detection. Another idea would be to start with a heuristic spin value. Spin types usually correlate with some other features of the ball trajectory such as initial height and velocity. A machine learning model trained on these features could generate a rough estimate of spin. This spin value could then be used to improve early trajectory predictions.

4.3 Trajectory Prediction

Trajectory prediction is performed in two phases. In the first phase, all measured 3D ball positions are used to estimate the current ball state (position, velocity and spin). The spin value is determined by the spin detection. We implemented three different position and velocity estimation algorithms. The first fits a parabola for each coordinate axis to the last 10 balls. Position and velocity are determined by evaluating and deriving the parabolic function at the most recent time point. The second algorithm iteratively feeds all measured balls positions into an extended Kalman filter (EKF). The state transition of the EKF is based on the physical ball model. The third algorithm estimates the state through optimization. The physical ball model is used to calculate state-dependend trajectories. The objective of optimization is to minimise the error between the calculated and measured trajectories. The Levenberg-Marquardt algorithm is applied to find the position and velocity with minimal error.

In the second phase, the estimated ball state is used to predict the ball's future trajectory. This is achieved by forward-solving the ordinary differential equation for the physical ball model. On hitting the table ($p_z < 0$), the ball state is modified using a linear bounce model. When the predicted x-position of the ball (along the long side of the table) exceeds a set x-value (10 cm from the end of the table), the iterative process is stopped, as this is the position at which the robot will hit the ball. The details are presented in Section 3.2

To evaluate the accuracy of the three state-estimation algorithms, we used the spin detection dataset described above. This dataset contains three spin magnitudes for topspin, backspin and sidespin. We recorded 50 balls for each of the nine combinations of spin type and magnitude served by a TTmatic 404 ball-throwing machine. All of the ball positions up to the point at which the ball arrives in the middle of the table are fed into the algorithms. The algorithms then predict the point at which the ball will reach the

Table 4.6: Hitting point prediction error for all three state estimation algorithms. The positional error (in mm) is denoted by *pos* and the time error in ms by *t*.

Spin type	Polyn. state fit		Optimization		EKF	
	pos	*t*	*pos*	*t*	*pos*	*t*
Topspin						
Low	46.2	21.3	39.4	25.0	15.0	30.8
Medium	33.7	13.7	23.2	15.7	31.1	22.7
High	46.2	7.2	22.6	10.0	39.0	17.6
Backspin						
Low	72.6	30.8	61.2	25.7	91.6	15.2
Medium	42.3	18.4	29.0	11.3	19.8	21.1
High	95.9	11.1	85.5	13.4	98.8	21.9
Sidespin						
Low	32.5	18.6	23.7	18.2	103.0	24.2
Medium	30.5	19.4	29.2	21.7	170.1	34.2
High	38.5	25.4	36.5	24.4	176.0	35.5

hitting plane, 10 cm before the end of the table. The results are shown in Table 4.6. The optimization method is slightly more accurate than the polynomial state fit. The EKF is especially worse at estimating the velocity for sidespin balls.

Trajectory prediction can also be found in various literature, with similar error values. The results are, however, not directly comparable between different papers. The accuracy highly depends on the difficulty of the dataset. In Huang *et al.* (2011), the hitting point of 12 trajectories with a spin of up to 100 rad/s was predicted with a precision of at least 40 mm. Zhang *et al.* (2014) reported an average error of 27.8 mm for the hitting point prediction, but they included only 15 trajectories, which had side spin of up to 200 rad/s. Zhao *et al.* (2017) predicted 85% of hitting points within a range of 50 mm error. The evaluation was done on approximately 900 balls, but the rotational velocity was always below 30 rad/s. In our case, we recorded 450 balls with different spin types rotating with up to 300 rad/s. In our approach we, thus, generalize over a wide range of spinning balls.

Limitiations Overall, trajectory prediction is pretty accurate, though it is somewhat sensitive to outliers. To correct for this, outlier detection is incorporated into the preprocessing step. Erroneous results occur in cases where one camera has a synchronisation

problem resulting in a sudden shift in timestamps, but this happens very rarely (less than once for every 100 balls) or spin detection returns an incorrect spin value (see Section 4.2).

4.4 Stroke Parameter Suggestion

Successfully returning the ball depends entirely on the state of the racket (pose and velocity) at hitting time. We therefore decided to first determine the racket state at hitting time, then use a path planner to generate an appropriate robot trajectory to achieve that state. During the early stages of robot development, we specified a human-engineered linear heuristic for each scenario, e.g. counter-hitting. This approach is very inflexible. Because it was developed using a limited number of trials, it works only for a very limited group of ball types.

Our most recent paper (Tebbe *et al.*, 2021) presents a self-learning policy using one-step reinforcement learning. We use a modified deep deterministic policy gradient (DDPG, Lillicrap *et al.* (2016)) based actor-critic model. The critic returns reward parameters rather than the calculated reward, reducing complexity. The actor is trained with the gradient of the composition of the critic and the reward function. We evaluated this method on the real robot for a number of scenarios of increasing difficulty:

- Simple backhand serves.

  The human always plays the same serve and the robot has to return to the middle of the table (target: $[2000, 0]$).

- Serve and human I-play.

  The human begins with a serve and the rally is continued along the mid-line of the table (target: $[2400, 0]$).

- Serve and human V-play.

  The human begins with a serve and is required to alternate ball placement between the left and right sides of the table, so that the ball traces an inverted 'V' (target: $[2400, 0]$).

- Serve and human X-play.

The human begins with a serve. In the following ball exchange the robot and the human place the ball alternately on both sides of the table, so that the ball traces an 'X' (target: $[2200, -300]$ and $[2200, 300]$).

The target positions for each scenario are given in the table coordinate system. $x = 0$ / 1370 / 2740 are the robot's table end / net line / human's table end, respectively, and $y = -762$ / 0 / 762 are the left side / middle line / right side of the table (in millimeters). The table is an ITTF standard size table (9 ft $\times$ 5 ft $=$ 2740 mm $\times$ 1524 mm). The target positions are shown in Fig. 4.4.

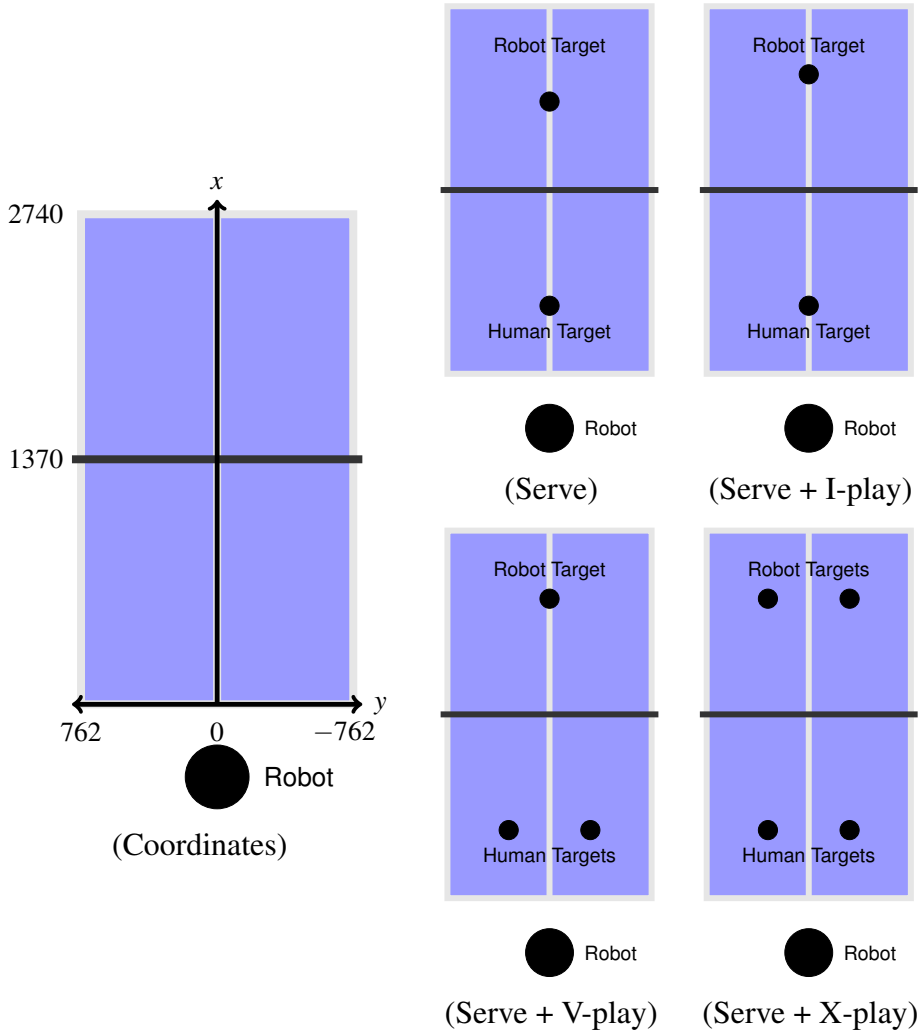


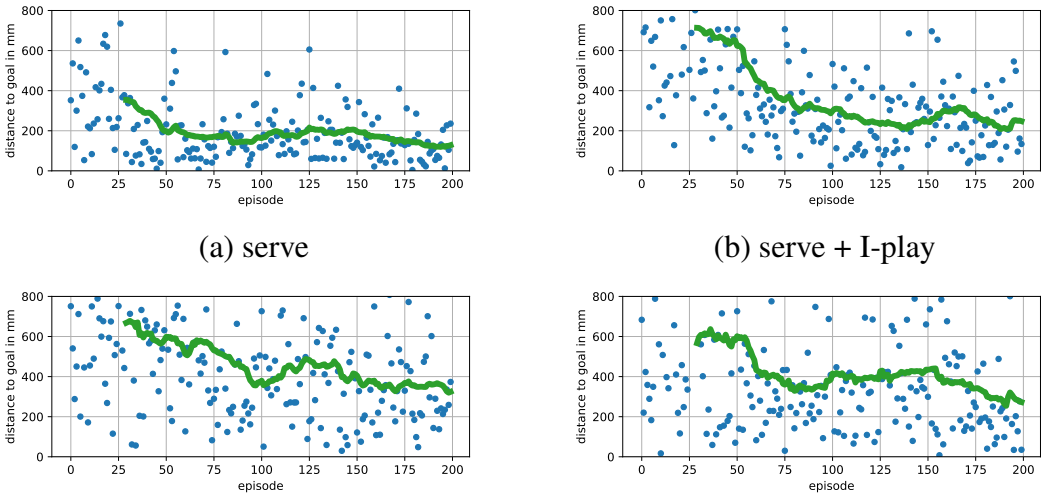
Figure 4.4: Coordinate system and target positions in the four scenarios.

The learning process of the scenarios is presented in figure 4.5. Balls returned from a ball machine with the same action each time deviate by 124*mm* on average. That gives us an idea of how accurate our robot can be. The serve-only scenario is close to that, with an average error of 136*mm* (x: 114*mm*, y: 47*mm*) to the goal. In I-play, the rally continues after the serve, making it more challenging. With 269.2*mm* accuracy in the last 50 episodes, performance is worse. However, the x-error of 243.3 mm and the y-error of 78 mm indicate accurate returns to the middle of the table. V-play is more diverse in human play since the ball placement alternates. In this case, the goal error is 329*mm* (x: 177*mm*, y: 126*mm*). It's even more complex in the X-play setting in which the goal error is 393*mm* (x: 282*mm*, y: 238*mm*).

Table 4.7 shows a performance comparison with table tennis robots from other publications. We aim to play the ball as close to the target position as possible. In this respect, our approach achieves a high level of accuracy compared to other papers. It should be noted, however, that most papers only record the proportion of balls successfully returned to the opponent's half of the table. The return rate is largely dependent on the robot control reliably reaching balls played to the extreme left and right of the table, and is much less dependent on hitting pose and velocity. Our robot nonetheless achieves high return rates. The robot described in Muelling *et al.* (2013) was the only robot to achieve a better return rate for balls served by an oscillating ball machine.

Limitations With so little data, the algorithm will fail if the served balls are too different from each other. Accuracy improves marginally if trained for much more than 200 balls. This is probably related to the measurement and prediction errors of the other system components. The robot performs poorly if trained first on one scenario, and then changing the type of play. Resolving this issue would require a more dynamic exploration strategy, which is one of the challenges we would like to address in addition to those detailed below.

Further work (Continuous Learning) The reinforcement learning approach achieved precise, successful returns in a range of scenarios. In each of these scenarios, the robot was trained from scratch. In a human-like learning process, we would learn continuously with each new experiment. This type of learning would use significantly more data, which might improve overall performance, especially in generalising between different stroke types. There are two main hurdles to overcome for a continuous learning



(a) serve

(b) serve + I-play

(c) serve + V-play

(d) serve + X-play

Figure 4.5: Results of training on the real robot in four scenarios. The experiments always started with a warm-up phase of 30 random actions. The green line represents the running average over the 30 episodes.

Table 4.7: Comparison with other table tennis robots.

Method	Robot model	Stroke type	# of balls	Return rate	Error to target
our	KUKA Agilus	Fixed BM	50	**98%**	**118mm**
Büchler *et al.* (2020)	Muscular Robot	Fixed BM	107	75%	769mm
Zhang *et al.* (2011)	Lab-made	Fixed pos.	-	80%	-
our	KUKA Agilus	Osc. BM	50	88%	**209mm**
Zhao *et al.* (2016)	Custom humanoid	Osc. BM	732	71%	-
Koç *et al.* (2018)	Barrett WAM	Osc. BM	200	80%	-
Muelling *et al.* (2013)	Barrett WAM	Osc. BM	30	**97%**	460mm
our	KUKA Agilus	I-play	50	**96%**	269mm
Muelling *et al.* (2013)	Barrett WAM	I-play	-	88%	-
our	KUKA Agilus	Serve	50	100%	135mm
our	KUKA Agilus	V-play	50	88%	329mm
our	KUKA Agilus	X-play	50	92%	393mm

approach:

Outlier detection. Sometimes the ball hits the edge of the table or net, resulting in an anomalous trajectory. In addition, if the ball hits the edge of the racket, the return will be entirely uncontrolled. The problem becomes even more severe in combination with spin, which can also cause significant deviations when hitting the racket. We do not want to discard these events, however, as the robot needs to be able to handle them.

Another challenge is uneven stroke distribution, which results in very unbalanced data points. Although some stroke and spin types occur very rarely, the robot still needs to learn to handle them. Some ball speeds, for example, will occur much more frequently, but very slow and very fast balls will be underrepresented in the data. In a continuous learning process, the robot's ability to handle balls with a lot of spin should not deteriorate following a sequence of 500 balls with very little spin. Further data imbalance will be introduced as changes in the robot system alter the dynamics of the problem. We have made regular changes to the robot control algorithms. The robot's racket has also been replaced on several occasions. Class imbalance in classification tasks can be effectively solved using under/oversampling, but this approach is not possible for our regression-type tasks. It is difficult to assess if a data point is underrepresented or an outlier.

4.5 Robot Control

The robot used in our system is a floor-mounted KUKA Agilus KR6 R900 sixx. It is capable of linear velocities of over $4m/s$. Unfortunately, the KUKA KR-C4 robot controller is designed for industrial use, and not for a dynamic environment like table tennis. In our setup, we continuously receive target pose and velocity values from the stroke parameter suggestion. The robot needs to start moving to the current suggested hitting position even though the final target pose and velocity are unknown. As the suggestion becomes more and more accurate, this movement needs to be constantly corrected.

There are two methods of communicating with the robot, neither of which perfectly achieves the control described. The first is the KUKA Ethernet KRL Interface (EKI). Target pose and velocity are written to variables in the KUKA robot language (KRL) using TCP requests. A KRL program on the robot controller then executes motions based on these variables. The fastest movements can be achieved using point-to-point (PTP)

commands. Basic PTP movements cannot, however, be corrected during execution. This can be improved by using approximated PTP commands, which involve the robot checking for the next target after executing half of the motion. If no new PTP target is defined, it will complete the motion, otherwise it will continue to the next target and the process is repeated. Control in this strategy is limited, because each PTP correction increases the total execution time, since the robot is always performing half the motion. In Tebbe *et al.* (2018) we nonetheless achieved reasonable precision with only two to four corrections.

The second interface is the KUKA robot sensor interface (RSI), used in Tebbe *et al.* (2021). RSI allows movements to be corrected in real-time using UDP messages. We send a new target position every 4 ms. This gives optimal control in terms of dynamic corrections, but each target position must be attainable within the 4 ms cycle time without exceeding any robot limitations (maximum joint velocity, acceleration, torque, etc.). Should any of the robot limits be exceeded, the robot will stop and terminate the program. With the exception of maximum joint velocity, none of these limits are documented, which makes generating a compliant trajectory difficult. This is the biggest disadvantage of RSI communication.

Ultimately, using RSI we need to iteratively generate the complete robot trajectory. Trajectory planners typically consider the target position only. In our case, however, we also need to achieve a specific velocity. We therefore use the Reflexxes Type IV library (Kröger, 2011). This library allows a target velocity to be specified and guarantees trajectory generation within 100 microseconds. The library can be used in both Cartesian space and joint space.

We performed three experiments to compare the different communication and control systems. First, we tested movements to predefined target end positions. We then set out to attain both a target position and a non-zero velocity simultaneously. Finally, we investigated how well each method copes with dynamic correction.

In the first experiment, the robot moves to three target positions (Fig. 4.6) with an end velocity of zero. Humans use this type of movement for blocks, to return balls with high velocity and spin. We measured the execution time of the movement, i.e., the time between sending the first command and the robot reaching a position with a positional error of less than 3 mm and an orientational error of less than 1 deg compared to the target pose. Results are shown in Table 4.8. The EKI had the shortest execution times in comparison to both RSI methods. The velocity profiles for motions to the second target are plotted in Fig. 4.7. The EKI interface generates a smooth motion that is syn-

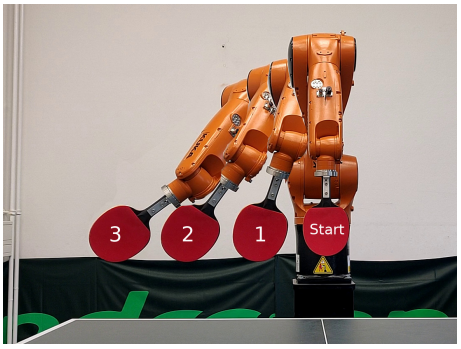


Figure 4.6: Target positions for the three movements. The starting position for all movements is labeled *Start*, the other positions represent Target 1, 2, and 3.

chronized between the joints. The RSI interface controlled in joint space produces a similar movement but with greater jerks. For the RSI in Cartesian space, the velocity profile is highly irregular. Nevertheless, the KUKA controller achieves these Cartesian movements without excessive velocities in joint space.

Table 4.8: Average execution times for the three movements over 5 trials, with standard deviations in parentheses. Cartesian RSI and Joint RSI denote control with the RSI interface using robot trajectories generated by the Reflexxes library in Cartesian and joint space, respectively.

Control method	Target 1 time in ms	Target 2 time in ms	Target3 time in ms
EKI	319.9 (0.3)	342.4 (4.7)	405.6 (3.1)
Cartesian RSI	460.6 (1.3)	471.9 (0.1)	484.0 (0.1)
Joint RSI	412.7 (1.3)	423.9 (0.1)	531.0 (0.1)

The second experiment evaluates non-zero velocities. The end effector should move to Target 2 while attaining a specific velocity in the x-direction. The desired velocities were between 0.2 and 1 m/s toward the opponent's half of the table. Table 4.9 shows the results. The target velocity is specified in the first column. The Cartesian-controlled RSI provides the most accurate velocities, but the maximum attainable speed is limited. The achievable velocities in joint-space RSI depend on the configuration of the robot. At Target 2, a higher maximum velocity can be reached, but the velocity control is less accurate than with the Cartesian RSI. The EKI is only position-based. A heuristic computes two positions, one before and one after the target position, so that the robot attains

the target velocity in between. Even though the heuristic allowed us to increase or lower the velocity, accurate velocity control was not possible.

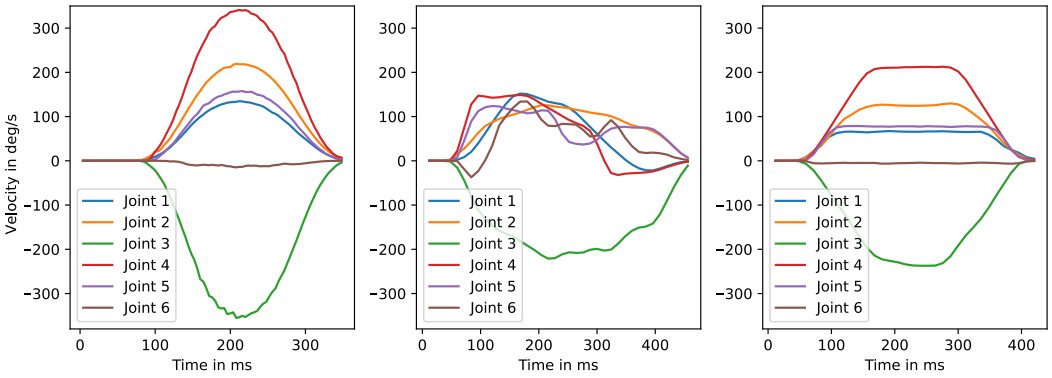
For the target velocity 0.6 m/s, we plotted the robot paths viewed from the top in Fig. 4.8. Both RSI paths hold the desired direction over a much longer time than the EKI interface. That is advantageous since the robot normally hits the ball earlier or later than planned.

In the last experiment, the movements are corrected multiple times. The final target is Target 2. The algorithms are fed with an initial position 100 mm away from Target 2. Over the next 350 ms the target are gradually updated. The results show, that the EKI system is better for only a few corrections, but the execution time increases with the number of corrections. The performance of both RSI methods is nearly independent of corrections, as is expected for methods recomputing the trajectory every 4 ms.

Limitations In the current setup, each method has some drawbacks. The EKI interface provides the fastest motions, and since the KUKA controller computes them, the execution never fails. This method was used for papers Tebbe *et al.* (2018, 2020). One drawback is that correcting the movement afterwards results in longer trajectories. Velocity control can only be achieved by executing multiple positions according to a heuristic. In terms of reaching the target velocity, the used heuristics were inaccurate.

In Tebbe *et al.* (2021), we controlled the robot in Cartesian space with the RSI interface. This ensures that the final movement velocity is controllable. At times, however, movements may exceed the robot's velocity, acceleration, or torque limits. In this case, the robot stops immediately. Positions on the far right or left of the table near workspace limits are affected by this. Large changes in orientation are equally problematic, making backspin strokes impossible to execute. Because the robot limits primarily depend on the joint configuration, we would expect RSI control using joint coordinates to perform better. Unfortunately, it is harder to control the trajectory between start and end poses using joint coordinates. Occasionally, trajectories are generated that would hit the table. Currently, this prevents reliable use of the joint-space RSI on the robot. However, we intend to switch to it in the long run.

Further work Robot control is probably the area that offers the greatest scope for improvement. A current weakness of the EKI communication lies in the velocity control. We need to identify better position sequences to achieve the specified velocity



(EKI) (Cartesian RSI) (Joint RSI)

Figure 4.7: Velocity profiles for movement to Target 2.

Table 4.9: Average Cartesian velocities in x direction (m/s). The target position is Target 2 and the target velocity is specified in the left column.

Target	EKI	Cart. RSI	Joint RSI
1	0.58	0.75	0.96
0.8	0.42	0.75	0.69
0.6	0.27	0.61	0.52
0.4	0.14	0.43	0.33
0.2	0.01	0.25	0.13

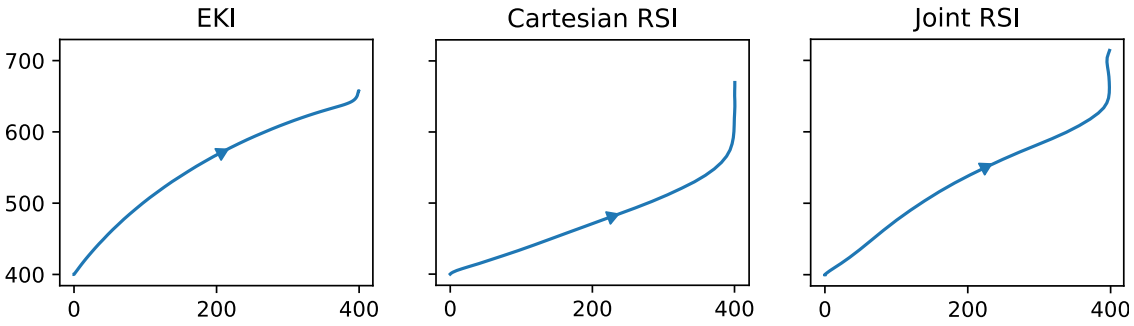


Figure 4.8: Path profiles for movements with 0.6 m/s target velocity. The x-axis is along the short side of the table and the y-axis along the long side towards the opponent's half of the table.

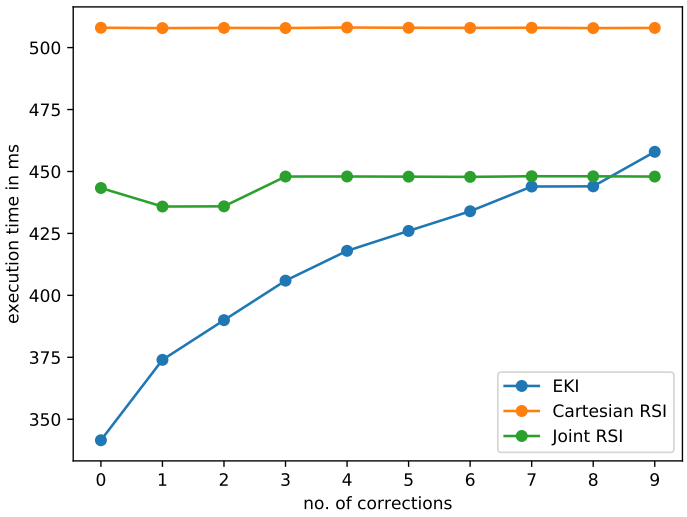


Figure 4.9: EKI and RSI execution times depending on the number of corrections averaged over 5 trials. The initial target and the final corrected target position are 100 mm apart and the execution time is tracked until 3 mm accuracy.

in-between. A realistic simulation of the robot behavior such as KUKA.Sim could be employed to optimize the sequences. Alternatively, position sequences might be learned on the real robot, similar to the RL approach from the previous chapter.

The RSI interface allows arbitrary trajectories to be sent to the robot. We currently provide it with trajectories exclusively in Cartesian space or joint space generated by the Reflexxes library. For accurate executable motions, we should consider both Cartesian and joint space. In our research group, Mario Laux has developed a path planner (Laux and Zell, 2021) to use both spaces in a combined metric and solve for a straight path on this metric using geodesic differential equations. Since the algorithm outputs paths only, it is not possible to set an intended racket velocity. However, we can transform a path with the appropriate final direction via the correct time parameterization into a trajectory that satisfies the velocity constraint.

A more direct method is trajectory optimization. The problem is typically solved with nonlinear programming, which tends to be too slow for real-time control. A near-optimal initial guess provided by the Reflexxes library is expected to accelerate the optimization process. The challenge is to formulate the optimization problem in a way that yields accurate and fast results.

4.6 Qualitative Video Comparison

In this chapter, we compare our robot system qualitatively to other state-of-the-art table tennis robots. This is especially important as qualitative results in this field are biased, because it is not possible to use a common dataset in robotic table tennis. There is significant variation in the balls received by each robot depending on the ball throwing machine or human serving the ball to the robot. Here we give an overview of video demonstrations for other robots. Note that comments are based on the videos and corresponding papers. It is possible that robots are able to cope with other scenarios not shown in the video footage.

MPI Tübingen The first robot we looked at was Mülling *et al.* (2011) at the Max-Planck-Institute in Tübingen. Human demonstrations are first obtained using kinesthetic teaching. By using a mixture of the demonstrated robot movements, the robot is able to learn to return the ball in only 60 trials. In the video[1] the robot faces a human player and has rallies with up to 7 ball contacts. Although the robot is able to learn with very few trials, the demonstration scenario is tightly constrained. Balls are played slowly, with no spin and to only one side of the table. The robot has more time to move, as it plays the ball from a point beyond the end of the table.

MPI Muscular Robot Muscular Robot is another robot developed at MPI Tübingen (Büchler *et al.*, 2020). This is a custom-made pneumatic robot able to perform fast movements with up to 12 m/s. For fast learning they used a mixture of simulated balls and real movement trials. The robot and learning procedure are very novel, but robot performance is limited. The return rate is below 90% even though the ball is always served to the same spot by a ball throwing machine[2]. The robot is, however, able to smash balls with high velocity, a unique skill not possessed by other table tennis robots.

Zhejiang University Xiong *et al.* (2012) have built two humanoid robots, Wu and Kong, able to play against each other or a human. The robots were able to have long rallies with over 100 strokes[3], with the ball played to the middle of the table. In a second video[4] they show the robots receiving balls with side spin from a ball throwing

[1] `https://youtu.be/SH3bADiB7uQ` [2] `https://youtu.be/GQtpSMEpn5A`
[3] `https://youtu.be/t_qN3dgYGqE` [4] `https://youtu.be/endPn8YaL10`

machine (Zhao *et al.*, 2017). The robots use a wooden racket with no rubber, which makes returning balls with spin easier. Results against topspin and backspin would be of more interest, as these spin types are more common in matches against human players.

Various companies have also undertaken industrial table tennis robot projects, often for marketing purposes and to showcase their technologies. These tend to play more realistic table tennis than research projects.

SIASUN SIASUN demonstrated their table tennis robot at the China International Industry Fair[5]. The robot arm is mounted on a linear unit for a greater range. The robot is therefore able to return human strokes played all over the table. The returns are very consistent, but all balls shown have very little spin.

Omron Omron's delta robot is arguably the world's best table tennis robot. Different versions have been shown at various trade fairs worldwide. The robot returns balls played by humans with some spin all around the table very reliably[6]. The only negative point is that the robot seems to be using a racket with no or antispin rubbers. The effect of spin at hitting is much smaller than with standard rackets.

Our robot We now offer a comparison between two demonstrations of our robot and the above robots. The first video[7] clearly shows that the robot is able to return both topspin and backspin strokes. The movements for these two spin types need to be very different, as we use a traditional rubber with high friction. This means that the angle at which the ball bounces off the racket is highly dependent on the spin. The ability to return balls with different types of spin sets our robot apart from almost all other robots. The only robot to perform better in this area is the Omron industrial robot, but it is unclear if it would be able to achieve the same using a racket with normal high-friction rubbers. One drawback in our demonstration is that all balls are played down the middle of the table. However, in our second video demonstration[8], we show that the robot can learn to play a series of increasingly challenging table tennis drills in less than 200 balls. The first drill is relatively simple, but the robot is able to play a long rally with as many as 64 successful

[5] `https://youtu.be/Ov8jwAKucmk`
[6] `https://youtu.be/kZzL2rDNSJk` and `https://youtu.be/EzXxUWQ7H48`
[7] `https://youtu.be/SjE1PtuObTo` [8] `https://youtu.be/uRAtdoL6Wpw`

returns. In the last, most difficult drill, in which both the robot and human play the ball to both sides of the table, the robot is still able to return consistently. Other robots were also able to successfully learn from only a small number of balls. In contrast to these robots, we do not use any prior knowledge from demonstrations or simulations. The scenarios we evaluated were also more challenging, in that our robot was programmed to play the ball to different points on the table, rather than just to the middle of the table, while at the same time adapting to balls played to different points by the human opponent. Overall our robot achieved state-of-the-art robot table tennis skills, in that it adapted to spinning balls, played long rallies and achieved data-efficient learning of robotic return strokes.

Chapter 5

Conclusion

Over the course of this thesis, we developed a fully operational table tennis robot. The system comprises six connected components: ball tracking, spin detection, trajectory prediction, stroke parameter suggestion, trajectory generation and robot control. The research project was focused on two areas. Firstly, we aimed to develop a robot able to play table tennis under more realistic conditions than existing robots. In keeping with the trend towards intelligent systems, we wanted our robot to also be able to learn and to adapt to previously unseen playing styles.

Details of the basic system were published in Tebbe *et al.* (2018). This system included ball detection, trajectory prediction and simple robot control. We achieved robust cooperative play against a human opponent without spin. When playing rallies, the robot achieved up to 64 returns in a row.

When humans play table tennis, spin is a crucial factor. For a more realistic playing style the robot therefore needs accurate, reliable spin detection. Tebbe *et al.* (2020) examined different approaches to spin detection. Two of the methods examined were based on detecting the movement of the logo printed on the ball. A third method involved analysing the ball's trajectory to determine the Magnus effect. All three methods were able to identify the spin type with over 80% accuracy. We performed a demonstration in which we incorporated the Magnus force fitting method into our robot system. This system was able to return real balls with spin served by a human player. Combining these methods might offer a good foundation for returning balls with heavy spin, as played by professional players.

Table tennis also shows the flexibility of human learning. Even basic strokes require complex hand-eye coordination. With some supervision, beginners can learn to return the ball consistently in only a few training sessions. This is a skill we want to imitate

on the robot. To reduce the complexity of the task, we do not input the whole ball trajectory, but only the predicted position and velocity of the ball at the time of hitting. A reinforcement learning method returns the desired racket state at the time of hitting (Tebbe *et al.*, 2021). An appropriate robot trajectory is then generated to achieve that state. Using this method, the robot was able to learn a number of common table tennis drills after training with only 200 balls. Evaluations of table tennis robots in the literature typically involve simpler scenarios. Our robot was able to outperform existing robots for accuracy and consistency in these scenarios.

In Chapter 4, we discussed current limitations of each of the components of our system and presented possible improvements. The ultimate goal, of a robot able to beat even the best human athletes at table tennis, remains a long way off. Nevertheless, this thesis represents a major step in the right direction.

Bibliography

Asai, K., Nakayama, M., and Yase, S. (2019).

Billingsley, J. (1983). Robot ping pong.

Blender Online Community (2016). *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam.

Büchler, D., Guist, S., Calandra, R., Berenz, V., Schölkopf, B., and Peters, J. (2020). Learning to play table tennis from scratch using muscular robots.

Gao, W., Graesser, L., Choromanski, K., Song, X., Lazic, N., Sanketi, P., Sindhwani, V., and Jaitly, N. (2020). Robotic table tennis with model-free reinforcement learning. *arXiv preprint arXiv:2003.14398*.

Gao, Y., Tebbe, J., Krismer, J., and Zell, A. (2019a). Markerless racket pose detection and stroke classification based on stereo vision for table tennis robots. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 189–196.

Gao, Y., Tebbe, J., and Zell, A. (2019b). Real-time 6d racket pose estimation and classification for table tennis robots. *International Journal of Robotic Computing*, **1**(1), 23–39.

Gao, Y., Tebbe, J., and Zell, A. (2021). Robust stroke recognition via vision and imu in robotic table tennis. In *International Conference on Artificial Neural Networks*, pages 379–390, Cham. Springer, Springer International Publishing.

Gomez-Gonzalez, S., Nemmour, Y., Schölkopf, B., and Peters, J. (2019). Reliable real-time ball tracking for robot table tennis. *Robotics*, **8**(4).

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. Technical report.

Huang, Y., Xu, D., Tan, M., and Su, H. (2011). Trajectory prediction of spinning ball for ping-pong player robot. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3434–3439.

Huang, Y., Xu, D., Tan, M., and Su, H. (2013). Adding active learning to lwr for ping-pong playing robot. *IEEE Transactions on Control Systems Technology*, **21**, 1489–1494.

Huynh, D. Q. (2009). Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, **35**(2), 155–164.

Ji, Y.-F., Zhang, J.-W., Shi, Z.-h., Liu, M.-H., and Ren, J. (2018). Research on real – time tracking of table tennis ball based on machine learning with low-speed camera. *Systems Science & Control Engineering*, **6**, 71–79.

Koç, O., Maeda, G., and Peters, J. (2018). Online optimal trajectory generation for robot table tennis. *Robotics and Autonomous Systems*, **105**, 121 – 137.

Kröger, T. (2011). Opening the door to new sensor-based robot applications—the reflexxes motion libraries. In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4.

Laux, M. and Zell, A. (2021). Robot Arm Motion Planning Based on Geodesics. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, Xian, China.

Li, H., Wu, H., Lou, L., Kühnlenz, K., and Ravn, O. (2012). Ping-pong robotics with high-speed vision system. In *2012 12th International Conference on Control Automation Robotics Vision (ICARCV)*, pages 106–111.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Lin, H.-I., Yu, Z., and Huang, Y.-C. (2020). Ball tracking and trajectory prediction for table-tennis robots. *Sensors*, **20**(2).

Mahjourian, R., Jaitly, N., Lazic, N., Levine, S., and Miikkulainen, R. (2018). Hierarchi-

cal policy design for sample-efficient learning of robot table tennis through self-play. *CoRR*, **abs/1811.12927**.

Muelling, K., Kober, J., Kroemer, O., and Peters, J. (2013). Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, **32**(3), 263–279.

Mülling, K., Kober, J., and Peters, J. (2011). A biomimetic approach to robot table tennis. *Adaptive Behavior*, **19**(5), 359–376.

Nakashima, A., Ogawa, Y., Kobayashi, Y., and Hayakawa, Y. (2010). Modeling of rebound phenomenon of a rigid ball with friction and elastic effects. In *Proceedings of the 2010 American Control Conference*, pages 1410–1415.

Peters, J., Mülling, K., and Altün, Y. (2010). Relative entropy policy search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, page 1607–1612. AAAI Press.

Qiao, F. (2021). Application of deep learning in automatic detection of technical and tactical indicators of table tennis. *PLOS ONE*, **16**, e0245259.

Tebbe, J., Gao, Y., Sastre-Rienitz, M., and Zell, A. (2018). A Table Tennis Robot System using an industrial KUKA Robot Arm. In *Pattern Recognition. GCPR 2018*, pages 33–45, Stuttgart, Germany.

Tebbe, J., Klamt, L., Gao, Y., and Zell, A. (2020). Spin Detection in Robotic Table Tennis. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9694–9700, Paris, France.

Tebbe, J., Krauch, L., Gao, Y., and Zell, A. (2021). Sample-efficient Reinforcement Learning in Robotic Table Tennis. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4171–4178, Xian, China.

Xiong, R., Sun, Y., Zhu, Q., Wu, J., and Chu, J. (2012). Impedance control and its effects on a humanoid robot playing table tennis. *International Journal of Advanced Robotic Systems*, **9**(5), 178.

Yu, Z., Liu, Y., Huang, Q., Chen, X., Zhang, W., Li, J., Ma, G., Meng, L., Li, T.,

and Zhang, W. (2013). Design of a humanoid ping-pong player robot with redundant joints. In *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 911–916.

Zhang, Y., Zhao, Y., Xiong, R., Wang, Y., Wang, J., and Chu, J. (2014). Spin observation and trajectory prediction of a ping-pong ball. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4108–4114.

Zhang, Y., Xiong, R., Zhao, Y., and Wang, J. (2015). Real-time spin estimation of ping-pong ball using its natural brand. *IEEE Transactions on Instrumentation and Measurement*, **64**(8), 2280–2290.

Zhang, Y.-h., Wei, W., Yu, D., and Zhong, C.-w. (2011). A tracking and predicting scheme for ping pong robot. *Journal of Zhejiang University SCIENCE C*, **12**(2), 110–115.

Zhang, Z., Xu, D., and Tan, M. (2010). Visual measurement and prediction of ball trajectory for table tennis robot. *IEEE Transactions on Instrumentation and Measurement*, **59**(12), 3195–3205.

Zhao, Y., Xiong, R., and Zhang, Y. (2016). Rebound modeling of spinning ping-pong ball based on multiple visual measurements. *IEEE Transactions on Instrumentation and Measurement*, **65**(8), 1836–1846.

Zhao, Y., Xiong, R., and Zhang, Y. (2017). Model based motion state estimation and trajectory prediction of spinning ball for ping-pong robots using expectation-maximization algorithm. *Journal of Intelligent & Robotic Systems*, **87**(3), 407–423.

Zhu, Y., Zhao, Y., Jin, L., Wu, J., and Xiong, R. (2018). Towards high level skill learning: Learn to return table tennis ball using monte-carlo based policy gradient method. In *2018 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 34–41.

Appendix A

A table tennis robot system using an industrial kuka robot arm

Authors

Jonas Tebbe, Yapeng Gao, Marc Sastre-Rienietz, and Andreas Zell

Published in

German Conference on Pattern Recognition. GCPR 2018.

Time of publication

February 2019

DOI

10.1007/978-3-030-12939-2_3

A Table Tennis Robot System using an industrial KUKA Robot Arm

Jonas Tebbe[1], Yapeng Gao[1], Marc Sastre-Rienietz[1], and Andreas Zell[1]

Cognitive Systems, Eberhard Karls University, Tübingen, Germany
{jonas.tebbe,yapeng.gao,andreas.zell}@uni-tuebingen.de,
marc.sastre-rienitz@student.uni-tuebingen.de
http://www.cogsys.cs.uni-tuebingen.de/

Abstract. In recent years robotic table tennis has become a popular research challenge for image processing and robot control. Here we present a novel table tennis robot system with high accuracy vision detection and fast robot reaction. Our system is based on an industrial KUKA Agilus R900 sixx robot with 6 DOF. Four cameras are used for ball position detection at 150 fps. We employ a multiple-camera calibration method, and use iterative triangulation to reconstruct the 3D ball position with an accuracy of 2.0 mm. In order to detect the flying ball with higher velocities in real-time, we combine color and background thresholding. For predicting the ball's trajectory we test both a curve fitting approach and an extended Kalman filter. Our robot is able to play rallies with a human counting up to 50 consequential strokes and has a general hitting rate of 87%.

Keywords: table tennis robot · ball detection · trajectory prediction

1 Introduction

In March 2014 KUKA was very successful with a commercial video of the best german table tennis player Timo Boll facing a KUKA Agilus R900 sixx robot [8]. As the robot could not really play table tennis, the producers used modern video editing to give "a realistic vision of what robots can be capable of in the future." Shortly afterward another video [11] went viral, showing a self-built robot arm playing table tennis in a garage. Several inconsistencies pointed to manipulation of the video. With the same KUKA robot as in the first video we want to test the current limits of real robotic table tennis.

1.1 Related Work

Robotic table tennis has been attracting many researchers in the domain of image processing and robot control since Billingsley [5] in 1983 announced a robot table tennis competition with 20 rules for improving the successful hitting rate. Following these rules, Anderson [3] developed the first robot able to play with human ping-pong players. He employed four cameras to detect the incoming

ball and a 6-DOF PUMA 260 arm to return the ball. Other early robot systems were designed to continue this work with some limitations like using low frame rate camera or playing just in a well defined environment [7,1,16].

In recent years, there have been multiple robots of different accuracy defining the current state of the art. Xiong et al. [28] developed two humanoid robots named Wu & Kong, which can rally with each other more than 100 rounds. Each humanoid robot has 30 DOF in total, which are composed of two 7-DOF arms, two 6-DOF legs, and 4-DOF for head and waist. Every robot was equipped with four cameras to detect the ball. Muelling et al. [17,18,19] learned a generalized stroke motion from different elementary hitting movements taught by a human tutor. They used an anthropomorphic 7-DOF robot arm, which could generate smooth hitting movements capable of adapting to the changes in ball speed and position. Nakashima et al. [21,22,20] designed the joint trajectory of a 7-DOF robot arm which hits a back-spin ball to target points at the speed of 2-3 m/s. The success hitting rate was 70% in 20 trials. Li et al. presented a robot system in [14] consisting of two high-speed cameras and a universal 7-DOF robot arm. He applied a novel two-phase trajectory prediction to determine the hitting position, and finally achieved a success rate of 88%. Silva et al. [25] attached a light card-board racket and an onboard camera to a quadrotor drone and utilized imitation learning adopted from [19] to reach hitting rates of 30% in simulation and 20% with a real quadrotor. There are also some industrial companies following the same work. Omron [12] has shown an impressive demonstration at various robotics fairs. The Forpheus robot repurposed from an Omron 5-axis parallel robot, seems to have the best hitting accuracy at the moment and as a robotic tutor it can evaluate the ability level of its opponent with deep learning. SIASUN [24] exhibited its table tennis robot named Pongbot. It can predict the hitting position within 4 ms with 20 mm error using a high-speed stereo camera, and they developed a 6-DOF flexible robot arm to hit the ball back with an intelligent strategy.

1.2 Our System

In this paper, we present a novel table tennis robot system (Figure 1) with high accuracy vision detection and fast robot reaction. Four PointGrey Chameleon3 cameras which are mounted in the corners of the ceiling are used for ball position detection. A table tennis paddle is rigidly fixed at the end-effector of a 6-DOF KUKA Agilus robot, and two Sick S300 laser scanners mounted on the floor are used in a safety system for area protection and access prevention. The whole system is controlled by one host computer with a 4 x 3.3GHz Intel i5-4590 CPU and 16GB RAM. We implement the proposed approaches using the Robot Operating System(ROS) [23] nodes with message communication between each other and OpenCV [6] for image processing.

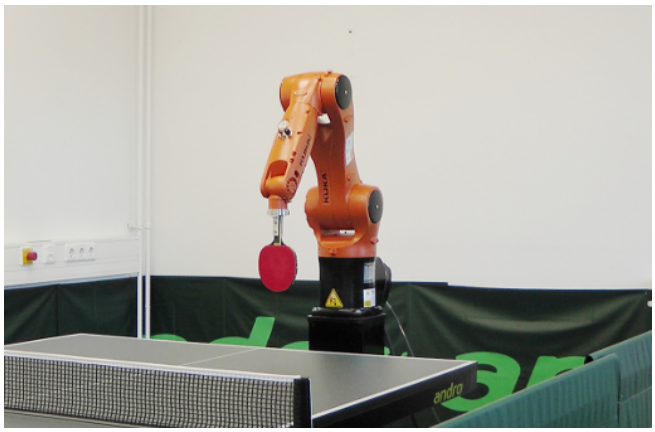


Fig. 1. Table tennis robot system with KUKA Agilus robot. The cameras, which are mounted on the ceiling, are not visible.

2 Methods

2.1 Ball Detection

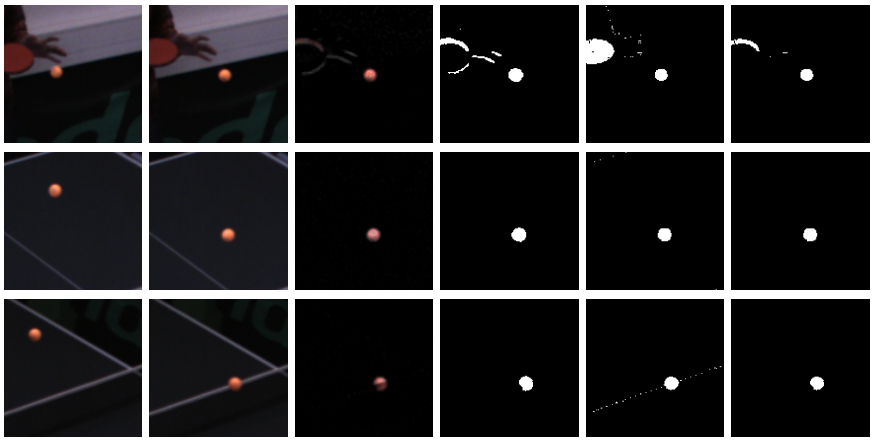
There is a ROS ball detection node which outputs the 3D ball positions. The method adopted in this paper is fusing multiple features including motion, color, area and shape, which are used in [14,29]. To improve the robustness of image segmentation, we transfer the raw images read from the cameras into HSV color space. Multiple CPU threads are generated for each camera to accelerate the processing. Figure 2 shows the ball detection process using motion and color features for three examples cropped to the regions of interest. The top row depicts a position close to racket and player's hand. The middle row details a state on the table and in the last row the ball is crossing the net.

We want to avoid the crescent shaped ball shown in [34] when using adjacent frame difference because of a slow ball and high frame rate. Therefore we store the images in a queue and compare the current (n-th) to the $n-7$th image, which is shown in Figure 2(a)-(c). The binary shown in Figure 2(d) uses thresholding according to the following equation:

$$Binary_{\mathrm{n}}(u,v) = \begin{cases} 255 & \text{if } L \leqslant HSV_{\mathrm{n}}(u,v) - HSV_{\mathrm{n\text{-}7}}(u,v) \leqslant U \\ 0 & otherwise \end{cases} \quad (1)$$

$HSV_n(u,v)$ is the vector of HSV values of the pixel (u,v) in the n^{th} image and comparison is done component-wise. L and U are the lower and upper HSV boundary values which are selected manually.

Restricting the setup to orange table tennis balls we are able to get the benefit of color thresholding the n^{th} frame, which results in Figure 2(e). By means of computing the bitwise conjunction of (d) and (e) in Figure 2(f), we can extract the orange moving objects including balls and possibly skin regions, the moving



(a) (b) (c) (d) (e) (f)

Fig. 2. Ball detection process using motion and color features: (a): the $(n-7)^{th}$ frame. (b): the n^{th} frame. (c): subtracting frame (a) from (b). (d): color thresholding of (c). (e): color thresholding of (b). (f): bitwise AND operation between (d) and (e).

bat and the robot. To extract the correct blob belonging to the ball, we exploit size and shape features to filter out non-ball objects described as follows:

$$\begin{cases} 10px \leqslant & Area & \leqslant 800px \\ 0.5 \leqslant & AreaExtent & \leqslant 1 \\ 1/1.4 \leqslant & AspectRatio & \leqslant 1.4 \end{cases} \tag{2}$$

where *Area* is the contour area extracted from the Figure 2(f) in pixels. *AreaExtent* is the ratio of *Area* to the area of the minimal containing up-right bounding box. *AspectRatio* is the aspect ratio of the bounding box. In rare cases this process results in multiple candidates because of other moving objects with similar properties existing. Therefore we select the one with the largest area as the detected ball.

Once a ball is recognized in the current frame, a region of interest (ROI) will be computed around the ball's center. Then we can track the next ball in this ROI within 5 ms/image. Processing in parallel gives real-time performance for a frame rate of 150fps.

The balls central pixel positions in multiple images are undistorted and triangulated to a 3d position according to calibration and triangulation described in section 3.1.

2.2 Ball Trajectory Prediction

Given the observation of the positions of the ball until the current time, we need to predict the future trajectory of the ball to plan the hitting stroke of our robot. The flying ball is exposed to three forces: Gravity, air resistance, and the Magnus force (caused by spin) as depicted in Figure 3. Even for humans this

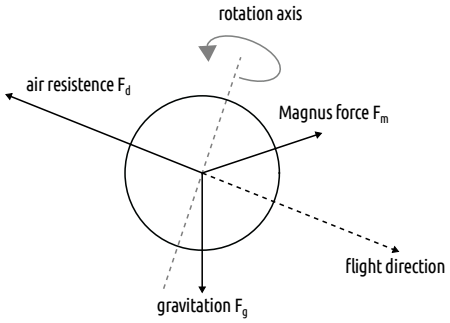


Fig. 3. Acting forces on a ball: gravitation pointing downwards, air resistance in the opposite direction of flight and Magnus force perpendicular to the spin axis and flight direction.

prediction is difficult and requires years to get a good estimation of balls with heavy spin. This is in particular due to the difficulty of measuring spin. Various research has been done on this topic. In [18] a physical model was used. The model in [32] could learn its parameters using a neural network. Many systems use an extended or unscented Kalman filter [18,31,30,27] or curve fitting [34,14]. For our system we employed both an extended Kalman filter and a curve fitting approach.

Curve Fitting For the curve fitting approach the 3d trajectory points are separated for every axis. In our case the x-axis is along the large side of table, the y-axis is parallel to the net and the z-axis gives the height of a point. A quadratic polynomial $P(t) = at^2+bt+c$ is fitted to the data for each axis, where the input is the time and the output is the ball's position on the specific axis. Its coefficients can be easily solved as this is a linear least squares problem. These polynomials describe the flight before bouncing on the table. With the roots of the z-axis polynomial we have the bounce time t_{bounce}. For the post-bounce trajectory we derive another set of polynomials Q. For the x-axis we define factors s_x and a_x for the change of velocity and acceleration in x-direction after the bounce. Without considering spin on the bounce we assume the post-bounce polynomial Q_x satisfies

$$Q_x(t_{\text{bounce}}) = P_x(t_{\text{bounce}}) \tag{3}$$
$$Q'_x(t_{\text{bounce}}) = s_x * P'_x(t_{\text{bounce}}) \tag{4}$$
$$Q''_x(t_{\text{bounce}}) = a_x * P''_x(t_{\text{bounce}}). \tag{5}$$

Analogously we have conditions for the other polynomials. With these conditions we can find unique solutions. The hitting point defined by the intersection with the plane $x = 0.1$ (in meters) can be solved from the post-bounce polynomials. The values used in our experiments are $s_x = s_y = 0.73$, $s_z = -0.85$, $a_x = a_y =$

0.3 and $a_z = 0.92$. Note that s_z is negative to affect for bouncing off the table in the opposite direction.

A disadvantage of the curve fitting approach is the large error for the first estimations, where only 5-10 ball positions have been recorded. To overcome this we suggest to use Tikhonov regularization for least-square solving the polynomial while regularizing the curvature. For a linear least squares problem $Ax = b$ we have an optimal solution

$$\hat{x} = (A^T A)^{-1} A^T b. \tag{6}$$

In Tikhonov regularization one defines a Tikhonov matrix Γ and

$$\hat{x} = (A^T A + \Gamma^T \Gamma)^{-1} A^T b. \tag{7}$$

For a typical Tikhonov matrix λI_n the solution minimizes the error while regularizing the norm of the solution. In our case we only want to regularize the acceleration $P''(t)$, so we use the matrix

$$\Gamma = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \lambda \end{bmatrix}. \tag{8}$$

Using the z-polynomial as an example we approximate $(z_i - a_{\text{prior}} t_i^2)_i$ with the Tikhonov matrix Γ. In this case the curvature a of the polynomial has a penalty to differ from 0. Therefore replacing the curvature a by $a' = a + a_{\text{prior}}$ results in a polynomial with a curvature near a_{prior} approximating the values $(z_i)_i$. We use the curvature from prior experiments which includes an approximation for gravity and air drag. This can significantly improve the first predictions. With a smaller λ for more ball position measurement the curve fitting is drawn less to the prior curvatures and more to the real one. Using Tikhonov regularization we could lower the average bounce point error after 5 balls from unreasonable values to 70 mm, which is acceptable for initially moving the robot arm in the right direction.

EKF approach Relative to the balls speed v and angular velocity ω the gravitation force F_g, drag force F_d and Magnus force F_m in Figure 3 are defined by the formulas

$$F_g = (0, 0, -mg)^T. \tag{9}$$

$$F_d = -\frac{1}{2} C_D \rho_a A \left\| v \right\| v \tag{10}$$

$$F_m = \frac{1}{2} C_M \rho_a A r (\omega \times v). \tag{11}$$

The constants appearing are drag coeficienthe mass of the ball $m = 2.7\text{g}$, the gravitational constant $g = 9.81\text{m/s}^2$, the drag coefficient $C_D = 0.4$, the density of the air $\rho_a = 1.29\text{kg/m}^3$, the lift coefficient $C_M = 0.6$, the ball radius $r = 0.02\text{m}$,

and the ball's cross-section $A = r^2\pi$. A discrete motion model is defined by

$$p_{t+1} = f(p_t) = \begin{pmatrix} x + v_x \Delta t \\ y + v_y \Delta t \\ z + v_z \Delta t \\ v_x - k_D \Delta t \, \|v\| \, v_x + k_M \Delta t (\omega_y v_z - \omega_z v_y) \\ v_y - k_D \Delta t \, \|v\| \, v_y + k_M \Delta t (\omega_z v_x - \omega_x v_z) \\ v_z - k_D \Delta t \, \|v\| \, v_z + k_M \Delta t (\omega_x v_y - \omega_y v_x) - g \Delta t \end{pmatrix}. \tag{12}$$

Here we shorten the coefficients to $k_D = \frac{1}{2} C_D \rho_a A$ and $k_M = \frac{1}{2} C_M \rho_a A r$. The state is defined as $p_t = (x, y, z, v_x, v_y, v_z)$ and can be estimated by an extended Kalman filter [31]. Using the estimated state we can predict the future trajectory using the discrete model. At the bounce point the speed and angular velocity from directly before the bounce denoted by $*^-$ are transformed to $*^+$ as follows

$$v_x^+ = \alpha_x v_x^- + \beta_x \omega_y^- \tag{13}$$
$$v_y^+ = \alpha_y v_y^- + \beta_y \omega_x^- \tag{14}$$
$$v_z^+ = -\alpha_z v_z^- \tag{15}$$
$$\omega_x^+ = \gamma_x \omega_x^- + \delta_x v_y^- \tag{16}$$
$$\omega_y^+ = \gamma_y \omega_y^- + \delta_y v_x^- \tag{17}$$
$$\omega_z^+ = \gamma_z \omega_z^-. \tag{18}$$

The values used in our experiments are $\alpha_x = \alpha_y = 0.75$, $\alpha_z = -0.97$, $\beta_x = \beta_y = 0.0015$, $\gamma_x = 0.53$, $\gamma_y = 0.6$, $\gamma_z = 0.9$, $\delta_x = -26$ and $\delta_y = 25$ according to [31].

A dataset is recorded with 50 balls played in by a human player with low spin, similar to the ones shown in the video. Our results on the data are shown in Table 1 and 2. For the accuracy after the first quarter (first half) 14.5 balls (32.2 balls) are used on average. For this data we are not considering spin, which is difficult to measure and is part of further research, such that the angular velocity ω used in the EKF is zero. In the future a constant spin value measured by another system can be integrated.

In evaluation the EKF and curve fitting are on par with each other. If only ball position from the first quarter are given the EKF gives better results and with positions from the first half the curve fitting is more accurate. The EKF may have the advantage to use a physical model, but ball positions early in time are getting out of the scope. In contrast, the curve fitting uses all available data to fit the curve. Therefore, it gets more accurate with more ball positions. It can also cope with small amounts of spin as such a trajectory can also be approximated by a parabola.

2.3 Robot Control

The robotic arm in the system (KUKA Agilus Kr6 R900 sixx) is a high speed industrial robot with six DOF, mounted on the floor and capable of achieving linear velocities of over 4m/s.

Table 1. Bounce and hitting point errors on balls measured until the first quarter of the table and until the first half of the table

	After first quarter		After first half	
/mm	Error	Stddev	Error	Stddev
Bounce point curve fitting	72.7	32.3	19.9	13.7
Bounce point EKF	46.6	29.4	27.8	15.2
Hitting point curve fitting	69.9	25.6	18.3	11.1
Hitting point EKF	55.3	22.8	34.7	13.2

Table 2. Bounce and hitting time errors on balls measured until the first quarter of the table and until the first half of the table

	After first quarter		After first half	
/ms	Error	Stddev	Error	Stddev
Bounce time curve fitting	21.8	5.5	6.5	2.7
Bounce time EKF	26.8	5.0	15.3	3.9
Hitting time curve fitting	64.9	9.2	8.4	7.3
Hitting time EKF	45.4	24.3	27.1	9.6

The robot is controlled by its own computer, integrated in the KR-C4 controller, which provides an interface to the user. The user of the robot can then move it manually at low velocities or move it via a program written in the KRL programming language. While we could not use the industrial controller for frequent dynamic trajectory changes, we used approximate movements described below for a similar effect for up to 3 end-effector target point changes.

We can establish a connection to the controller PC via the network using the KUKA EthernetKRL package, which allows to read or write KRL variables from an external PC via an Ethernet connection. Figure 4 shows a diagram of the communication system. The delay in sending or receiving data via the EthernetKRL connection depends on the amount of read/write function calls on the robot side (about 4 to 12 ms per call). In order to minimize the delay, we put all the data of a message in a single array so that we only need one function call on the robot computer.

We use two kinds of motions: exact and approximate. Exact motions move the robot exactly to a specified point, whilst approximate motions move the robot in the direction of the specified point, as if it were an exact motion, but changing direction to the next point in the trajectory when a certain limit distance to the initial point is achieved (see Figure 5). Approximated motions allow for the trajectory of the robot to change before it arrives to the initial target. Thus we are able to send an initial guess of the hitting point ahead of time and then send corrections with more precise information of the hitting point as we get to see more of the actual trajectory of the ball.

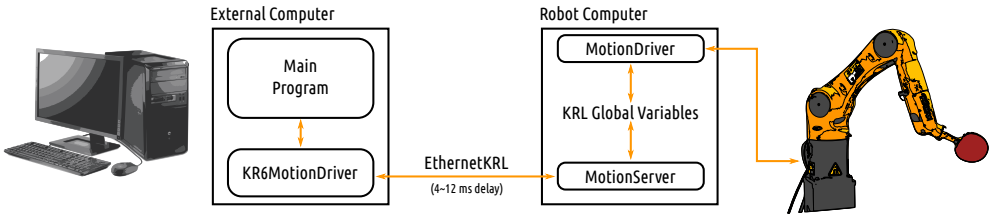


Fig. 4. To control the robot, two programs run on the robot controller: the `MotionServer`, which runs on the Submit Interpreter, handles the connection and is responsible for reading or writing the data that the connected PC requests, while the other program called `MotionDriver`, running with higher priority, is the one that executes all movement commands. On the external PC side, the C++ class called `KR6MotionDriver` communicates with the `MotionServer` on the robot side.

3 Experiments

3.1 Multi-camera Calibration

To achieve high accuracy for ball position detection, we propose a method for multi-camera calibration. Our multi-camera calibration is different from [13] where they divided four cameras as two camera pairs and reconstructed the ball's 3D position from one pair. Our system can estimate the 3D position from more than two cameras which has the same idea as [15]. We first estimate the camera intrinsics, distortion coefficients based on [33] with a 4×11 asymmetric circle grid pattern. We use the stereo calibration method in [9] to initialize the extrinsic camera parameters, which denote the coordinate system transformations from three slave cameras to the master camera. To optimize these extrinsic matrices simultaneously, we place the pattern at different locations and orientations in overlapping fields of view of all cameras in order to extract the same centers of circular blobs for every camera. Then, we employ a modified sparse bundle adjustment approach [26], which minimizes the following error function:

$$E(P, X) = \sum_{i=1}^{m} \sum_{j=1}^{n} \|P_j X_i - x_{ij}\|^2 + \lambda \sum_{i=1}^{m} | \, \|X_{i+1} - X_i\| - D|^2 \qquad (19)$$

where P_j is the estimated projection transformation composed of the camera intrinsics and extrinsics. X_i is the center's position in the 3D scene, which is reprojected to the image plane by P_j. x_{ij} is the observed 2D image coordinate. The second error term accounts for the 3D distance D between two circular blobs, which is not used in the normal bundle adjustment. A factor λ is added to account for the different units (pixels, mm). This equation is solved by the Ceres Solver library [2].

If a ball is found in every camera image, we need to find its 3d position. We use an iterative N-view triangulation method, as in [10], instead of the linear least-squares triangulation, which is performed in OpenCV [6]. For every camera, the linear triangulation solves the homogeneous equation $\alpha(u, v, 1)^\intercal = PX$, where

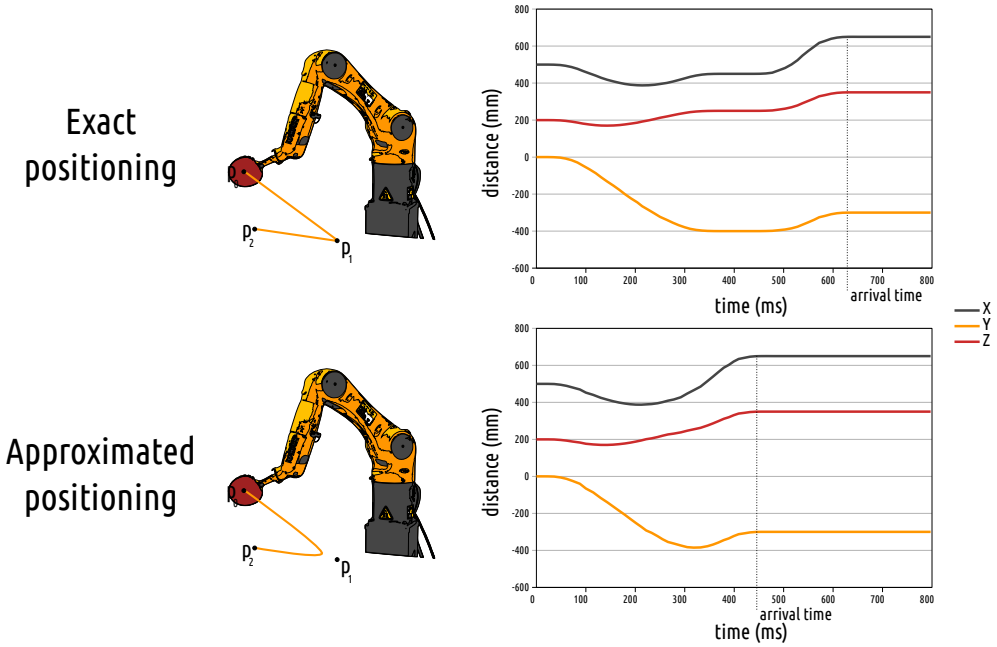


Fig. 5. On the left side we can see the difference between the spatial trajectories of the robotic motion when the point p_1 is reached exactly vs. approximately. On the right side, we plot two real trajectories between the points $p_0 = (500, 0, 200)$, $p_1 = (450, -400, 250)$ and $p_2 = (650, -300, 350)$. When p_1 is reached using exact positioning, the robot lowers the velocity extremely near p_1 giving the impression that it stays still for a little moment and reaching the point p_2 about 175 ms later than with approximate positioning.

α is an unknown scale factor and $(u, v)^\intercal$ are the ball's pixels in the image, and P is the projection matrix of the camera. This is the same as solving:

$$uP_3X = P_1X, \quad vP_3X = P_2X, \quad \alpha = P_3X \tag{20}$$

where P_i denotes the i-th row of P. So in linear least square triangulation the u-error $e_u = uP_3X = P_1X$ is minimized and similar the v-error. Ideally we would like to minimize the error $e'_u = u - P_1X/P_3X$ where P_1X/P_3X is the reprojection of X to the image plane. To come nearer to the ideal solution [10] iteratively solves $X = X_i$ from the error function $e'_u = e_v/w_i$ where $w_0 = 1$ and $w_i = P_3X_i$. Likewise, it is done for v and all other cameras.

The system's accuracy is evaluated by mounting a table tennis ball on the end-effector and comparing against the robot's end-effector localization including the difference vector from end-effector to fixed ball. In this fashion we capture 40 static locations of the ball with both systems resulting in two 3D points sets. The two systems operate in different coordinate systems and a coordinate transformation is estimated using the two 3D points sets according to [4]. The camera calibration errors are shown in Table 3, which includes two tests using both

two and four cameras. Adopting our multi-camera calibration and linear least-squares triangulation we achieve an error of 2.5 mm for four cameras. Iterative triangulation improves the error to 2.0 mm.

Table 3. Calibration errors comparison in mm

	Stereo Calibration	Multi-View Calibration	Iterative Triangulation
Two Cameras	11.0	3.2	3.2
Four Cameras	15.0	2.5	2.0

3.2 Cooperative play against a human

Even without considering spin we were able to play consistent rallies against a human player. In the experiment the player plays a simple stroke using the counter-hitting technique. For predicting the table tennis ball trajectory we use the extended Kalman Filter. An initial position near the base line is sent to the robot at 400 ms before the predicted hitting time. The actual stroke is sent 300 ms before hitting. It consists of a two movements, first to a preparation position and then to the hitting point. In that way we can cope with the lack of velocity control and hit the ball at the predicted hitting point with an approximated velocity between 2 and 4 m/s. On a total of 315 strokes our robots was able to return 87% of the balls back to the table. A video of several such rallies can be found on our YouTube channel (https://youtu.be/AxSyXMbV3Yg).

4 Conclusion and Future Work

In this paper we present a new table tennis robot system. Against balls played from a human with little topspin or no spin, the robot is very successful. In a next step we need to adapt to different spin types. Using a fifth high-speed camera we already started to detect the rotation of the table tennis ball using its logo brand. In the future we will use the system to make the robot play different types of strokes depending on the ball's spin.

Acknowledgment

This work was supported in part by the Vector Stiftung and KUKA.

References

1. Acosta, L., Rodrigo, J.J., Mendez, J.A., Marichal, G.N., Sigut, M.: Ping-pong player prototype. IEEE Robotics Automation Magazine **10**(4), 44–52 (Dec 2003). https://doi.org/10.1109/MRA.2003.1256297

2. Agarwal, S., Mierle, K., Others: Ceres solver. http://ceres-solver.org
3. Anderson, R.L.: A Robot Ping-pong Player: Experiment in Real-time Intelligent Control. MIT Press, Cambridge, MA, USA (1988)
4. Arun, K.S., Huang, T.S., Blostein, S.D.: Least-squares fitting of two 3-d point sets. IEEE Transactions on Pattern Analysis and Machine Intelligence **PAMI-9**(5), 698–700 (Sept 1987). https://doi.org/10.1109/TPAMI.1987.4767965
5. Billingsley, J.: Robot ping pong (05 1983)
6. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
7. Fssler, H., A. Beyer, H., T. Wen, J.: Robot ping pong player. optimized mechanics, high performance 3d vision, and intelligent sensor control **6**, 161–170 (09 1990)
8. Group, K.R.: The duel: Timo boll vs. kuka robot (03 2014), https://www.youtube.com/watch?v=tIIJME8-au8
9. Hartley, R.I., Zisserman, A.: Multiple View Geometry in Computer Vision. Cambridge University Press, ISBN: 0521540518, second edn. (2004)
10. Hartley, R.I., Sturm, P.: Triangulation. Comput. Vis. Image Underst. **68**(2), 146–157 (Nov 1997). https://doi.org/10.1006/cviu.1997.0547, http://dx.doi.org/10.1006/cviu.1997.0547
11. Hoffmann, U.: Mann gegen maschine - ulf hoffmann tischtennis roboter (uhttr-1) (03 2014), https://www.youtube.com/watch?v=imVNg9j7rvU
12. Kawakami, S., Ikumo, M., Oya, T.: Omron table tennis robot forpheus, https://www.omron.com/innovation/forpheus.html
13. Lampert, C.H., Peters, J.: Real-time detection of colored objects in multiple camera streams with off-the-shelf hardware components. Journal of Real-Time Image Processing **7**(1), 31–41 (Mar 2012). https://doi.org/10.1007/s11554-010-0168-3
14. Li, H., Wu, H., Lou, L., Khnlenz, K., Ravn, O.: Ping-pong robotics with high-speed vision system. In: 2012 12th International Conference on Control Automation Robotics Vision (ICARCV). pp. 106–111 (Dec 2012). https://doi.org/10.1109/ICARCV.2012.6485142
15. Lourakis, M., Argyros, A.: sba: A generic sparse bundle adjustment c/c++ package based on the levenberg-marquardt algorithm. t (01 2008)
16. Miyazaki, F., Matsushima, M., Takeuchi, M.: Learning to Dynamically Manipulate: A Table Tennis Robot Controls a Ball and Rallies with a Human Being, pp. 317–341. Springer Berlin Heidelberg, Berlin, Heidelberg (2006). https://doi.org/10.1007/978-3-540-37347-6_15
17. Müling, K., Kober, J., Peters, J.: A biomimetic approach to robot table tennis. Adaptive Behavior **19**(5), 359–376 (2011). https://doi.org/10.1177/1059712311419378
18. Mülling, K., Kober, J., Peters, J.: Simulating human table tennis with a biomimetic robot setup. In: Doncieux, S., Girard, B., Guillot, A., Hallam, J., Meyer, J.A., Mouret, J.B. (eds.) From Animals to Animats 11. pp. 273–282. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
19. Mlling, K., Kober, J., Kroemer, O., Peters, J.: Learning to select and generalize striking movements in robot table tennis. The International Journal of Robotics Research **32**(3), 263–279 (2013). https://doi.org/10.1177/0278364912472380
20. Nakashima, A., Ito, D., Hayakawa, Y.: An online trajectory planning of struck ball with spin by table tennis robot. In: 2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics. pp. 865–870 (July 2014). https://doi.org/10.1109/AIM.2014.6878188
21. Nakashima, A., Ogawa, Y., Kobayashi, Y., Hayakawa, Y.: Modeling of rebound phenomenon of a rigid ball with friction and elastic effects. In: Pro-

ceedings of the 2010 American Control Conference. pp. 1410–1415 (June 2010). https://doi.org/10.1109/ACC.2010.5530520
22. Nakashima, A., Nonomura, J., Liu, C., Hayakawa, Y.: Hitting back-spin balls by robotic table tennis system based on physical models of ball motion. IFAC Proceedings Volumes **45**(22), 834 – 841 (2012). https://doi.org/https://doi.org/10.3182/20120905-3-HR-2030.00107, http://www.sciencedirect.com/science/article/pii/S1474667016337132, 10th IFAC Symposium on Robot Control
23. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: Ros: an open-source robot operating system. In: Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics. Kobe, Japan (May 2009)
24. SIASUN: Siasun table tennis robot pongbot, https://youtu.be/Ov8jwAKucmk
25. Silva, R., Melo, F.S., Veloso, M.: Towards table tennis with a quadrotor autonomous learning robot and onboard vision. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 649–655 (Sept 2015). https://doi.org/10.1109/IROS.2015.7353441
26. Triggs, B., McLauchlan, P.F., Hartley, R.I., Fitzgibbon, A.W.: Bundle adjustment — a modern synthesis. In: Triggs, B., Zisserman, A., Szeliski, R. (eds.) Vision Algorithms: Theory and Practice. pp. 298–372. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
27. Wang, Q., Zhang, K., Wang, D.: The trajectory prediction and analysis of spinning ball for a table tennis robot application. In: The 4th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent. pp. 496–501 (June 2014). https://doi.org/10.1109/CYBER.2014.6917514
28. Xiong, R., Sun, Y., Zhu, Q., Wu, J., Chu, J.: Impedance control and its effects on a humanoid robot playing table tennis. International Journal of Advanced Robotic Systems **9**(5), 178 (2012). https://doi.org/10.5772/51924, https://doi.org/10.5772/51924
29. Zhang, H., Wu, Y., Yang, F.: Ball detection based on color information and hough transform. In: 2009 International Conference on Artificial Intelligence and Computational Intelligence. vol. 2, pp. 393–397 (Nov 2009). https://doi.org/10.1109/AICI.2009.21
30. Zhang, Y., Xiong, R., Zhao, Y., Wang, J.: Real-time spin estimation of ping-pong ball using its natural brand. IEEE Transactions on Instrumentation and Measurement **64**(8), 2280–2290 (Aug 2015). https://doi.org/10.1109/TIM.2014.2385173
31. Zhang, Y., Zhao, Y., Xiong, R., Wang, Y., Wang, J., Chu, J.: Spin observation and trajectory prediction of a ping-pong ball. In: 2014 IEEE International Conference on Robotics and Automation (ICRA). pp. 4108–4114 (May 2014). https://doi.org/10.1109/ICRA.2014.6907456
32. Zhang, Y., Xiong, R., Zhao, Y., Chu, J.: An adaptive trajectory prediction method for ping-pong robots. In: Proceedings of the 5th International Conference on Intelligent Robotics and Applications - Volume Part III. pp. 448–459. ICIRA'12, Springer-Verlag, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33503-7_44
33. Zhang, Z.: A flexible new technique for camera calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence **22**(11), 1330–1334 (Nov 2000). https://doi.org/10.1109/34.888718
34. Zhang, Z., Xu, D., Tan, M.: Visual measurement and prediction of ball trajectory for table tennis robot. IEEE Transactions on Instrumentation and Measurement **59**(12), 3195–3205 (Dec 2010). https://doi.org/10.1109/TIM.2010.2047128

Appendix B

Spin Detection in Robotic Table tennis

Authors

Jonas Tebbe, Lukas Klamt, Yapeng Gao, and Andreas Zell

Published in

2020 IEEE International Conference on Robotics and Automation (ICRA)

Time of publication

September 2020

DOI

10.1109/ICRA40945.2020.9196536

Spin Detection in Robotic Table Tennis*

Jonas Tebbe[1], Lukas Klamt[1], Yapeng Gao[1], and Andreas Zell[1]

***Abstract*— In table tennis, the rotation (spin) of the ball plays a crucial role. A table tennis match will feature a variety of strokes. Each generates different amounts and types of spin. To develop a robot that can compete with a human player, the robot needs to detect spin, so it can plan an appropriate return stroke. In this paper we compare three methods to estimate spin. The first two approaches use a high-speed camera that captures the ball in flight at a frame rate of 380 Hz. This camera allows the movement of the circular brand logo printed on the ball to be seen. The first approach uses background difference to determine the position of the logo. In a second alternative, we train a CNN to predict the orientation of the logo. The third method evaluates the trajectory of the ball and derives the rotation from the effect of the Magnus force. This method gives the highest accuracy and is used for a demonstration. Our robot successfully copes with different spin types in a real table tennis rally against a human opponent.**

I. INTRODUCTION

One of the most difficult tasks when playing table tennis is judging the amount of spin on a ball. To achieve the goal of beating human players of different levels, a table tennis robot needs to be able to accurately predict spin. A lot of prior knowledge is required to assign the right spin to a shot. The major factor used by human players to judge spin is the opponent's stroke. It is, however, difficult to detect stroke movement with a camera. Such an approach would also require training with a number of different people and rackets.

German table tennis professional Timo Boll has excellent eyesight and claims to see the rotation of the ball from the movement of the brand logo [1]. By recording the ball with high-speed cameras, it is possible to identify markers on the ball and detect its rotation. This is the most common approach in the literature. Tamaki et al. [2] use black lines on the ball for tracking. Zhang et al. [3], [4] use the logo printed on the ball.

Another promising approach is to use measurements of the ball's trajectory to determine spin. Huang et al. [5] used a similar approach, involving a physical force model which included the Magnus force, to determine the rotation of the ball. Zhao et al. [6], [7] replace the norm of the velocity necessary to calculate the air resistance. Thus, a differential equation can be solved and one can fit the speed and spin values. Blank et al. [8] capture stroke motion using an IMU mounted on the bat to predict the rotation of the ball. Gao et al. [9] track the table tennis bat using stereo cameras and use a neural network to classify the different types of strokes.

Fig. 1. Spin Detection used to return balls with high rotation on a real robot. Supplementary Video: `https://youtu.be/SjE1Ptu0bTo`

Our goal is to develop a table tennis robot that can successfully return spin strokes from a human opponent. To achieve this we introduce and evaluate three different methods for spin detection using the movement of the ball's logo or its trajectory. Our key contributions are summarized by the following:

- State of the art spin detection using logo extraction is improved by circular segment fitting.
- A CNN is trained on ball images outperforming standard logo extraction methods. For training and evaluation of the CNN a dataset of 4656 images with manual logo orientation label is created.
- Fitting the different forces to the ball trajectory gives a robust spin estimation independent of the ball's logo.
- The trajectory fitting is employed on a real robot system and gives convincing results playing against a human opponent, featuring a diversity of strokes (and corresponding spin types) from both human and robot. To our knowledge this level of stroke adaption has not yet been shown by any other robot.

Going beyond the topic of this research work, these methods could have an impact on spin detection in other research areas, especially research focusing on sports with fast flying and strongly rotating balls, like tennis, baseball or football. As well as developing robots for these sports, spin detection can also be used for match analysis or for evaluating and improving player technique. There are various general robotic applications where it is necessary to determine the rotation of objects. In the case of table tennis, processing time is the key factor in determining whether or not the application will be successful. In modern highly-dynamic robotic systems, time-optimized object pose

*This work was supported in parts by the Vector Stiftung and KUKA

[1]Jonas Tebbe, Lukas Klamt, Yapeng Gao and Andreas Zell are with the Cognitive Systems group, Computer Science Department, University of Tuebingen, 72076 Tuebingen, Germany `[jonas.tebbe, yapeng.gao, andreas.zell]@uni-tuebingen.de, lukas-raphael.klamt@student.uni-tuebingen.de`

detection is essential, e.g. when grasping objects in human-robot collaboration or during autonomous driving in high-speed traffic.

II. Spin Estimation from the Brand Logo by Background Subtraction

A PointGrey Grasshopper3 camera is mounted on the ceiling above the center of the tablet tennis table. The camera can achieve 162 fps at full resolution (1920 x 1200). A very high ball spin exceeds 100 revolutions per second. In this case the ball's brand logo could be visible only every second frame. Therefore, we selected a ROI of 1920 x 400 and record at 380 fps, which is possible with this camera type. The exposure time was 0.25 ms.

A. Ball Detection

The ball is extracted from the image using a frame difference method taken from [10]. Figure 2 presents two example sequences of cropped ball images showing the movement of the brand logo.

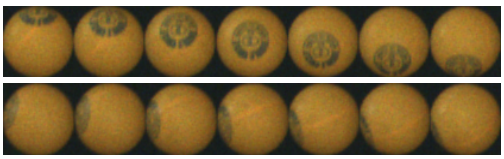
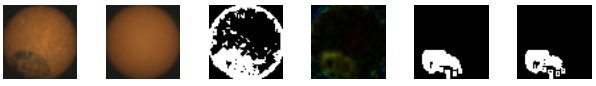


Fig. 2. Top row: A sequence of ball images in which the rotation of the brand logo is fully visible. Bottom row: The logo is also visible throughout the sequence, but the movement at the edge of the ball's image is more difficult to see.

B. Logo Contour Detection

Ball detection yields an image containing only the ball at a size of around 70 x 70 pixels. The process of marking all the pixels that belong to the brand logo is described in figure 3. For all pixels of the logo contour, we want to know the 3D positions on the ball.



(a) (b) (c) (d) (e) (f)

Fig. 3. logo detection process using motion and color features: (a) current frame, (b) ball without logo, (c) color threshold of a, (d) difference of a and b (e) binary threshold of d, (f) bitwise or of (c) and (e)

C. 3D Projection

The ball extraction also gives the radius of the ball in pixels. This is calculated by fitting a circle to the ball blob. For each contour pixel, we first calculate its position relative to the ball's center. The x and y components are then divided by the ball's radius. Since our 3D point must lie on the unit sphere, the z component can be derived from

$$1 = x^2 + y^2 + z^2.$$

D. Brand Logo Center

The next step involves calculating the logo center. In our first approach, we simply normalize the average of all 3D contour points. This does not take into account the fact that contour points closer to the ball's center in the image are more frequent. The centroid can also be calculated iteratively using Ritter's bounding sphere [11] on the 3D contour points. Normalization projects the centroid onto the unit sphere. As this did not significantly boost accuracy, we used the first approach, which was faster.

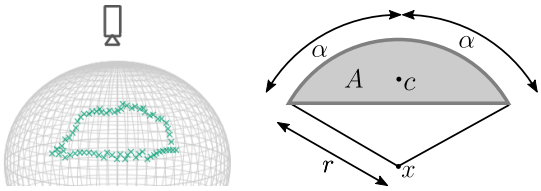


Fig. 4. Left: 3d projected logo contour for a partially visible ball seen from the top. Right: Schematic representation of a circular segment

On the camera images only one side of the ball is visible. Therefore, brand logos may be only partially in view when they are located at the edge of the shown area. The left of figure 4 shows a contour transformed into the 3D ball coordinate space for such an edge case. In this case the contour does not form a circle but a 2D circle segment, so the center position cannot be obtained as before.

We approximate the area A as π times the average distance from the contour (green crosses in figure 4 left) to the centroid. We know the actual radius r of the logo from measurements. We can therefore derive the distance from the centroid c to the real center x from the area A [12]:

$$A = \frac{r^2}{2}(2\alpha - \sin(2\alpha))$$

$$d(x,c) = \frac{4r\sin^3(\alpha)}{3(2\alpha - sin(2\alpha))}.$$

To get the real 3D center we rotate the centroid c by angle $\beta = 360°d(x,c)/2\pi r$ around the axis $(0,0,1)$. The circular segment fitting stabilizes the spin detection for the challenging edge case compared to the original approach of Zhang et al. [3].

E. Fitting Rotation

After processing 10 to 30 images captured every $1/380s$ from the ball's trajectory as described above, we can estimate the ball's spin. For this purpose, we fit a plane through the center points. The fitted plane should minimize the distance to the points. Additionally, the distance of the points to the circle created by intersecting the plane with the ball, represented as the unit sphere, should be minimized. The normal of the plane corresponds to the axis around which the ball rotates. An example is shown in figure 5.

To get the angular velocity, we project the logo positions onto the plane and calculate the angle ω between two consecutive logo positions. If the logo was not found on

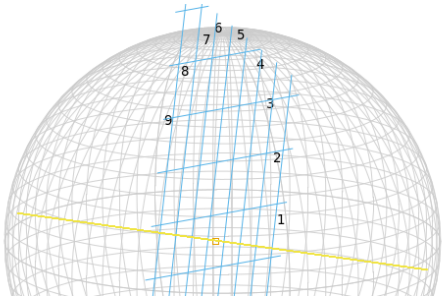


Fig. 5. Detected logo positions are labeled in chronological order. The fitted plane is visualized by a grid of lines. In green one can see the its normal, which represents the rotation axis of the ball.

two or more successive images, the ball has made a half revolution. The rotation is therefore described not by the short angle ω between the points before and after but by the large angle $360° - \omega$. At the end we have a sequence of the accumulated angles and fit a regression line to the sequence. The gradient of that line gives us the angular velocity.

III. Spin Estimation by CNN on Ball Image

Our second approach uses a Convolutional Neural Network (CNN) to estimate the visibility of the logo and the 3D pose of the ball. We then use the same algorithm as in section II-E to estimate the rotation axis and angular velocity.

A. Dataset

To train and test the network, a total of 4656 images were recorded using our PointGrey Grasshopper3 camera. The images were cropped around the table tennis ball to have a fixed size of 60 x 60 pixels. 46.7% were labeled as having no visible brand logo. The ball's pose was labeled with the help of a 3D scene containing a ball with realistic logo texture. The 3D scene was modeled with the open-source 3D computer graphics software Blender [13]. Each real ball image was placed transparently over the scene. Next, the 3D ball model can be optimized to fit the actual image and the pose can be read out by the Blender Python API.

B. Network Architecture

Related work on pose detection with neural networks [14], [15] favours the residual network architecture from He et al. [16]. To use the information for our robot, the model has to run approximately in real-time. Therefore we use the smallest of the ResNet architectures from [16] having 18 layers.

Expanding the idea of Mahendran et al. [14] we tested networks with two additional fully-connected layers (FC) with 512 neurons each right before the final regressor. This modification should improve the transformation from feature space to pose space. The effectiveness of the additional layers at various dropout rates can be observed in table I.

C. Network Output

There are several mathematical representations of a rotation. One can use rotation matrices, Euler angles, axis angles representation, or quaternions. Matrices do not fit as output of our network, as more parameters need to be estimated and one needs to ensure that the result is within the matrix subgroup $SO(3)$ of rotation matrices. With Euler angles it is difficult to represent continuous rotations. As a result, we trained networks to predict the pose of the table tennis ball in either axis angle representation or in quaternions. For either representation, the output is concatenated with a real number for the visibility of the brand logo. The range of the visibility value is $[-1, 1]$ to match the z-positions away from the camera. In the dataset non-visible logos are labeled as -1.

D. Loss Functions

The proposed neural network has to learn two tasks simultaneously. It needs to classify whether the brand logo of the ball is visible and predict the pose of the ball. If the logo is not visible, the pose cannot be determined. In this case, the network should not learn any incorrect poses. Hence, we define a conditional loss function that splits the loss into the two tasks:

$$\mathcal{L} = \mathcal{L}_{classification} + t_v \ \mathcal{L}_{orientation}$$

where t_v denotes the binary ground truth visibility value. For a visible logo, the value is 1. Otherwise it is 0. Therefore, we call it conditional loss.

When outputting in axis angle or quaternion representation, we adjust the pose losses for ambiguity. In both representations, the negative value gives the same orientation since the rotation in the opposite direction about the negative axis corresponds to the original rotation. We tested both L_1 and L_2 norms to get the following conditional losses:

$$\mathcal{L}_1 = (o_v - t_v)^2 + t_v \min\left(\sum_{i=0}^{n}(o_i - t_i)^2, \sum_{i=0}^{n}(o_i + t_i)^2\right)$$

$$\mathcal{L}_2 = |o_v - t_v| + t_v \min\left((\sum_{i=0}^{n}|o_i - t_i|, \sum_{i=0}^{n}|o_i + t_i|\right)$$

where $o = (o_1, \cdots, o_n, o_v)$ is the output vector of the network and $t = (t_1, \cdots, t_n, t_v)$ is the target vector.

A more complex, but fairly exact measurement of the accuracy of rotations is the geodesic distance in $SO(3)$. For two rotations this metric returns the angle (from axis-angle representation) of the rotation aligning them both. If R_1, R_2 are rotation matrices the geodesic distance is calculated as

$$d_{GD}(R_1, R_2) = arccos\left(\frac{tr\left(R_1^T R_2\right) - 1}{2}\right)$$

For quaternions q_1, q_2 the geodesic distance is computed by

$$d_{GD}(q_1, q_2) = 2arccos(| < q_1, q_2 > |)$$

where $|\cdot|$ denotes the absolute value and $< \cdot, \cdot >$ is the inner product of 4-dimensional quaternion vectors. As before, we define a new loss function

$$\mathcal{L}_{GD} = |o_v - t_v| + t_v \ d_G D\left(o, t\right).$$

The best performance was achieved by training quaternions with the conditional $\mathcal{L}_2$ loss (see table II).

GAP	FC	dropout	train. loss cond.-$\mathcal{L}_1$	test loss cond.-$\mathcal{L}_1$	classification accuracy	geodesic in deg.	vector angle in deg.
1	-	-	**0.0309**	0.3028	0.974	33.97	7.72
-	3	-	0.0342	0.3212	**0.975**	36.32	8.01
-	3	0.5	0.2674	0.4288	0.974	49.86	19.81
-	3	0.8	0.1347	0.3162	0.974	34.55	13.74
1	3	-	0.1006	0.2199	**0.975**	24.01	5.08
1	3	0.5	0.0976	0.2215	0.974	23.90	5.16
1	3	0.8	0.1022	**0.2161**	0.974	**23.06**	**4.89**

TABLE I

ALL MODELS ARE RESNET ARCHITECTURES. THE CLASSIFICATION COLUMN SHOWS THE ACCURACY OF THE BINARY CLASSIFICATION TASK. GEODESIC DESCRIBES THE GEODESIC DISTANCE BETWEEN GROUND TRUTH LABEL AND PREDICTION. VECTOR ANGLE DENOTES THE METRIC FROM CHAPTER III-D. THE BEST RESULTS ARE MARKED IN **BOLD**.

loss function	rotation representation	geodesic in deg.	vector angle in deg.
cond.-$\mathcal{L}_1$	quaternions	**23.06**	**4.89**
	axis angle	27.40	9.93
cond.-$\mathcal{L}_2$	quaternions	27.92	6.69
	axis angle	30.90	11.20
L2	quaternions	43.38	13.54
	axis angle	37.63	17.90
Geodesic	quaternions	23.49	5.97

TABLE II

NETWORK METRICS EVALUATED ON NETWORKS TRAINED WITH DIFFERENT LOSS FUNCTIONS AND ROTATION REPRESENTATIONS.

E. Metrics

The most difficult part of the rotation for the networks to determine is the logo's orientation about its center. We therefore also want to evaluate the accuracy of the network's prediction of the position of the logo on the ball only, i.e. without considering whether it is rotated in itself. For that we need an additional metric not affected by the orientation. We convert the rotation of the ball to logo positions, represented by points on the unit sphere, by rotating the base logo position $(0, 0, 1)$. The metric is then the vector angle describing the angle between two positions.

The network is used on several images of the ball trajectory. For the final spin estimation the poses outputted from the networks are converted to logo positions as described in the previous paragraph. We then use the same algorithm as in section II-E to estimate rotation axis and angular velocity.

F. Training Setup

The dataset from section III-A was split into training and test set with a $4 : 1$ ratio. As a result, 3725 images were used for training and 931 for testing. The networks were trained with Tensorflow using an Nvidia GeForce GTX 1080 Ti graphics card.

G. Inference Time

In our scenario it is not just accuracy that matters - time for the evaluation (inference time) is also important. From the camera we get images at 380 Hz. This gives us a processing time of 2.6 ms per image for real-time performance. Segmenting the ball out and cropping takes 0.5 ms, leaving 2.1 ms for the network. The best network has an inference time of 3.7 ms on a GTX1080 Ti graphics card. We solve the problem by processing in batches of 5 images, taking only 5.5 ms in total due to reduced overhead.

Our best performing network is an 18-layer ResNet plus global average pooling and two fully connected layers trained to output quaternions with conditional $\mathcal{L}_2$ loss. Augmenting the data with $90°$ rotations and Gaussian noise with standard deviation of 5% achieves the following result:

class. acc.	geodesic	vector angle
0.96	$20.14°$	$4.23°$

IV. SPIN ESTIMATION FROM THE TRAJECTORY

In this section we introduce a way of estimating the spin from the trajectory of the ball. We utilize the fact that the rotation of the ball acts on the ball via the Magnus force. Similar work on the topic has been done by Huang et al. [5].

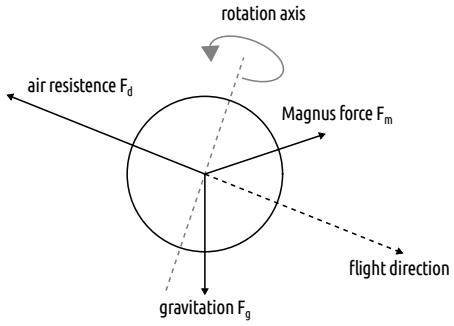


Fig. 6. The graphic visualizes the three forces acting on the ball: gravitation pointing downwards, air resistance or drag force in the opposite direction to the flight, and Magnus force perpendicular to the spin axis and flight direction.

The forces are depicted in figure 6. The gravitational force F_g is directed towards the ground. The drag force coming from the air resistance acts in the opposite direction to the flight of the ball. The Magnus force is perpendicular to the rotation axis and the flight direction. The acceleration of the

ball is therefore calculated by

$$\dot{v} = -k_D \|v\| v + k_M \omega \times v - \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix}. \quad (1)$$

The notation is shortened with $k_D = -\frac{1}{2} C_D \rho_a A/m$ and $k_M = \frac{1}{2} C_M \rho_a A r/m$, where the constants are the mass of the ball $m = 2.7\text{g}$, the gravitational constant $g = 9.81\text{m/s}^2$, the drag coefficient $C_D = 0.4$, the density of the air $\rho_a = 1.29\text{kg/m}^3$, the lift coefficient $C_M = 0.6$, the ball radius $r = 0.02\text{m}$, and the ball's cross-section $A = r^2\pi$. For a ball with medium to heavy spin the forces all have similar magnitudes, as can be seen in figure 8.

A. Fitting

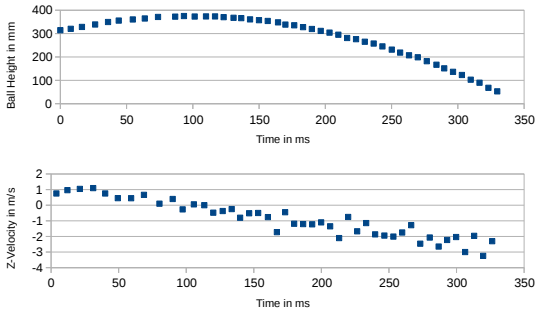


Fig. 7. The top diagram shows the height or z-positions for an example trajectory. For the same trajectory the z-velocity, calculated between each pair of neighbouring points, is shown below.

Given 3D observations $b_1, ..., b_n$ of the ball with ball positions $b_i = (x_i, y_i, z_i)$ at times $t_1, ..., t_n$ we need to estimate velocity and acceleration of the ball to derive the Magnus force. Calculating the velocity between two points is error prone as seen in figure 7. The problem is solved by fitting a third degree polynomial $P(t) = (P_x(t), P_y(t), P_z(t))$ for each axis using a standard least-squares algorithm. At time step t the velocity is approximated by $P'(t)$ and the total acceleration by $P''(t)$. Rewriting equation (1) with these approximations yields

$$k_M \omega \times P'(t) = P''(t) + k_D \|P'(t)\| P'(t) + \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix}.$$

Here we assume that the rotation vector ω is constant within the time of flight. We get this equation for each $t = t_1, ..., t_n$. All the equation can be transformed into the equation system $M\omega = a$ with a $m \times 3$ matrix M and an m-dimensional vector of accelerations a. We then get a least-squares solution for ω. Note that the acceleration caused by drag force is perpendicular to the Magnus acceleration on the left. As an effect our fitting of ω does not depend on the coefficient k_D but only on k_M in contrast to other work [5], [7]. In figure 8 the forces in each step are displayed for an example trajectory.

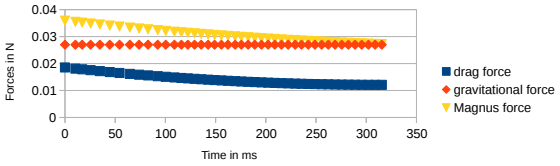


Fig. 8. In the diagram the three involved forces for the Magnus fitting are displayed for an example trajectory of a topspin ball.

B. Preprocessing: Outlier filtering

The process is error prone to outliers. Even for a slight impact for the fitted trajectory these outliers can produce unrealistic fitted spin values. Especially at the beginning of the trajectory incorrect recognitions can occur when a part of the human body, e.g. the hand, is detected instead of the ball due to its roughly circular shape. For the first 20 balls we select the last 5 balls to form a queue Q and make a polynomial fit as above. The polynomial is used to predict ball $\#14$. If the positional error is above a specific threshold, we mark ball $\#14$ as an outlier. Otherwise, we change Q by adding $\#14$ to the front and removing the last element (i.e. $\#20$). Then we continue to test ball $\#13$. Repeating this process we remove detected objects which do not belong to the trajectory at the beginning.

With the position $b = P(t_n)$, speed $v = P'(t_n)$ and spin ω we predict the future trajectory. The improvement for the prediction can be seen in table III. We tested backspin, side spin and topspin at three different speed settings with our TTmatic ball throwing machine. For comparison a Kalman filter is used to only predict position and speed without considering the angular velocity of the ball. The statistic includes 90 trajectories in total divided into 10 trajectories each. The estimated spin values significantly improves bounce estimation. In contrast to the first two approach using the ball's logo, the spin can used for predicting the future trajectory without adjusting the Magnus coefficient C_M. As we divide by it for spin estimation we multiply again for prediction. For the other methods, we found no constant C_M independent of the spin type, which gave good results.

		With fitted spin		Without spin value	
in mm		Error	Stddev	Error	Stddev
Backspin	Low	10.28	5.09	36.78	6.57
	Medium	27.02	11.22	125.76	17.08
	High	43.37	32.14	170.75	25.15
Sidespin	Low	9.68	5.56	43.15	7.99
	Medium	16.35	10.47	82.74	13.82
	High	27.99	9.80	108.24	11.23
Topspin	Low	19.01	5.62	90.10	16.96
	Medium	23.36	11.24	167.17	14.76
	High	86.84	52.70	338.28	31.00

TABLE III

RESULTS ON BOUNCE POINT PREDICTION FOR BALLS SERVED FROM A BALL THROWING MACHINE WITH DIFFERENT SETTINGS. FOR EACH SETTING WE RECORDED 10 TRAJECTORIES.

V. Comparison

In this paper, we looked at three algorithms to detect the spin of a table tennis ball. The first two can be compared by evaluating the angular error between the actual and the predicted logo position. The original background subtraction method gives an angular error of $5.77°$. We improved it to $5.06°$ by using circular segment fitting. Our best convolutional neural network reached an error of only $4.23°$. However, both background subtraction methods are faster, with an average processing time of 0.3 ms, compared to the network inference at 3.7 ms per image. Batch processing accelerates inference slightly (section III-G).

For the final spin prediction there are no ground truth values available. Therefore we evaluate how good the algorithms are for the classification of spin types. Using a TTmatic ball throwing machine we recorded 50 trajectories each for 3 spin types and 3 different powers, 9 settings in total. Unfortunately the machine does not allow the speed and rotation of balls to be set independently. Faster spin is therefore accompanied by a higher velocity. The median spin is calculated for each algorithm and setting. This 3D vector defines a cluster in three-dimensional space. Each spin value is then assigned to the nearest cluster center. The accuracy values in table IV show how many of the trajectories were correctly classified for each setting.

Surprisingly, the algorithms are similar in accuracy, slightly in favor of the trajectory fitting. A drop in performance is noticeable for balls with a lot of side spin. For this spin type the logo often rotates around itself at the top and hardly changes position. Then the first two variants reached their limit. For the same case appearing on the invisible side the logo cannot be seen and evaluated with these methods. The third algorithm does not suffer from brand logo dependence. However, it had difficulty distinguishing between the medium and high backspin. For these, the trajectories were not different enough leading to two median values relatively close to each other. All in all, a good classification can be achieved with all methods. An improvement would probably achieved by combining an approach using the brand logo with the Magnus force fitting.

Spin type	Background subtraction	Bg. sub. + segment fit	CNN	Trajectory fitting
Backspin				
Low	88.0%	94.0%	96.0%	100.0%
Medium	84.0%	92.0%	94.0%	58.0%
High	70.0%	86.0%	80.0%	60.0%
Sidespin				
Low	94.0%	98.0%	98.0%	100.0%
Medium	68.0%	58.0%	74.0%	94.0%
High	60.0%	68.0%	66.0%	100.0%
Topspin				
Low	84.0%	90.0%	88.0%	86.0%
Medium	78.0%	86.0%	88.0%	96.0%
High	90.0%	96.0%	96.0%	100.0%
in total	79.6%	85.3%	86.7%	88.2%

TABLE IV
Clustering accuracy of all the algorithms.

VI. Evaluation on a table tennis robot

The success of spin detection is demonstrated on a KUKA Agilus KR6 R900 robot arm. The table tennis robot system has to respond to different spin types generated by a human opponent. For this demonstration, we decided to go with the trajectory Magnus force fitting. It is more accurate, easier to set up and uses fewer resources. No additional camera hardware is necessary since the ball's positions are already captured to predict the trajectory.

We originally developed a table tennis robot system to play without spin [10]. In short, the ball position is extracted from a multi-camera system using color and movement information. Then the ball's 3D position is triangulated. To track the position and velocity of the ball we use an extended Kalman filter. The future trajectory is predicted from this using the force equation (1). As soon as we roughly know, where to hit the ball, we move the robot to this position with predefined bat orientation and velocity using a custom driver software and the KR-C4 controller.

In the new spin scenario, the return strokes use a different bat orientation. It is given in Euler angles in the order zyx. The z- and x-angle are still only dependent on the y-position of the hitting point. However, the y-angle β is linearly dependent on the y-component β_{spin} of the fitted rotational velocity of the ball. For a topspin or backspin ball with $\beta_{spin} = \pm 360°/s$, we use a β of $28°$ and $-40°$, respectively. For other values of β_{spin}, we interpolate linearly. At the time of hitting the table tennis racket has a velocity of approximately 1 m/s in the direction of the human opponent.

A video demonstration of the experiment is available online[1]. The rubber of the bat is a professional table tennis rubber with high friction. A lot of spin therefore acts on the ball after contact with the bat and our robot is still able to return the ball consistently. As far as we are aware, no other table tennis robot has yet achieved the feat of returning the ball under such challenging conditions.

VII. Conclusion and future work

Three methods for spin detection of a table tennis ball were introduced or further enhanced. These approaches were compared. Yielding the best accuracy, the trajectory fitting method is used to generate consistent returning strokes with our KUKA Agilus robot in cooperative, but highly challenging spin-play against a human opponent.

In future, we plan to go from cooperative to competitive strokes. Although our robot control approach is effective for cooperative spin play, it clearly has limits in terms of adaptability. Only the angle of the bat while hitting is adapted using a basic linear model. In a next step, the predicted spin may help to train the speed and orientation of the bat in a more sophisticated way using reinforcement learning.

[1] https://youtu.be/SjE1Ptu0bTo

REFERENCES

[1] "Ich fixiere den Stempel auf dem Ball. Timo Boll, 37, spricht im Interview über den siebten EM-Triumph seiner Tischtenniskarriere." *Spiegel Online*, Sep 2018. [Online]. Available: https://www.spiegel.de/sport/timo-boll-ich-fixiere-den-stempel-auf-dem-ball-a-00000000-0002-0001-0000-000159674382

[2] T. Tamaki, H. Wang, B. Raytchev, K. Kaneda, and Y. Ushiyama, "Estimating the spin of a table tennis ball using inverse compositional image alignment," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2012, pp. 1457–1460.

[3] Y. Zhang, Y. Zhao, R. Xiong, Y. Wang, J. Wang, and J. Chu, "Spin observation and trajectory prediction of a ping-pong ball," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 4108–4114.

[4] Y. Zhang, R. Xiong, Y. Zhao, and J. Wang, "Real-time spin estimation of ping-pong ball using its natural brand," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 8, pp. 2280–2290, Aug 2015.

[5] Y. Huang, D. Xu, M. Tan, and H. Su, "Trajectory prediction of spinning ball for ping-pong player robot," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2011, pp. 3434–3439.

[6] Y. Zhao, Y. Zhang, R. Xiong, and J. Wang, "Optimal state estimation of spinning ping-pong ball using continuous motion model," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 8, pp. 2208–2216, Aug 2015.

[7] Y. Zhao, R. Xiong, and Y. Zhang, "Model based motion state estimation and trajectory prediction of spinning ball for ping-pong robots using expectation-maximization algorithm," *Journal of Intelligent & Robotic Systems*, vol. 87, no. 3, pp. 407–423, Sep 2017. [Online]. Available: https://doi.org/10.1007/s10846-017-0515-8

[8] P. Blank, B. H. Groh, and B. M. Eskofier, "Ball speed and spin estimation in table tennis using a racket-mounted inertial sensor," in *Proceedings of the 2017 ACM International Symposium on Wearable Computers*, ser. ISWC '17. New York, NY, USA: ACM, 2017, pp. 2–9. [Online]. Available: http://doi.acm.org/10.1145/3123021.3123040

[9] Y. Gao, J. Tebbe, J. Krismer, and A. Zell, "Markerless racket pose detection and stroke classification based on stereo vision for table tennis robots," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, Feb 2019, pp. 189–196.

[10] J. Tebbe, Y. Gao, M. Sastre-Rienietz, and A. Zell, "A table tennis robot system using an industrial kuka robot arm," in *Pattern Recognition*, T. Brox, A. Bruhn, and M. Fritz, Eds. Cham: Springer International Publishing, 2019, pp. 33–45.

[11] J. Ritter, "Graphics gems," A. S. Glassner, Ed. San Diego, CA, USA: Academic Press Professional, Inc., 1990, ch. An Efficient Bounding Sphere, pp. 301–303. [Online]. Available: http://dl.acm.org/citation.cfm?id=90767.90836

[12] Wikipedia contributors, "List of centroids — Wikipedia, the free encyclopedia," 2019, [Online; accessed 27-February-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=List_of_centroids&oldid=883001666

[13] Blender Online Community, *Blender - a 3D modelling and rendering package*, Blender Foundation, Blender Institute, Amsterdam, 2016. [Online]. Available: http://www.blender.org

[14] S. Mahendran, H. Ali, and R. Vidal, "3D Pose Regression using Convolutional Neural Networks," Tech. Rep., 2017. [Online]. Available: https://shapenet.cs.stanford.edu/media/syn

[15] S. S. M. Salehi, S. Khan, D. Erdogmus, and A. Gholipour, "Real-time Deep Pose Estimation with Geodesic Loss for Image-to-Template Rigid Registration," Tech. Rep. [Online]. Available: https://github.com/SadeghMSalehi/DeepRegistration

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Tech. Rep. [Online]. Available: http://image-net.org/challenges/LSVRC/2015/

Appendix C

Sample-efficient Reinforcement Learning in Robotic Table Tennis

Authors

Jonas Tebbe, Lukas Krauch, Yapeng Gao, and Andreas Zell

Published in

2021 IEEE International Conference on Robotics and Automation (ICRA)

Time of publication

October 2021

DOI

10.1109/ICRA48506.2021.9560764

Sample-efficient Reinforcement Learning in Robotic Table Tennis*

Jonas Tebbe[1], Lukas Krauch[1], Yapeng Gao[1], and Andreas Zell[1]

***Abstract*— Reinforcement learning (RL) has achieved some impressive recent successes in various computer games and simulations. Most of these successes are based on having large numbers of episodes from which the agent can learn. In typical robotic applications, however, the number of feasible attempts is very limited. In this paper we present a sample-efficient RL algorithm applied to the example of a table tennis robot. In table tennis every stroke is different, with varying placement, speed and spin. An accurate return therefore has to be found depending on a high-dimensional continuous state space. To make learning in few trials possible the method is embedded into our robot system. In this way we can use a one-step environment. The state space depends on the ball at hitting time (position, velocity, spin) and the action is the racket state (orientation, velocity) at hitting. An actor-critic based deterministic policy gradient algorithm was developed for accelerated learning. Our approach performs competitively both in a simulation and on the real robot in a number of challenging scenarios. Accurate results are always obtained within under 200 episodes of training. The video presenting our experiments is available at `https://youtu.be/uRAtdoL6Wpw`.**

I. Introduction

Reinforcement learning (RL) is, next to supervised and unsupervised learning, one of the three basic machine learning areas. RL is a technique in which an artificial agent or a robot learns an optimal decision-making policy in a specific environment by trial and error. In recent times, reinforcement learning has come to great success in various video and board games such as Atari-Games [1], [2] and Go [3]. After OpenAI introduced new robotic environments [4], [5], strong results were also achieved for simulations of various robotic scenarios [6], [7], [8].

This suggests that these learning algorithms might also be used to control real robots. It could shorten the development process for new control algorithms and thus bring robotics into other previously unavailable application areas. However, it is not possible to adapt these successful methods directly [9]. Millions of attempts are often required to solve a task such as playing an Atari game. On a real robot this is not feasible in a reasonable amount of time. In addition, exhaustive exploration strategies are often not suitable without damaging the robot and its environment.

In this paper we present a reinforcement learning algorithm for a table tennis playing robot, in which we address various problems of realistic reinforcement learning applications in robotics:

*This work was supported in parts by the Vector Stiftung and KUKA.

[1]Jonas Tebbe, Lukas Krauch, Yapeng Gao and Andreas Zell are with the Cognitive Systems group, Computer Science Department, University of Tuebingen, 72076 Tuebingen, Germany `[jonas.tebbe, yapeng.gao, andreas.zell]@uni-tuebingen.de`, `lukas.krauch@student.uni-tuebingen.de`

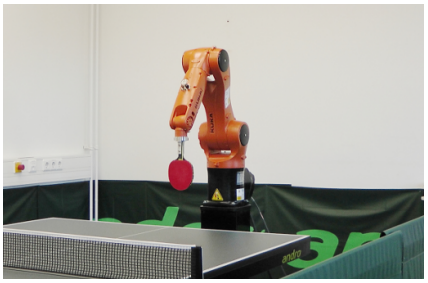


Fig. 1. Table tennis robot system with a KUKA Agilus robot. The goal is to learn the orientation and the velocity of the racket at hitting time.

- Sample efficiency.
  In every scenario learning is possible with only a small dataset of fewer than 200 ball returns.
- Robustness.
  The robot is facing multiple sources of noise in ball measurements, trajectory predictions and arm movements.
- Robot safety.
  Exploration is only used sparsely to avoid dangerous and unreachable configurations.

II. Related Work

A. Reinforcement Learning in Robotics

RL is particularly successful in applications for which information, such as the dynamics, would otherwise be necessary to solve the task [10], [11], [12]. In those cases thousands of episodes could be generated, which is often not possible in the field of robotics. Different approaches are needed to overcome this drawback. Often most of the learning phase is done in simulation and afterwards applied to the real world [13], [14], [15]. Using multiple robots in parallel can increase efficiency, for example in a door opening task [16] or to collect grasping attempts [17]. To accelerate learning a difficult task one often includes human knowledge into the RL algorithm. This can mean shaping the reward function [18] directly or including human feedback within the reward signal [19], [20], [21]. Often expert demonstrations are used for initialization or within training [22], [23], [24]. Building upon and improving conventional controllers can make learning in real world scenarios possible [25], [26]. The RL-algorithm of this paper is embedded into a robot software environment. This way prior knowledge of the system is utilized to simplify the learning problem. By using data from a prediction algorithm and passing the resulting robot target state to a trajectory planner, we could reduce

complexity and learn in very few examples, i.e. playing only 200 balls with our table tennis robot.

B. Learning in Robotic Table Tennis

Robotic table tennis is a challenging field for learning algorithms needing accurate control in a fast-changing noisy environment. Most of the research is done in simulation. [27] showed that their Relative Entropy Policy Search method works in a simulated table tennis environment using only a sparse reward. Using a one-step DDPG approach similar to ours [28] could learn very precise policies by simulation up to 200,000 balls. In [29] a 8-DOF robot was controlled in joint space with an evolutionary search CNN-based policy training. [30] developed a trajectory-based monotonic policy optimization and applied it to learning to hit a simulated spinning table tennis ball. [31] used a virtual reality environment to collect human example strokes and self-train a policy on top of these.

Applying these techniques on a real robot is another challenge and approaches are much fewer. In [32] a drone is equipped with a cardboard racket and learns to hit a table tennis ball use Dynamic Motion Principles. One key element was also playing the ball in a flat manner. [33] have their robot learn a table tennis stroke as a combination of movement primitives. The motion of the opponent's racket is used in [34] to predict the aim of the opponent and adjust movement timing and generation accordingly. [35] even developed a new pneumatic robot arm capable of moving with high accelerations and taught it to smash table tennis balls using PPO.

All these approaches brought promising results, but could only play table tennis in a very limited scenario, such as against a ball throwing machine or using really slow balls.

III. The Learning Problem

Our goal is to teach a KUKA Agilus industrial robot (see. Fig. 1) how to play table tennis. Two high-speed cameras are mounted on the ceiling of the robot laboratory to determine the position of the ball. The robot arm is to perform the table tennis stroke in such a way that the ball then hits a target point on the other side of the table with the highest possible precision. An end-to-end learning model using the raw images from the cameras can only be realized with an extremely large number of examples and would need a lot of processing power. We have therefore already developed a tracking system that predicts the trajectory of the ball up to the moment of impact on the bat [36]. As only the point of hitting between ball and racket is essential, we parameterize the stroke movement by position, speed and orientation of the racket at the point of impact with the ball. The position is estimated by our trajectory prediction algorithm. Speed and orientation are outputs of the reinforcement learning problem. Finally, we use them to iteratively plan the arm trajectory using the Reflexxes Library [37].

A. Interpretation as a Reinforcement Learning Problem

Following the usual practice in reinforcement learning, we define our problem as a Markov decision process

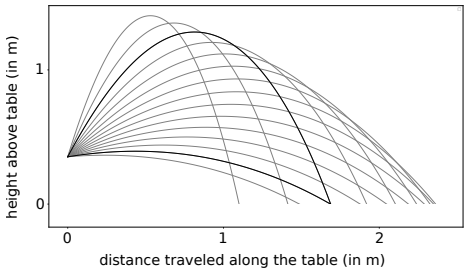


Fig. 2. The figure shows several simulated ball tracks with different starting angles, viewed from the side. The ambiguity is evident by the two black trajectories with the same achieved goal position.

(S, A, p, γ, r). To reduce complexity, episodes have length 1, i.e. the transition function $p : S \times A \times S$ maps all states with probability 1 to the end state e. The state space $S \subset \mathbb{R}^9 \cup \{e\}$ is a 9-dimensional interval plus end state e. Its elements are the vectors concatenating the 3D position, velocity and spin of the table tennis ball just before the stroke. The action space $A \subset \mathbb{R}^2$ contains elements (α, β) consisting of the Euler angles α, β of the bat at hitting time. For episodes of length 1 the discounting factor γ does not come into play.

B. Reward

The reward should depend on the distance between achieved goal position and target goal. However, this makes the optimal solution ambiguous. By only changing one angle of the racket orientation we can get two ball trajectories with the same achieved goal reached as illustrated in Figure 2. One of the trajectories belongs to a very high ball, which is undesirable as it gives the opponent a good opportunity to smash the ball. Thus, we also penalize the height of the ball and define the reward by

$$r = -|g_t - g_a| - \alpha \cdot h$$

where g_t is the targeted goal position on the table, g_a is the achieved goal position, h is the height value of the ball halfway to the goal and α is the coefficient that weights the influence of the height value.

IV. The Environments

A. Simulation

To verify the functionality of the learning algorithm and for hyper-parameter optimization a simulation was designed (see. Fig. 3). The ball trajectory is calculated by forward solving the following differential equation using a forth order Runge-Kutta method. The underlying equation model [36] is

$$\dot{v} = -k_D \|v\| v + k_M \omega \times v - \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix}. \qquad (1)$$

Here k_D is the coefficient for the drag force (air resistance), k_M the coefficient for the Magnus force coming from the angular velocity (spin) of the ball, g is the gravitational constant, and $v(\omega)$ are the translational (angular) velocity of

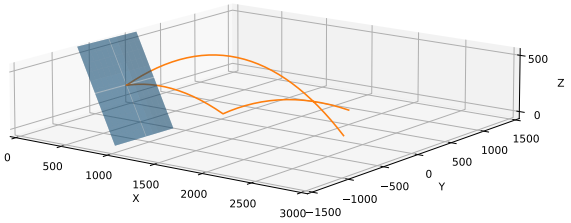


Fig. 3. Simulated example trajectory. The racket is represented by the blue plane.

the ball. With this we can estimate the trajectory in midair. For the bounce at the table we apply an elastic collision model, where the weight of the ball is negligible in relation to the weight of the table, i.e. $m_b \ll m_t$. In this case we obtain the new z component of the velocity vector by $v_{b_z}' = -v_{b_z}$. Analogously we proceed for the collision between ball and racket. Again, the racket connected to the robot arm is much heavier, $m_b \ll m_r$. First we transform the velocity vectors v_b and v_r so that the z axis is in the direction along the normal of the racket plane. We refer to this transformation as T. Then by one-dimensional elastic collision we have

$$(Tv_b)_z' = 2 * (Tv_r)_z - (Tv_b)_z.$$

While the flight model is rather realistic, the bounce models are now oversimplified. Still, the simulation provides a solid, repeatable test-bed for performance evaluation of the algorithms.

B. Robot

On the real robot we use the Robot Operating System (ROS). The trained actor network is evaluated in a Python ROS node. The process is illustrated in figure 4. We use a stereo system with two PointGrey Chameleon3 cameras to record the table tennis ball. The ball tracking node finds the ball on each camera using traditional image processing and triangulates the pixel positions to output the position of the ball in 3D [36]. In the high-level node the sequence of positions is stored. After an outlier removal the sequence is used to predict the state of the ball at the time it hits the racket. The velocity and the position are estimated using an extended Kalman filter [36]. The spin is derived from the trajectory by using our Magnus force fitting approach [38]. Each new prediction is forwarded to the stroke parameter node where the actor is evaluated. It outputs the desired state of the racket at hitting time which is then sent to the trajectory generation node. Using the Reflexxes library [37] the trajectory of the robot arm is calculated and finally sent to the KUKA Agilus KR6 R900 robot using the Robot Sensor Interface (RSI).

To give the robot more time for the movement execution we begin with actions computed from early hitting state predictions and gradually refined as more accurate measurements become available. For the purpose of training only the last, most accurate, values are used.

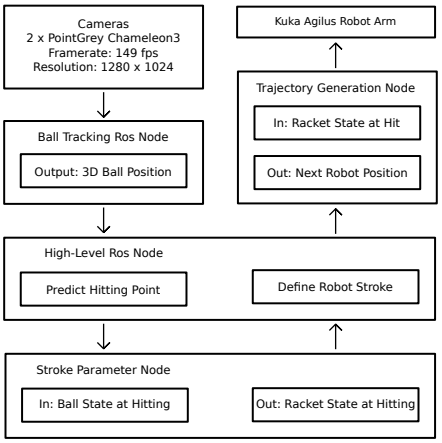


Fig. 4. Process on the real robot.

V. The Algorithm

Our algorithm uses an actor-critic model similar to DDPG [6] / HER [7]. The training of the critic is adapted in such a way that a parameter vector independent of the target goal is learned instead of the reward. Together with the target goal the corresponding reward can be calculated. Using the gradient of this function, the output of the actor is trained to maximize the reward. The approach is depicted in figure 5 and it will be denoted by APRG (accelerated parametrized-reward gradients).

The deterministic actor-critic model consists of two components. A supervised trained critic network and an actor model outputting the learned policy trained with the help of the critic's gradient (see Fig. 5).

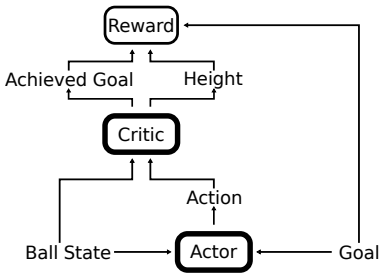


Fig. 5. Modified actor-critic model using a parameterized reward.

A. Critic

In our scenario the critic receives the ball state (predicted position, velocity and spin at hitting time) and the action (orientation and velocity of the bat) as input and outputs the achieved goal position and average ball height above the table estimated for the specified state and action. The L_2-loss is used for training. Learning reward parameters and not the reward itself has several advantages. The critic does not need the desired goal as input and the parameters are less complex

Algorithm 1 Training algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s, g|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize replay buffer B
Initialize a random process $\mathcal{N}$ for action exploration (with large variance in warm-up phase)
for episode = 1, E **do**
  Receive observation state s_e and desired goal g_e
  Select action $a_e = \mu(s_e, g_e|\theta^\mu) + \mathcal{N}_e$ according to the current policy and exploration noise
  Post-optimize action a_e using the gradient of the reward function R:

$$\nabla_a R(Q(s, a|\theta^Q), g)|_{s=s_e, a=a_e, g=g_e}$$

  Execute action a_e and observe reward parameters r_e
  Store episode (s_e, a_e, r_e) in B
  if after warm-up phase **then**
    for training_step = 1, S **do**
      Sample a random minibatch of N episodes (s_i, a_i, r_i) from the replay buffer B
      Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (r_i - Q(s_i, a_i|\theta^Q))^2$
      Update actor policy using the sampled policy gradient:

$$\frac{1}{N}\sum_i \nabla_{\theta^\mu} R(Q(s, \mu(s, g|\theta^\mu)|\theta^Q), g)|_{s=s_i, g=g_i}$$

    end for
  end if
end for

than the complete reward function. This helps to reduce complexity. Also, the outputs are easier to be understood by a human which helps in debugging.

B. Actor

The actor is fed with the ball state and the target goal position and should return the action. To train the actor we assume the critic weights fixed and derive the reward with respect to the actor weights. Using this gradient in the optimization step, the actor will use actions which maximize the reward calculated from the critic's output. The training procedure is written down in algorithm 1.

C. Exploration

Exploration on the real robot is not suitable for the whole action space. Part of the search space might include robot configurations which are not reachable at all or in the available time. We decided to start recording actions with small Gaussian noise added to a safe action. With enough samples the gradient of the critic is roughly pointing in the correct direction for improvements, and we can start training. In most cases the actions are then changing in a way using only feasible configurations.

A one-step DDPG approach was already proposed for robotic table tennis [28]. Unfortunately they only showed performance for between $10,000$ and $200,000$ training examples generated in simulation. Our approach is tested also on a real robot and most experiments are conducted on as few as 200 episodes. It is also differing from classical DDPG by the following modifications:

- The critic is to output the parameters needed to calculate the reward instead of the reward.
- We post-optimize the actions via SGD.
- We start with a warm-up phase of random actions instead of ϵ-random action in-between learning.

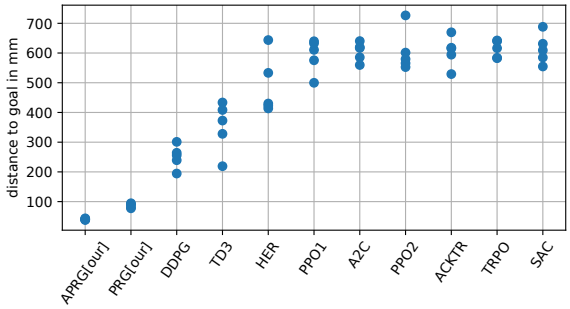
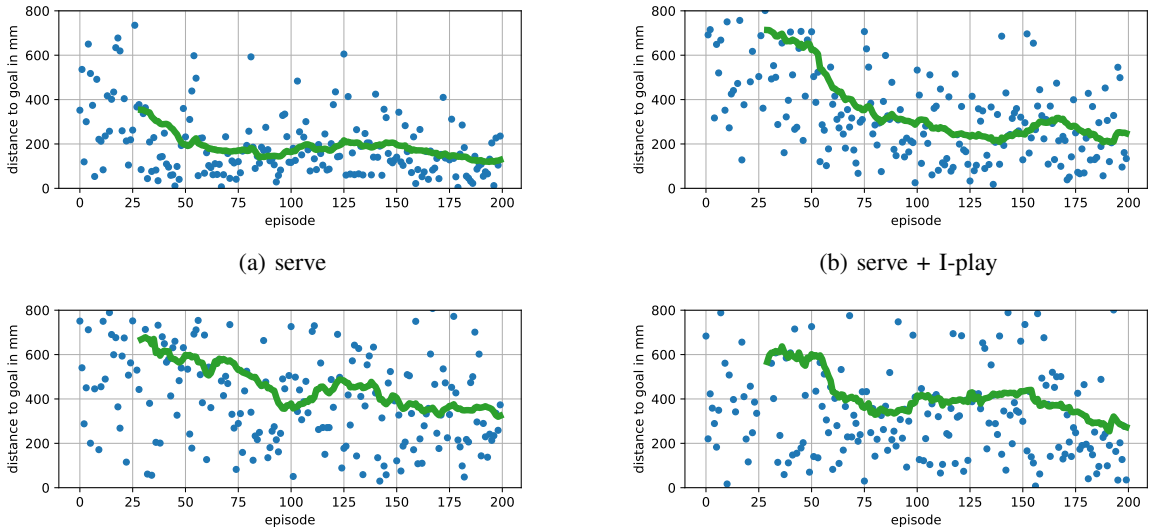


Fig. 6. Comparison against the baseline algorithms. Showing the five results for the best parameters from 100 tested trials using the Optuna hyperparameter optimization framework.

VI. Experiments in Simulation

To get a set of effective hyperparameters we have conducted a parameter search on our simulation with the Optuna framework [39]. For comparison, we also train policies using the state-of-the-art algorithms from the stable baseline repository [40] ([41], [42], [6], [7], [43], [8], [44], [45]). For a fair evaluation we did hyperparameter optimization for each of the baseline models. As the learning process seems to have a high variance, we decided to average over five tries for each set of parameters. With the Optuna framework we tested 100 parameter configurations for each method. Figure 6 shows results for the best parameters, respectively. It is



(a) serve

(b) serve + I-play

(c) serve + V-play

(d) serve + X-play

Fig. 7. Result for training on the real robot in four scenarios. The experiments always started with a warm-up phase of 30 random actions. The green line represents the running average over the 30 episodes.

evident, that DDPG-based methods (APRG, PRG, DDPG, TD3, and HER) are performing better.

Our APRG algorithm stays at the top with an average goal error of $40.6mm$. It also shows that post-optimization (APRG) gives better results than unoptimized parametrized reward gradients (PRG), but one can get faster inference time with PRG at the expose of a little accuracy.

For be fair, we must mention that the difference between the algorithms becomes smaller when learning over 2000 episodes or more. However, performance on a smaller number is more relevant, because in table tennis, rapid adaption plays a major role. In cooperative play with the real robot, human players quickly became impatient when they could not see any improvement in the robot returns.

VII. Experiments on the Robot System

To show that our method also works on the real robot we conducted several experiments of increasing complexity.

A. System noise

In a first experiment we want to find out how much noise the learning process has to cope with on the real robot. For this purpose we let the ball machine TTMatic 404 serve the ball 200 times in the same way and let the robot return the ball with unchanging, predefined action parameters. In fact, the balls have the same hitting position with an accuracy of $46.6mm$ and an average speed deviation of $0.92m/s$. The deviation of the achieved target positions for the resulting robot returns is much larger with an average accuracy of $123.9mm$. We assume that to be the limit achievable in the best case. This shows how challenging the scenario is.

B. Human play in regular exercises

In our main experiment we deducted experiments against a human player on four increasingly challenging exercise scenarios. The player is playing the ball in a predefined sequence. In this way we can judge the performance for increasing difficulty. But these types of exercises are a regular part of table tennis training for amateurs as well as professionals. So a robot capable of learning these could augment human training procedures. The algorithm is starting from scratch using a warm-up phase of 30 random actions and a total of 200 episodes/ball contacts.

The following scenarios are tested:

- Simple backhand serves.
  The human is always playing the same serve and the robot has to return to the middle of the table (Goal: $[2000, 0]$).
- Serve and human I-play.
  The human begins with a serve and the rally is continued along the mid-line of the table (Goal: $[2400, 0]$).
- Serve and human V-play.
  The human begins with a serve and has to alternate the ball placement between the left and right side of the table, on success forming a V-shape (Goal: $[2400, 0]$).
- Serve and human X-play.
  The human begins with a serve and in the following ball exchange the robot and the human place the ball alternately on both sides of the table, forming a X shape if successful (Goals: $[2200, -300]$ and $[2200, 300]$).

The goal coordinates are specified in the coordinate system of the table, where $x = 0/1370/2740$ is the robot's table end / net line / human's table end and $y = -762/0/762$ is the left end / middle line / right end of the table. The odd numbers

are coming from the standard table size of $9ft \times 5ft \times 2.5ft$.

The results are presented in figure 7. To put the result into perspective, consider the limit of $124mm$ of the fixed action evaluated by the first experiments. So for the serve-only scenario an average of $136mm$ (x: $114mm$, y: $47mm$) to the goal for the last 50 balls is coming very close to that. In the I-play scenario the rally is continued after the serve making it more challenging. A result of $269.2mm$ accuracy in the last 50 episodes is worse, but the x-error is $243.3mm$ and the y-error is $78mm$ showing that the ball is still playing accurately to the middle of the table. The V-play has more deviation from the human player, as each ball is played differently. In this exercise we could achieve a goal error of $329mm$ (x: $177mm$, y: $126mm$). Even more challenging is the X-play achieving a goal error of $393mm$ (x: $282mm$, y: $238mm$).

C. Human play with different opponents

The experiments from the last section were all conducted with a player very familiar with the robot and its behavior. To test the robot also against different play styles, we invited three players of the local table tennis club. They were just instructed to play cooperatively with the robot.

Results are presented in figure 8. Performance losses are visible when players have tried new strokes or placements. But the error always converged to an acceptable value.

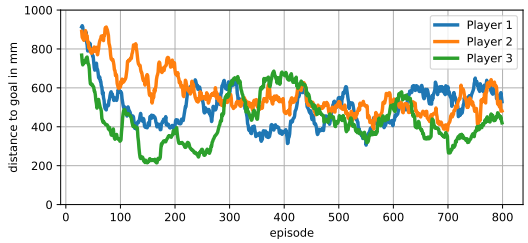


Fig. 8. Result for training on the real robot against three human players. The experiments always started with a warm-up phase of 30 random actions. The lines represent the running average over the last 50 episodes.

D. Using a ball throwing machine

While our main focus is playing against a human opponent, we also did a learning experiment with a ball throwing machine. This scenario is particularly suitable for comparisons, as this is the most common test for table tennis robots in the literature. At first the machine should place the ball only on one spot, analogous to the system noise experiment. The robot should learn the action for returning the ball to the middle of the table (Goal:$[2400, 0]$). This results in a very accurate return with a goal error of $118mm$ (x: $85mm$, y: $63mm$) over the last 50 of 200 episodes in total. Secondly we change the ball throwing machine to distribute balls evenly on the side of the robot. Here we have an accuracy to the target of $209mm$ (x: $172mm$, y: $88mm$).

A comparison of our results to other table tennis robots in the literature can be found in table I. Since most papers only record the return rate of balls successfully played to the opponent half of the table we also included these. It is clearly visible that our approach is achieving state-of-the-art performance. Only [46] has a better return rate for an oscillating ball machine distributing on an area of $0.7mx0.4m$. But the area of our ball throwing machine is larger with $1mx0.3m$ covering more extreme angles on both right and left side of the table, making our scenario slightly more challenging.

robot model	stroke type	sample size	hitting rate	return rate	error to goal
KUKA Agilus KR6 [our]	serve	50	100%	100%	135mm
[our]	I-play	50	98%	96%	269mm
[our]	V-play	50	100%	88%	329mm
[our]	X-play	50	98%	92%	393mm
[our]	occ. BM	50	98%	88%	209mm
[our]	fixed BM	50	98%	98%	118mm
Barrett WAM [47]	occ. BM	200	-	80%	-
Barrett WAM [46]	occ. BM	30	-	97%	460mm
[46]	I-play	-	-	88%	-
Muscular Robot [35]	fixed BM	107	96%	75%	769mm
Wu/Kong [48]	occ. BM	732	-	71%	-
lab-made [49]	fixed pos.	-	-	80%	-

TABLE I
COMPARISON AGAINST OTHER TABLE TENNIS ROBOTS.

VIII. CONCLUSION AND FUTURE WORK

In this research work a RL algorithm was developed for sample efficient learning in robotics. Extensive experiments were conducted to test it in a real robotic environment. It should determine the parameters for the optimal return of a table tennis ball. The results are measured by the accuracy with respect to a defined target on the table. The learning process is integrated into an existing robot system using a KUKA Agilus KR 6 R900 robot arm. The robot could learn an accurate return in under 200 balls. This demonstrates robust and effective learning in a very noisy environment. Comparing the success rate of the returns, our algorithm beats the previous research approaches. Beyond the application for robotic table tennis, our method can be used in all cases where the trajectory of a robot can be represented by a lower-dimensional parameter vector, as in our case orientation and speed at the hitting point.

On the way to competitive play against top human players there is still a lot to do. In the future we plan to let our robot learn in even more challenging match realistic scenarios. This requires generalization for many more domains like serve/no serve, topspin/backspin/sidespin, short/long balls etc. The goal parameters should also include speed and spin, which will be needed for a successful strategy capable of beating advanced human players.

REFERENCES

[1] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," 2017.

[2] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, "Agent57: Outperforming the atari human benchmark," 2020.

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. [Online]. Available: https://science.sciencemag.org/content/362/6419/1140

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[5] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba, "Multi-goal reinforcement learning: Challenging robotics environments and request for research," 2018.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1509.02971

[7] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5048–5058. [Online]. Available: http://papers.nips.cc/paper/7090-hindsight-experience-replay.pdf

[8] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 1861–1870. [Online]. Available: http://proceedings.mlr.press/v80/haarnoja18b.html

[9] A. Irpan, "Deep reinforcement learning doesn't work yet," https://www.alexirpan.com/2018/02/14/rl-hard.html, 2018.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 02 2015.

[11] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[12] J. Kober, J. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, pp. 1238–1274, 09 2013.

[13] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3803–3810.

[14] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8973–8979.

[15] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 262–270. [Online]. Available: http://proceedings.mlr.press/v78/rusu17a.html

[16] S. Gu*, E. Holly*, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proceedings 2017 IEEE International Conference on Robotics and Automation (ICRA)*. Piscataway, NJ, USA: IEEE, May 2017, *equal contribution.

[17] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018. [Online]. Available: https://doi.org/10.1177/0278364917710318

[18] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning*, ser. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, p. 278–287.

[19] R. Loftin, J. MacGlashan, B. Peng, M. E. Taylor, M. L. Littman, J. Huang, and D. L. Roberts, "A strategy-aware technique for learning behaviors from discrete human feedback," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, ser. AAAI'14. AAAI Press, 2014, p. 937–943.

[20] W. Saunders, G. Sastry, A. Stuhlmüller, and O. Evans, "Trial without error: Towards safe reinforcement learning via human intervention," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '18. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2018, p. 2067–2069.

[21] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4299–4307.

[22] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[23] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," 2018.

[24] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 6292–6299.

[25] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6023–6029.

[26] T. Silver, K. R. Allen, J. Tenenbaum, and L. Kaelbling, "Residual policy learning," *ArXiv*, vol. abs/1812.06298, 2018.

[27] J. Peters, K. Mülling, and Y. Altün, "Relative entropy policy search," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, ser. AAAI'10. AAAI Press, 2010, p. 1607–1612.

[28] Y. Zhu, Y. Zhao, L. Jin, J. Wu, and R. Xiong, "Towards high level skill learning: Learn to return table tennis ball using monte-carlo based policy gradient method," in *2018 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, 2018, pp. 34–41.

[29] W. Gao, L. Graesser, K. Choromanski, X. Song, N. Lazic, P. Sanketi, V. Sindhwani, and N. Jaitly, "Robotic table tennis with model-free reinforcement learning," *arXiv preprint arXiv:2003.14398*, 2020.

[30] R. Akrour, A. Abdolmaleki, H. Abdulsamad, J. Peters, and G. Neumann, "Model-free trajectory-based policy optimization with monotonic improvement," *Journal of machine learning research*, vol. 19, no. 14, p. 1–25, 2018. [Online]. Available: https://jmlr.csail.mit.edu/papers/volume19/17-329/17-329.pdf

[31] R. Mahjourian, N. Jaitly, N. Lazic, S. Levine, and R. Miikkulainen, "Hierarchical policy design for sample-efficient learning of robot table tennis through self-play," *CoRR*, vol. abs/1811.12927, 2018. [Online]. Available: http://arxiv.org/abs/1811.12927

[32] R. Silva, F. S. Melo, and M. Veloso, "Towards table tennis with a quadrotor autonomous learning robot and onboard vision," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 649–655. [Online]. Available: http://www.cs.cmu.edu/~mmv/papers/15iros-PingPong.pdf

[33] J. Peters, J. Kober, K. Mülling, O. Krämer, and G. Neumann, "Towards robot skill learning: From simple skills to table tennis," in *Machine Learning and Knowledge Discovery in Databases*, H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 627–631.

[Online]. Available: http://is.tuebingen.mpg.de/fileadmin/user_upload/files/publications/2013/peters_ECML_2013.pdf
[34] Z. Wang, C. H. Lampert, K. Mülling, B. Schölkopf, and J. Peters, "Learning anticipation policies for robot table tennis," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011, pp. 332–337. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.233.939&rep=rep1&type=pdf
[35] D. Büchler, S. Guist, R. Calandra, V. Berenz, B. Schölkopf, and J. Peters, "Learning to play table tennis from scratch using muscular robots," 2020.
[36] J. Tebbe, Y. Gao, M. Sastre-Rienietz, and A. Zell, "A table tennis robot system using an industrial kuka robot arm," in *Pattern Recognition*, T. Brox, A. Bruhn, and M. Fritz, Eds. Cham: Springer International Publishing, 2019, pp. 33–45.
[37] T. Kröger, "Opening the door to new sensor-based robot applications—the reflexxes motion libraries," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1–4.
[38] J. Tebbe, L. Klamt, and A. Zell, "Spin detection in robotic table tennis," *CoRR*, vol. abs/1905.07967, 2019. [Online]. Available: http://arxiv.org/abs/1905.07967
[39] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.
[40] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.
[41] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1928–1937. [Online]. Available: http://proceedings.mlr.press/v48/mniha16.html
[42] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5279–5288. [Online]. Available: http://papers.nips.cc/paper/7112-scalable-trust-region-method-for-deep-reinforcement-learning-using-kronecker-factored-approximation.pdf
[43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms." *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1707.html#SchulmanWDRK17
[44] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 1587–1596. [Online]. Available: http://proceedings.mlr.press/v80/fujimoto18a.html
[45] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1889–1897. [Online]. Available: http://proceedings.mlr.press/v37/schulman15.html
[46] K. Muelling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013. [Online]. Available: https://doi.org/10.1177/0278364912472380
[47] O. Koç, G. Maeda, and J. Peters, "Online optimal trajectory generation for robot table tennis," *Robotics and Autonomous Systems*, vol. 105, pp. 121 – 137, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889017306164
[48] Y. Zhao, R. Xiong, and Y. Zhang, "Rebound modeling of spinning ping-pong ball based on multiple visual measurements," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 8, pp. 1836–1846, 2016.
[49] Y.-h. Zhang, W. Wei, D. Yu, and C.-w. Zhong, "A tracking and predicting scheme for ping pong robot," *Journal of Zhejiang University SCIENCE C*, vol. 12, no. 2, pp. 110–115, Feb 2011. [Online]. Available: https://doi.org/10.1631/jzus.C0910528

Appendix D

Real-time 6D Racket Pose Estimation and Classification for Table Tennis Robots

Title

Real-time 6D Racket Pose Estimation and Classification for Table Tennis Robots

Authors

Yapeng Gao, Jonas Tebbe and Andreas Zell

Published in

International Journal of Robotic Computing

Time of publication

2021

DOI

10.35708/rc1868-126249

Note: After the International Conference on Robotic Computing 2019 (ICR) conference we were invited to submit this journal paper. For brevity the conference version (Gao *et al.*, 2019a) of the paper is left out in the disseration.

International Journal of Robotic Computing

Real-time 6D Racket Pose Estimation and Classification for Table Tennis Robots

Yapeng Gao*, Jonas Tebbe†, Julian Krismer‡ and Andreas Zell§

Cognitive Systems Group at the Computer Science Department
University of Tuebingen, Germany
**yapeng.gao@uni-tuebingen.de*
†*jonas.tebbe@uni-tuebingen.de*
‡*julian.krismer@student.uni-tuebingen.de*
§*andreas.zell@uni-tuebingen.de*

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

For table tennis robots, it is a significant challenge to understand the opponent's movements and return the ball accordingly with high performance. One has to cope with various ball speeds and spins resulting from different stroke types. In this paper, we propose a real-time 6D racket pose detection method and classify racket movements into five stroke categories with a neural network. By using two monocular cameras, we can extract the racket's contours and choose some special points as feature points in image coordinates. With the 3D geometrical information of a racket, a wide baseline stereo matching method is proposed to find the corresponding feature points and compute the 3D position and orientation of the racket by triangulation and plane fitting. Then, a Kalman filter is adopted to track the racket pose, and a multilayer perceptron (MLP) neural network is used to classify the pose movements. We conduct two experiments to evaluate the accuracy of racket pose detection and classification, in which the average error in position and orientation is around 7.8 mm and 7.2 by comparing with the ground truth from a KUKA robot. The classification accuracy is 98%, the same as the human pose estimation method with Convolutional Pose Machines (CPMs).

Keywords: racket pose detection; pose classification; stereo matching; table tennis robot.

1. Introduction

Racket sports such as tennis, table tennis and badminton are popular worldwide. From a robotic point of view these sports pose several challenges, which should be addressed in real-time, for example, human motion analysis [1], racket 3D pose detection [2], flying ball position estimation [3] and robot trajectory planning [4]. With motion tracking technology for players or rackets, the robots can achieve an anticipatory action predicted from the human's movements, so that there is more execution time left for hitting movements. Tracking human motions or racket motions also allows robots to imitate the human motion to learn how to play human-like table tennis. When a ball flying towards the robot is recognized, a precise hitting

Fig. 1. Table tennis robot system with KUKA Agilus robot. There are four PointGrey Chameleon3 cameras mounted on the ceiling corners far away from each other to occupy more scenario, where two cameras opposite to human are used to detect the racket and another pair is for table tennis ball detection. A table tennis racket is rigidly fixed at the end effector of robot in a type of penhold grip. The robot coordinate is set as the world coordinate and the center of racket is defined as the tool coordinate center.

position will be estimated by combing ball position and spin together using a curve fitting algorithm [5] or an extend Kalman filter [6]. Finally, the robot will strike the ball back with an optimal human-like action determined by large amounts of training data.

With various racket movements generating different spin categories, racket sports are full of fun and challenges. To detect the 3D racket pose (position and orientation), much research has been done with sensors and markers. Ohya et al. [7] positioned four stationary cameras in order to cover a large field of view. By assuming the tennis racket to be modeled as ellipse shape, they estimated the 3D racket position with the fundamental matrix, which had ten to forty percent more success rate than only using one camera. Elliott et al. [8] employed a markerless approach with a master camera fixed and a slave camera dynamically located at 21 different positions to detect a set of tennis racket silhouette views. With single view

fitting techniques, the 3D racket position was estimated with a spatial accuracy of 1.9 ± 0.14 mm. Chen et al. [9] established a high-speed monocular vision system to track a table tennis racket labeled with some special marker lines in the form of a black rectangle in the middle and a white line parallel to one of the black lines. They can be extracted into several corners as feature points and the pose is computed based on perspective-n-point and orthogonal iteration algorithms. Blank et al. [10] attached inertial sensors into table tennis rackets to detect and classify 8 different stroke types from 10 amateur players. The success rates for detection and classification did reach 95.7% and 96.7%, respectively. Zhang et al. [2] fused inertial measurement unit (IMU) data with the method [9] based on an extended Kalman Filter for obtaining an accurate and robust racket pose. The racket position was computed from cameras and its orientation was estimated from both cameras and IMU resulting in an average angle error of 1.1.

In this paper, we present a novel approach for table tennis racket pose detection without markers or IMU based on stereo vision in a table tennis robot system [11]. The system is shown in Figure 1. As the black side of a table tennis racket is nearly invisible against our very dark field enclosure, the current system is restricted to detect the red side only. It can be extracted as a binary contour using a color thresholding method similar to the table tennis ball detection in [11]. To accelerate the detection process, we use bucket fill to find a connected component starting from the estimated point with the specified color threshold that determines the amount of connectivity. By ellipse fitting this contour, we can extract isolated point features located at the intersection area of ellipse and contour. Combing the epipolar constraint with the 3D geometrical size of the racket, we can match the corresponding feature points from two cameras. Triangulation results in 3D points, which are used for fitting the orientation of the plane going through the racket center. A Kalman filter is used to track the 3D pose and smooth the trajectories. Next, we classify these trajectories into five categories using a neural network in order to determine with which kind of spin the ball is played back. In the experimental results, we evaluate the poses against ground truth from a KUKA robot and compare our classification with a different method using human pose estimation.

The subsequent part of this paper has the following structure: Section 2 introduces related work. The proposed method is presented in Section 3. The evaluation and comparison are examined in Section 4. This paper is concluded in Section 5.

2. Related Work

2.1. *Feature Extraction*

Feature extraction involves a detector in the form of points, lines, blobs, or shapes, and a descriptor to generate a unique vector representing these features. ORB (oriented FAST and rotated BRIEF) is one such descriptor [12], which is currently popular. It incorporates the FAST key point detector with modified BRIEF descriptor to provide a fast and efficient alternative for SIFT, SURF, KAZE and

BRISK. However, there is an inherent disadvantage in the point-based method in low-textured scenarios in that it will fail due to the lack of reliable feature points. Consequently, line based methods are a possible solution since there are many surfaces like desks, doors and walls in low-textured scenarios, which are rich in line features. In [12], a proposed method with line segments for indoor visual localization is employed to handle low-texture images with a wide baseline, which is far better than other point based methods. In our case, the racket lacks both texture and lines, so that the above methods are not suitable to extract features from the racket face.

2.2. *Stereo Matching*

Stereo matching defines the correspondence problem, in which we find the corresponding points in two camera images. It is is divided into feature based stereo and area based stereo [13] . Following the feature extraction, feature based stereo utilizes the $L1$ *norm* or $L2$ *norm* for string based descriptors (SIFT, SURF, KAZE etc.) or Hamming distance for binary descriptors (ORB, BRISK etc.) to differentiate features in corresponding pairs [14]. Area based algorithms depend on the epipolar constraint for rectified images to search the corresponding points in the same image rows including local (NCC, SAD) and global methods (dynamic programming, graph cuts). A well known approach for real-time stereo vision is Semi-Global Matching (SGM) [15], which approximates a global 2D matching cost aggregation by minimizing the energy function from 8 or 16 different directions through the image. It can obtain the same accuracy as global matching but with lower runtime. Recently, end-to-end deep stereo gas become very popular to solve the stereo matching problem with CNNs models, consisting of embedding, matching, regularization and refinement modules [16]. However, they currently cannot yet fulfill real-time requirements.

2.3. *Pose Classification*

Player motion analysis is beneficial because the motion of the player determines the motion of the racket, and consequently the speed and spin of the ball. Chu et al. [17] extracted histogram of oriented gradient (HOG) features from badminton videos and employed a support vector machine (SVM) to classify a player's stroke into six types (clear, drive, drop, lob, smash), which resulted in 83.33% average accuracy. Srivastava et al. [18] developed a sports analytics engine based on an IMU to detect the tennis shot with a modified Pan-Tompkins algorithm, and proposed a time-warping based hierarchical shot classifier by using Dynamic Time Warping (DTW) at the first level (forehand, backhand and serve) and Quaternion Dynamic Time Warping (QDTW) at the second level (slice and non-slice). The accuracy at DTW and QDTW were 99.6% and 90.7% for professional players, 99.3% and 86.2% for novice players. With CNNs, Bearman et al. [19] addressed the human joint location as a regression problem and used weight initialization from a trained AlexNet to

classify human activity into 20 categories with the accuracy of 80.51%. In our work, a neural network including two hidden layers is utilized to train the racket pose trajectories and classify them into five types to achieve an accuracy of 98.7% on strokes of the same player.

3. Approach

3.1. *Racket Detection*

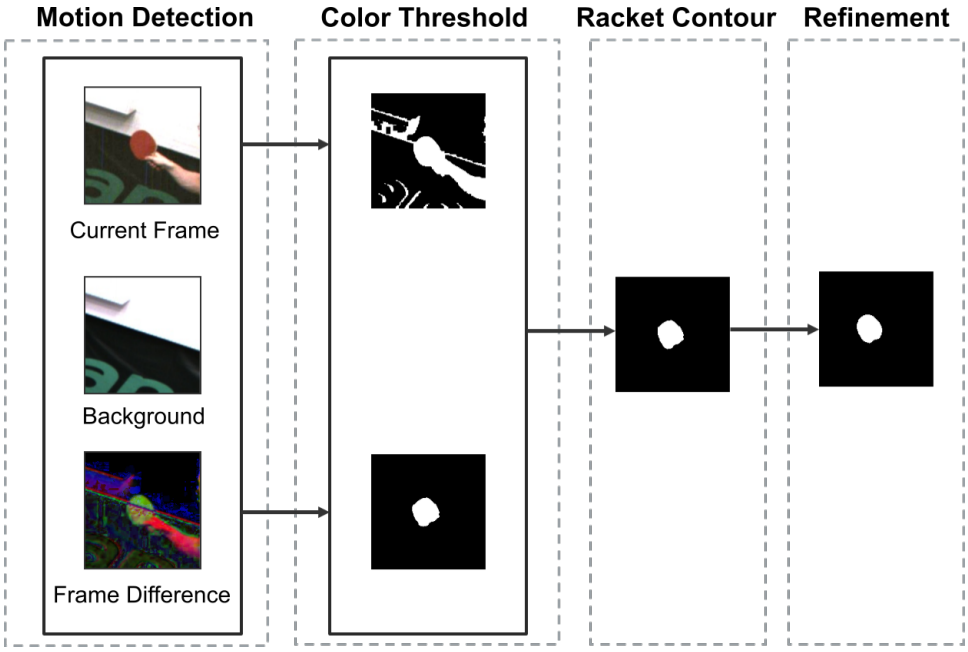


Fig. 2. Racket detection process with dynamic window from the right camera. *Motion detection*: subtract the background from current frame. *Color Threshold*: compute the binary image from RGB space. *Racket Contour*: bitwise AND operation from previous step. *Refinement*: bucket fill results.

To lower the impact on lighting variations, we choose the HSV color space instead of RGB and adopt the color thresholding algorithm similar to [11] with different boundary values to detect the red side of the racket. Multiple features of the racket are fused to extract the whole racket contour, like area and aspect ratio. Fig. 2 illustrates the pipeline of racket detection in the right camera, which includes four steps.

We primarily find the moving objects using a static frame difference method by subtract the background from current frame. The lighting between current frame and background is slightly different because we use the auto-exposure mode that

dynamically adjusts parameters including gain, shutter time and white balance. Performing thresholding and morphology operations , we can get the binary images in the *Color Threshold* step resulting in the racket contour processed by bitwise AND operation.

Considering the property of the racket contour, we can determine it based on the following conditions:

$$200px \leq Area \leq 3000px \tag{1}$$

$$1/3 \leq AspectRatio \leq 3 \tag{2}$$

$$0.3 \leq AreaExtent \leq 1 \tag{3}$$

where *Area* is the contour area in pixels. *AspectRatio* is the contour aspect ratio of the minimal containing up-right bounding box. *AreaExtent* is the ratio of *Area* to the bounding box. The contour with the largest area satisfying the condictions is chosen. Its center is used to triangulate the racket's center 3D position.

Once the racket is recognized in both current and previous frames, we first predict the position of the racket in the next frame by adding the current position with the position difference of the last two frames. Then, we exploit a region of interest (ROI) around the predicted position to crop the full image into a dynamic window in order to accelerate the detection process. Secondly, a multithreading technique supplied by C++ is used to execute image processing concurrently for the left and right camera images. The third acceleration method called bucket fill is applied to find a connected component spreading from the seed point until the color value is out of specified range computed as follows:

$$C(x,y)_H - L_H \leq C'(x,y)_H \leq C(x,y)_H + U_H \tag{4}$$

$$C(x,y)_S - L_S \leq C'(x,y)_S \leq C(x,y)_S + U_S \tag{5}$$

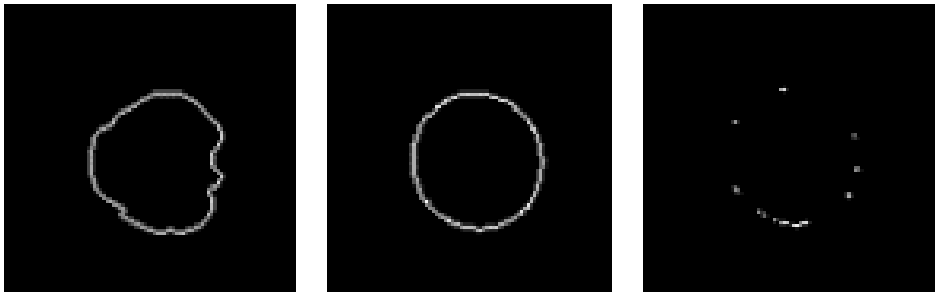
$$C(x,y)_V - L_V \leq C'(x,y)_V \leq C(x,y)_V + U_V \tag{6}$$

where H, S, V are the components from HSV model. $C(x,y)_H$ is the H component value at the seed point (x,y). $C'(x,y)_H$ is the repainted H component domain presenting the new racket contour. L or U is maximal lower or upper color difference between the seed point and one of its neighbours. Fig. 2 shows that bucket fill yields better results than color thresholding with a runtime decreasing from 6.5 ms to 2 ms.

3.2. *Racket Matching*

When the 2D pixel coordinates of the racket's center are available from the two cameras, we can reconstruct these points as the racket 3D position by triangulation. The 3D orientation can be defined as the unit normal vector of the racket plane. Matching the left and right contours directly is difficult because of their random and uncertain shapes.Therefore, we want to find some corresponding feature points on the edge of the racket to recover the 3D plane in point-normal form.

3.2.1. *Features Extraction*



(a) Contour (b) Ellipse (c) Features

Fig. 3. Features (c) extraction from the intersection area between contour (a) and ellipse (b).

Since the racket plane is low-textured and the two cameras are far away from each other, features detection and description is difficult. The strong edge around the racket contour is used to extract the feature points in the undistorted images, which will not be rectified due to wide baseline and large rotation angle.

We approximate the edge by three ellipse fitting methods supported by [20] including normal least squares (LS), Approximate Mean Square (AMS) and Direct least square (Direct) aimed at finding the best one which has the largest degree of overlapping D between the edge and ellipse formulated as following:

$$D = \frac{N_{overlapping}}{N_{edge}} \tag{7}$$

where $N_{overlapping}$ and N_{edge} are the pixel numbers of the intersection area and the edge. We calculate D for a sequence of images and show the comparison in Table 1. The direct method is adopted for ellipse fitting because of its performance. Then, we choose the points on the intersection area as feature points, shown in Fig. 3.

Table 1. Comparison of ellipse fitting models.

Methods	LS	AMS	Direct
Degree	53.77%	56.60%	58.33

3.2.2. *Stereo Matching*

Next, we do not intend to match the two sets of feature points to each other, but find the corresponding points in another contour's edge. Depending on the epipolar geometry, we can narrow down the choice of candidates of corresponding points

on the epipolar line. The points lying on both edge and epiline are the potential corresponding points of the feature points. Fig. 4 gives an example where P_R and P'_R are the intersection of edge and epiline related to the left point P_L. By means of the racket size, we can find the correct corresponding point from these two candidates described as:

$$P_1 = Triangulate(P_L, P_R) \tag{8}$$

$$P_2 = Triangulate(P_L, P'_R) \tag{9}$$

$$75 \leq |Pos - Center| \leq 90 \quad Pos \in [P_1, P_2]. \tag{10}$$

Here 75 mm and 90 mm are the length of the minor and major semi-axes. $Center$ is the 3D position of the racket. Therefore the inequality should be satisfied for a correct edge point. The algorithm chooses the point having the shortest distance by ($||Pos - Center| - \frac{75+90}{2}|$).

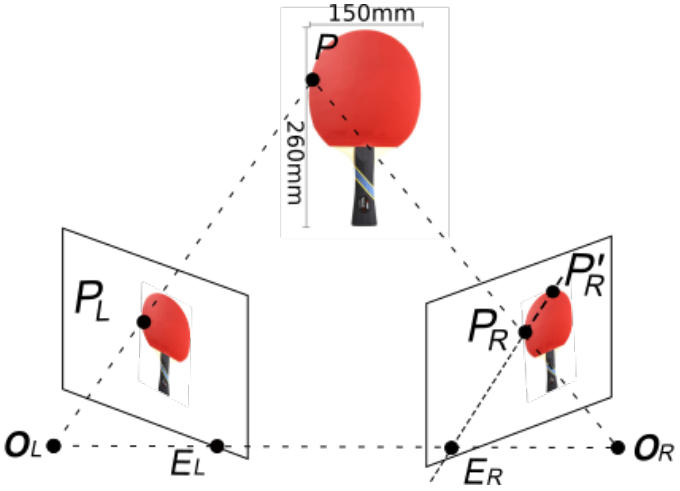


Fig. 4. Finding the potential candidates P_R and P'_R in the right camera corresponding to P_L. O_L and O_R are the optical centers of the cameras lenses. The epipolar line in the right camera passes through the epipole E_R, the image points P_R and P'_R.

3.2.3. *Outliers Removal*

The feature matching method aforementioned can produce many corresponding pairs consisting of inliers and outliers. Because of these pairs on same surface, a homography matrix H for removing outliers can be derived as a 3x3 matrix but with 8 DoF estimated by:

$$s \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{11}$$

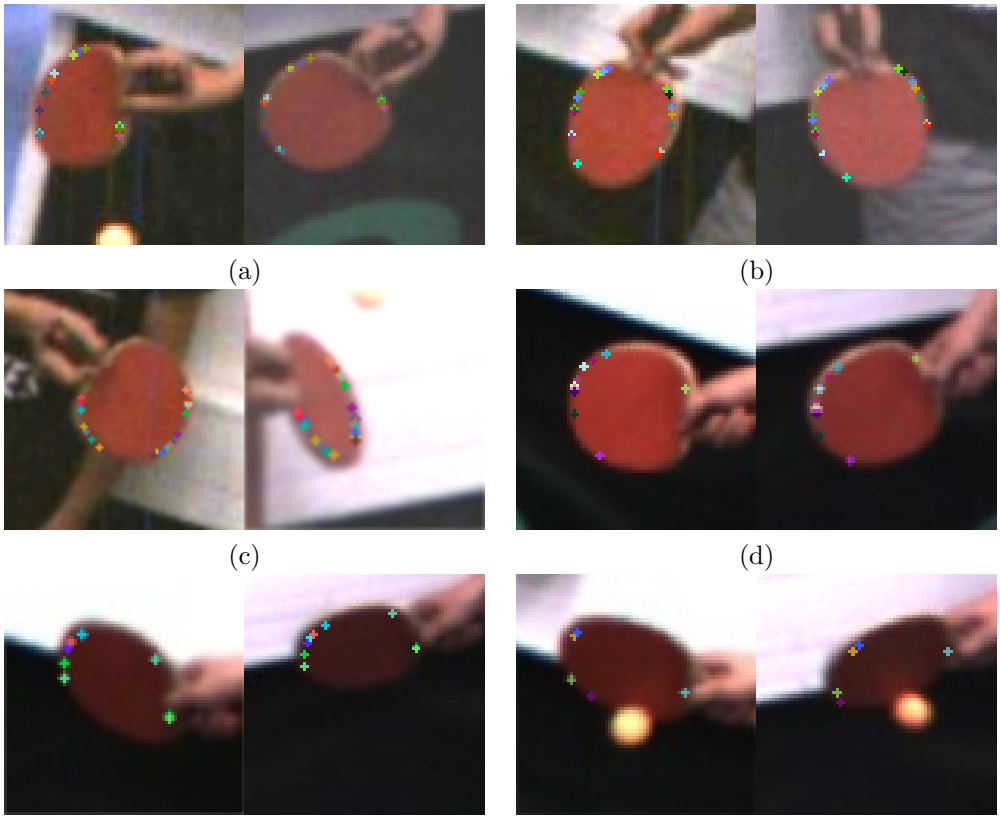


Fig. 5. Feature matching results including six examples in the left and right cameras. The grip type in (a)-(c) is penhold grip. (d)-(f) use the shakehand grip. The corresponding pairs are labeled with same color in order to distinguish the correct pairs.

where s is a scale factor. $[x_i, y_i]$ and $[x'_i, y'_i]$ are the i^{th} pixel coordinates from left and right cameras. According to this transformation, we can minimize the re-projection error function after projecting points from one image into another given by:

$$\sum_i \left(x'_i - \widehat{x_i}\right)^2 + \left(y'_i - \widehat{y_i}\right)^2 \tag{12}$$

where $\widehat{x_i} = \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + 1}$, and $\widehat{y_i} = \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + 1}$. They are the reprojected image coodinates.

However, using the whole pairs for matrix estimation will lead to a poor result. We utilize the Random SAmple Consensus (RANSAC) [21] to estimate the homography matrix by randomly selecting different subsets of the corresponding pairs and select the subset with the minimal re-projection error. Here, the outliers will be removed if the reprojection error is more than 3 pixels.

The final matching results are shown in Fig. 5 including 3 penhole and 3 shake-hand types. Each corresponding features pair in the left and right cameras is labeled with the same color to be distinguished clearly.

3.2.4. *Plane Fitting*

Reconstructing the corresponding pairs by triangulation, we can get a series of 3D points $[x_i, y_i, z_i]^T$ that can be used to estimate the equation of the racket plane $ax + by + c = z$. The centroid of these points is defined by the 3D racket center. The normal vector $[a, b, c]^T$ is described as:

$$\begin{bmatrix} x_0 & y_0 & 1 \\ y_1 & y_1 & 1 \\ & \cdots & \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} z_0 \\ z_1 \\ \cdots \\ z_n \end{bmatrix} \tag{13}$$

This can be written in the form $AX = B$. A common method to solve X is Singular Value Decomposition (SVD) by which A is decomposed as:

$$A_{n\times 3} = U_n S_{n\times 3} V_{3\times 3}^T \tag{14}$$

where U and V are orthogonal matrices, S is a diagonal matrix, and n is the number of corresponding pairs. Then, the last column of V indicates the value of normal vector $[a, b, c]^T$. Normalizing this vector, we can get the unit norm vector representing the racket's orientation. We measure the processing time for racket detection and matching shown in Fig. 6. The stereo matching needs around 8 ms. The total time for racket pose estimation needs about 10 ms, which means we can estimation the racket 6D pose at 100 FPS.

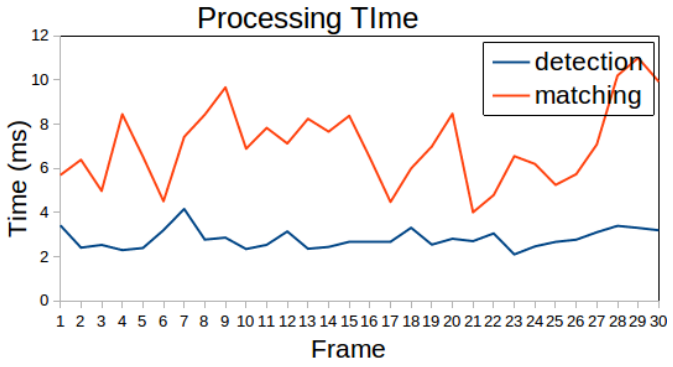


Fig. 6. Processing time for racket detection and matching.

3.3. *Tracking*

Tracking the racket pose offers two advantages. We can use the estimated pose when there is an occlusion or the racket is disappearing. Also it can provide a much smoother estimation of the racket pose. In this paper, we employ a discrete Kalman filter that is very efficient and powerful for estimating the pose $[x, y, z, a, b, c]^T$. We define the racket state X_t with 15 variables:

$$X_t = [x_t, y_t, z_t, \dot{x}_t, \dot{y}_t, \dot{z}_t, \ddot{x}_t, \ddot{y}_t, \ddot{z}_t, a_t, b_t, c_t, \dot{a}_t, \dot{b}_t, \dot{c}_t] \tag{15}$$

A simple motion model is used to compute the next expected state X_{t+1}:

$$\boldsymbol{p}_{t+1} = \boldsymbol{p}_t + \dot{\boldsymbol{p}}_t * \Delta t + \frac{1}{2}\ddot{\boldsymbol{p}}_t * \Delta t^2 \tag{16}$$

$$\dot{\boldsymbol{p}}_{t+1} = \dot{\boldsymbol{p}}_t + \ddot{\boldsymbol{p}}_t * \Delta t \tag{17}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \dot{\boldsymbol{\theta}}_t * \Delta t \tag{18}$$

where $\boldsymbol{p} \in [x, y, z]$ and $\boldsymbol{\theta} \in [a, b, c]$. Then, we can project the next state and error covariance ahead from current time and update them with current measurement. From Fig. 7, we note that in 30 frames, the estimated pose appears considerably smoother than the original one without Kalman filter.

3.4. *Classification*

Realizing the exact pose trajectory is not possible for humans, but people can still play table tennis really well due to their ability to recognize different stroke types. For robots, it is important to know not only what the exact pose is, but also which strike type is generated. There are many different types of stroke, but we can divide them into five basic categories: 1) *Counter Hit*. It is used to stop an aggressive, attacking stroke from your opponent by moving the racket and keeping it at the same angle. 2) *Left Spin*. It will be imparted when the racket moves to the left, which make the ball to bounce off in the same direction. 3) *Right Spin*. It is the opposite of left spin. 4) *Top Spin*. It is produced by starting the racket below the ball and hitting the ball in an upward and forward direction, which causes the ball to jump forwards after bouncing off the table and the opponent need to return with the racket face closed. 5) *Back Spin*. It is the opposite of top spin, with a downward stroke of the racket. If the opponent does not reply with a back spin or a strong top spin himself, the ball will drop down and into the net.

We stored the previous 30 frames to extract the trajectories of the racket pose once the ball flying towards robot is detected. To distinguish which spin type these trajectories belong to, we created a classifier based on a neural network containing two operations and two hidden layers able to predict in testing set:

3.4.1. *Flatten operation*

The input shape is 30×6, which means each frame from the previous 30 frames includes six values (x, y, z, a, b, c). This layer converts the 30×6 matrix into a 1D

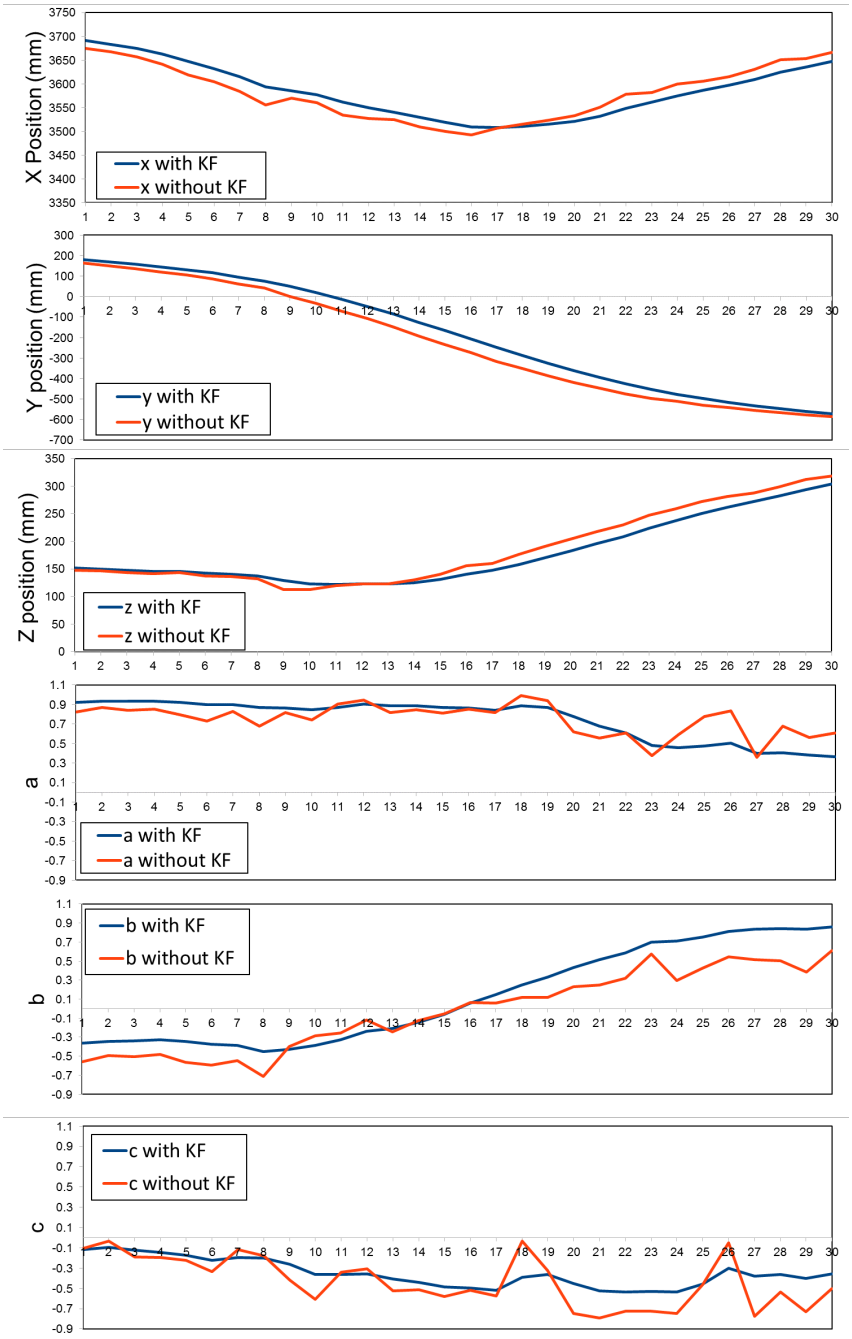


Fig. 7. Kalman filter tracking in 30 frames for racket position (x, y, z) and orientation (a, b, c).

feature vector 1×180 used in the artificial neural network (ANN) classifier. To simplify the dataset and make training more robust, we use the relative position to the last position in the 30^{th} frame instead of the absolute value, and normalize them into unit vectors.

3.4.2. *Dense layer*

It is also called fully connected layer and first performs a linear operation in which every neuron from the previous layer is fully connected to this layer by a weight matrix *kernel* as following equation:

$$output = ReLU(input \cdot kernel + bias) \tag{19}$$

where the shape of *output* adopted in this paper is 128-dimensional. As activation function *ReLU* (Rectified Linear Unit) is used to introduce non-linearity. *bias* is a bias vector created by this layer.

3.4.3. *Dropout operation*

By randomly setting a *rate* of input units to zero during the training phase of this set of units, we can reduce the over-fitting of training data. Here, *rate* is assigned to 20%.

3.4.4. *Dense layer*

This layer performs classification on input units into five categories. We choose the softmax function to activate the Dropout layer.

For each spin type, we recorded 200 videos to generate the racket trajectories and human pose, respectively. Among them, 80% of the dataset are used to learn the classification model, and the remaining 20% are used as test dataset. In training, we use the Adam optimizer and a loss function with sparse categorical cross-entropy. Then we can train the model for a specified number of epochs. To compare the classification difference of pose, position and orientation, we experiment with them, respectively. From Table 2, we can find the best performing is the 6D pose. The accuracy with 3D orientation is much better than the 3D position

Table 2. Classification accuracy comparison.

	6D Pose	3D Position	3D Orientation
Training Set	98.7%	51.58%	94.8%
Testing Set	98.2%	50.63%	93.6%

4. Experiments

In this section, we conduct two experiments to evaluate the performance of our proposed methods. All processes are executed on one host PC with an Intel i5-4590 CPU, 16GB RAM and a GeForce GTX 1050 Ti GPU.

We first use a pair of cameras facing the robot to detect the pose of the racket mounted at the robot end effector shown in Fig. 1, and compare it with the ground truth data read from the robot controller. Then, we adopt an existing 2D human pose estimation model, Convolutional Pose Machine [22], to extract human joints as feature points. We compare this deep neural network with the classified network presented before. The comparison results are shown in the following subsections.

4.1. *Evaluation on the KUKA robot*

We have already transferred the 3D coordinates from camera to robot by employing a least-squares fitting method with two 3D point sets in our table tennis robot system [11]. The tool coordinate system in the robot was moved from the end effector to the racket center. In our work, the unit norm vector u_T of the red side on the racket in tool coordinates is always $[-1, 0, 0]^T$ by the negative direction of the x axis shown in Fig. 8. Next, we transform this vector to robot coordinates (namely, world coordinates).

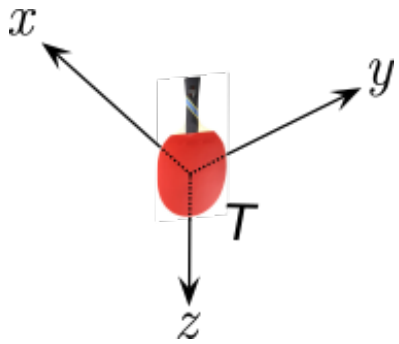


Fig. 8. The tool coordinate system.

The values $[X, Y, Z, A, B, C]$ can be read from the KUKA controller, where X, Y, Z are the racket's 3D position and A, B, C are the Z-Y-X Euler angles. The 3×3 rotation matrices about X, Y, Z axes are written as: R_X, R_Y, R_Z.

$$R_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos C & -\sin C \\ 0 & \sin C & \cos C \end{bmatrix} \tag{20}$$

$$R_Y = \begin{bmatrix} \cos B & 0 & \sin B \\ 0 & 1 & 0 \\ -\sin B & 0 & \cos B \end{bmatrix} \tag{21}$$

$$R_Z = \begin{bmatrix} \cos A & -\sin A & 0 \\ \sin A & \cos A & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{22}$$

Then, the norm vector u_W in world coordinate is derived by:

$$u_W = R_Z R_Y R_X * u_T \tag{23}$$

Now, the $[X, Y, Z]$ and u_W are the ground truth data from the robot. To know the exact racket pose error, we manually control the robot to achieve 50 different poses with various position or Euler angles, and compute the racket pose from robot and cameras. Those angle between two norm vectors from the robot and cameras are defined as the orientation error. As shown from Fig. 9 , the position error is below 13 mm with an average of 7.8 mm and the orientation error is under 15 with 7.2 average value.

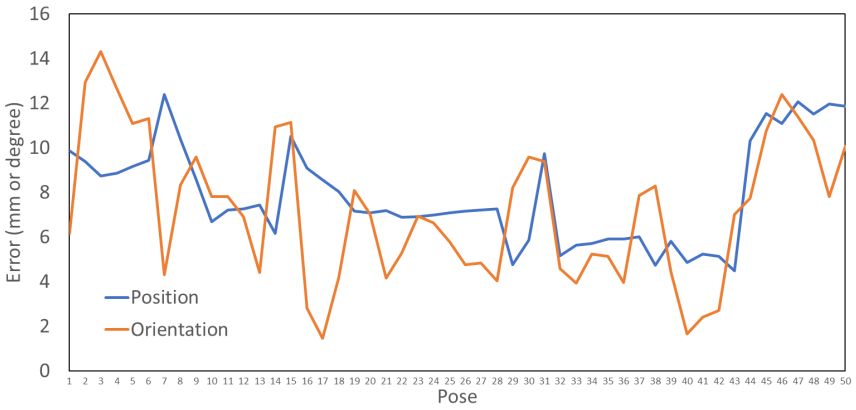


Fig. 9. Racket pose evaluation.

4.2. *Comparison with human pose estimation*

We directly apply the Convolutional Pose Machines (CPMs) to extract the human body keypoints including ear, eye, nose, neck, shoulder, elbow, wrist and hip (14 keypoints in total) in the left camera. A Kalman filter is used to track these keypoints. Human poses are calculated and stored as matrices to express which parts of the body are connected to each other. The visualization is shown in Fig. 10.

By means of the same dataset and classification approach with different input shape 30 frames × 14 keypoints, we can obtain the test accuracy of 98.4% , which is similar to the proposed method 98.7%.

However, CPMs has a crucial issue of hardware consumption. It can not satisfy the real-time requirement in table tennis. Meanwhile, it just provides the approximate pose information that can not be used to calculate the exact 3D position or orientation. In contrast, our proposed method can be run in 10 ms (100 FPS) and give the opportunities to train the robot having a human-like movement.

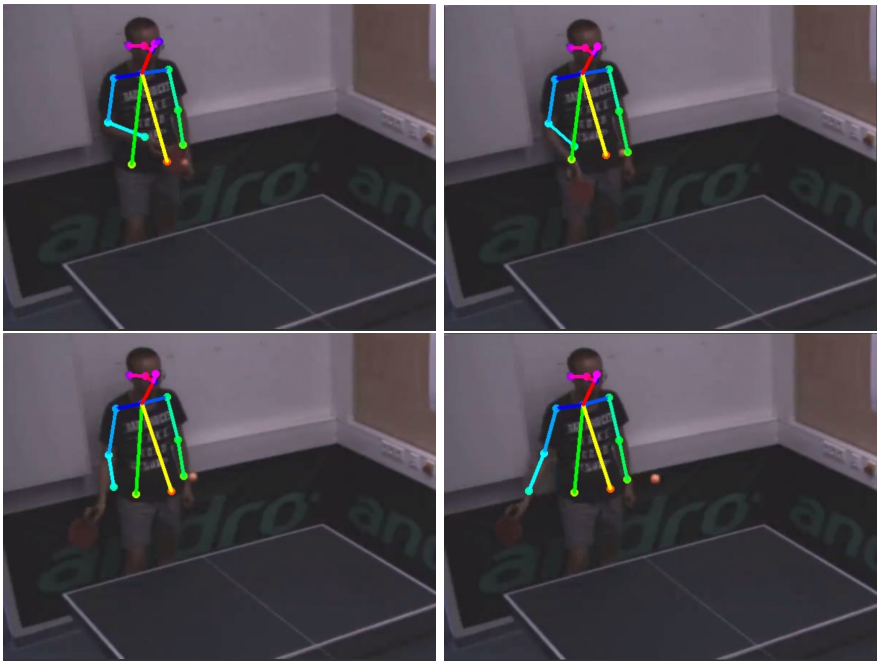


Fig. 10. Human pose estimation in image sequences.

5. Conclusions

In this paper, we have presented a novel table tennis racket pose detection method based on stereo vision. Through the color and motion segmentation, we can extract the racket contours, then feed them into the proposed wide baseline stereo matching method to generate the 6D pose. With a multilayer perceptron (MLP) neural network, the pose trajectories can be classified into five kinds of spin types. Finally, two experiments are performed to evaluate the accuracy of pose detection and classification. In the future, we will teach the KUKA robot to mimic a human-like movement by imitation learning aiming to cope with various spin.

Acknowledgment

This work was supported in part by the Vector Stiftung and KUKA.

References

[1] K. Mlling, J. Kober and J. Peters, A biomimetic approach to robot table tennis, in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* Oct 2010, pp. 1921–1926.

[2] K. Zhang, Z. Fang, J. Liu, Z. Wu and M. Tan, Fusion of vision and imu to track the racket trajectory in real time, in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)* Aug 2017, pp. 1769–1774.

[3] C. H. Lampert and J. Peters, Real-time detection of colored objects in multiple camera streams with off-the-shelf hardware components, *Journal of Real-Time Image Processing* **7** 31–41 (Mar 2012).

[4] Y. Huang, B. Schlkopf and J. Peters, Learning optimal striking points for a ping-pong playing robot, in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* Sept 2015, pp. 4587–4592.

[5] H. Li, H. Wu, L. Lou, K. Khnlenz and O. Ravn, Ping-pong robotics with high-speed vision system, in *2012 12th International Conference on Control Automation Robotics Vision (ICARCV)* Dec 2012, pp. 106–111.

[6] Y. Zhang, R. Xiong, Y. Zhao and J. Wang, Real-time spin estimation of ping-pong ball using its natural brand, *IEEE Transactions on Instrumentation and Measurement* **64** 2280–2290 (Aug 2015).

[7] H. S. Hiroshi Ohya, Tracking racket face in tennis serve motion using high-speed multiple cameras, in *2004 Joint 2nd International Conference on Soft Computing and Intelligent Systems and 5th International Symposium on Advanced Intelligent Systems* Sept 2004.

[8] N. Elliott, S. Choppin, S. Goodwill, T. Senior, J. Hart and T. Allen, Single view silhouette fitting techniques for estimating tennis racket position, *Sports Engineering* **21** 137–147 (Jun 2018).

[9] G. Chen, D. Xu, Z. Fang, Z. Jiang and M. Tan, Visual measurement of the racket trajectory in spinning ball striking for table tennis player, *IEEE Transactions on Instrumentation and Measurement* **62** 2901–2911 (Nov 2013).

[10] P. Blank, J. Hoßbach, D. Schuldhaus and B. M. Eskofier, Sensor-based stroke detection and stroke type classification in table tennis, in *Proceedings of the 2015 ACM International Symposium on Wearable Computers ISWC '15*, (ACM, New York, NY, USA, 2015), pp. 93–100.

[11] J. Tebbe, Y. Gao, M. S. Rienitz and A. Zell, A table tennis robot system using an industrial kuka robot arm, in *German Conference on Pattern Recognition* (Springer, Stuttgart, Germany, 2018). in press.

[12] B. Micusik and H. Wildenauer, Descriptor free visual indoor localization with line segments, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* June 2015, pp. 3165–3173.

[13] R. Lane and N. Thacker, Tutorial: Overview of stereo matching research, *Imaging Science and Biomedical Engineering Division, Medical School, University of Manchester* (1998).

[14] S. A. K. Tareen and Z. Saleem, A comparative analysis of sift, surf, kaze, akaze, orb, and brisk, in *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)* March 2018, pp. 1–10.

[15] H. Hirschmuller, Accurate and efficient stereo processing by semi-global matching

and mutual information, in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* **2**June 2005, pp. 807–814 vol. 2.

[16] S. Tulyakov, A. Ivanov and F. Fleuret, Practical deep stereo (pds): Toward applications-friendly deep stereo matching, *CoRR* **abs/1806.01677** (2018).

[17] W.-T. Chu and S. Situmeang, Badminton video analysis based on spatiotemporal and stroke features, in *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval ICMR '17*, (ACM, New York, NY, USA, 2017), pp. 448–451.

[18] R. Srivastava, A. Patwari, S. Kumar, G. Mishra, L. Kaligounder and P. Sinha, Efficient characterization of tennis shots and game analysis using wearable sensors data, in *2015 IEEE SENSORS* Nov 2015, pp. 1–4.

[19] A. Bearman and C. Dong, Human pose estimation and activity classification using convolutional neural networks, *CS231n Course Project Reports* (2015).

[20] G. Bradski, The OpenCV Library, *Dr. Dobb's Journal of Software Tools* (2000).

[21] M. A. Fischler and R. C. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, *Commun. ACM* **24** 381–395 (June 1981).

[22] S.-E. Wei, V. Ramakrishna, T. Kanade and Y. Sheikh, Convolutional pose machines, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* 2016, pp. 4724–4732.

Appendix E

Robust Stroke Recognition via Vision and IMU in Robotic Table Tennis

Title

Robust Stroke Recognition via Vision and IMU in Robotic Table Tennis

Authors

Yapeng Gao, Jonas Tebbe, and Andreas Zell

Published in

International Conference on Artificial Neural Networks

Time of publication

September 2021

DOI

10.1007/978-3-030-86362-3_31

Robust Stroke Recognition via Vision and IMU in Robotic Table Tennis*

Yapeng Gao[1], Jonas Tebbe[1], and Andreas Zell[1]

Cognitive Systems, Eberhard Karls University Tübingen, Germany
{yapeng.gao,jonas.tebbe,andreas.zell}@uni-tuebingen.de
http://www.cogsys.cs.uni-tuebingen.de/

Abstract. Stroke recognition in table tennis is a challenging task, due to the variety of the movements. Many different sensors have been adopted in robotic table tennis, with the goal of detecting the players' movements. In this paper, we propose a two-stage approach to directly recognize the table tennis racket's movement. A bounding box around the racket can be extracted from an RGB image in the first stage. An efficient and lightweight CNN architecture is then developed to regress the racket 3D position by fusion of the cropped image and the 3D rotation data from an IMU in the second stage. Together with the rotation data, a robust 6D racket pose is available at a frame rate of 100 Hz. In the experiments, two datasets are collected from our KUKA table tennis robot for evaluation and comparisons, which show a position error of 4.7 cm at a range of 6 m. One behavior cloning experiment is performed in order to reveal the potential of this work.

Keywords: Racket Pose Estimation · Sensor Fusion · Table tennis Robot.

1 Introduction

Human activity detection has spawned a large amount of research in many applications, such as gesture recognition, video surveillance, health care and sports performance analysis. Typically, it includes two steps: feature extraction and action classification. In recent years, a variety of sensors have been applied to obtain the human pose, thereby resulting in different kinds of techniques.

Vision-based methods extract the 2D human joints [6], hand keypoints [18] or 3D human pose [16] as features from RGB cameras. To get more accurate information, the depth maps from RGB-D sensors are included to derive the full 3D human pose [29]. Motion sensor based methods adopt low-cost accelerometers, gyroscopes, and sometimes magnetometers to detect the human's linear acceleration and angular velocity [28] as features. With the fusion of multiple inertial measurement units (IMUs) and a single camera, one can recover accurate 3D human pose in the wild [14].

To understand the performance of the players and provide them with a guide to tactics and skills, some systems with different sensors have been designed for

* Supported by the Vector Stiftung and KUKA.

Fig. 1: Playing with our KUKA table tennis robot. A wearable IMU is mounted at the bottom of the player's racket handle. The quaternion value q_{IMU} streamed from it, is defined as the racket orientation in the IMU frame. One of the stationary cameras fixed on the ceiling is used to capture the human player movements from above (Fig. 2 left). By fusing the images with IMU signals, we can take them as inputs and regress the 3D racket position robustly with the proposed approach. The camera and the IMU are synchronized with a software trigger.

sports. An AI Coach system for athletic training [23] is built with a single camera. They design a binary player detector to extract a single player as bounding box in the first frame. To accelerate the detection step, a tracking model based on the detected bounding box is used from the second frame to the last frame. After knowing each player's tubelet, the player 2D pose can be regressed by a pose estimation model. In order to estimate and track player's 3D pose, Bridgeman et al. [5] calculate the correspondences between 2D poses in different camera views. The 2D pose associations can be used to generate the player 3D skeletons.

In robotic table tennis we face many challenges, especially due to the movement of the human opponent, also including some deceptive actions. Each movement creates different types of spin and speed. Therefore, instead of recognizing the human 2D or 3D pose, the main focus in this paper is the table tennis racket pose estimation. This gives our table tennis robot (shown in Fig. 1) the ability to recognize the human stroke pose and consequently mimic the human motion with imitation learning. To achieve this we use a single camera fused with an IMU and develop a novel approach for robustly recognizing human strokes. The main contributions of this paper are as follows:

- We propose a novel two-stage position estimation network for table tennis rackets via vision and IMU. Together with the 3D rotation data retrieved from the IMU, a robust 6D racket pose is available at a frame rate of 100Hz without any special markers.
- The training dataset is created based on simulated views of a racket CAD model. The evaluation dataset is collected from our KUKA robot, which

can be annotated automatically with the pre-calibrated transformation matrix between the robot and the camera. Therefore, manually labeling is not needed in our work.
– The experiment shows that our approach achieves the best performance with a position error of 4.7 cm at a range of 6 m. To reveal the goal of this work, we perform an experiment to operate the robot in a human-like way, which is a clone of the human movements.

2 Related work

Image-based 6D object pose estimation is one of the trendiest topics in computer vision. Recent state-of-the-art methods have shown huge success in detecting the 6D pose of objects in close range to the camera. PoseCNN [26] directly estimates the 6D object pose with an end-to-end network from a single image. Sundermeyer et al. [20] present an implicit method for 3D orientation estimation based on Augmented Autoencoders (AAEs), which is trained on synthetic images. The 3D translation is then computed according to the pinhole camera model. A pixel-wise voting network (PVNet) [17] localizes 2D keypoints on the object using RANSAC and aligns them with 3D keypoints to obtain the 6D pose. The Coordinates-based Disentangled Pose Network (CDPN) [11] uses a Dynamic Zoom In (DZI) technique to compensate the 2D object detection error, which achieves accurate and robust results. However, if the object is too small in the camera or, like the racket, has a texture-less surface and very thin paddle, it is prone to failure using these methods, because of insufficient features.

By labeling special markers on the racket, Zhang et al. [27] could use color thresholding to extract them from two cameras, and the initial racket pose is then computed by the perspective-n-point (PnP) method. To generate a robust pose, they employed an IMU sensor and fused all of the data by means of an extended Kalman filter (EKF), which lead to a 1.1° rotation error. They don't test the position error since there is no dataset available. Gao et al. [8] employ a markerless method by segmenting the racket red side contours from stereo cameras. A stereo matching method is used to align the points on the contours. The final position error is 7.8 mm and the rotation error is 7.2°. Omron [10] puts 9 small and round markers on each racket side for their Forpheus robot, which can accurately predict the moving direction of the racket based on a high-speed camera. However, these methods are neither convenient nor robust, since they are sensitive to the color and brightness and need to be manually adjusted to find the better color thresholding values.

Inspired by the aforementioned methods, we decompose the 6D pose into position and rotation components. A wireless IMU mounted at the bottom of the racket handle is continuously streaming rotation data. By deeply fusing the IMU information and the camera images, a novel CNN-based method is proposed. The output is the racket 3D position and it is trained fully based on a synthetic dataset.

3 Methodology

3.1 Overview

IMUs are widely used in wearable devices to measure human activity in real-time and with high accuracy. In this paper, we mount a MetaMotionR (MMR) IMU [15] at the bottom of the racket handle, as shown in Fig. 2. With Bosch sensor fusion technology [4], the MMR sensor can provide robust linear acceleration and quaternion values via Bluetooth 4.0 at 100 Hz. Kristen Beange [3] has assessed the MMR sensors, which have a robust performance at 1° error in all axes when considering the absolute angle orientation. They compare the IMU with an optical motion capture equipment (Vicon Motion Systems) during controlled, repetitive sinusoidal motion at frequencies of 20 cpm and 40 cpm (i.e., 0.33Hz and 0.67 Hz, respectively). Therefore, we mainly focus on the 3D racket position estimation by fusing the IMU and camera in this part.

To estimate the racket position of the human player, we propose a novel approach, as shown in Fig. 2. Compared to the single-stage object pose estimation, two-stage methods usually comprise one step for object detection and another for pose regression, which leads to a very fast inference time and is well suited for the real-time operation in sports. The first stage can be easily replaced with

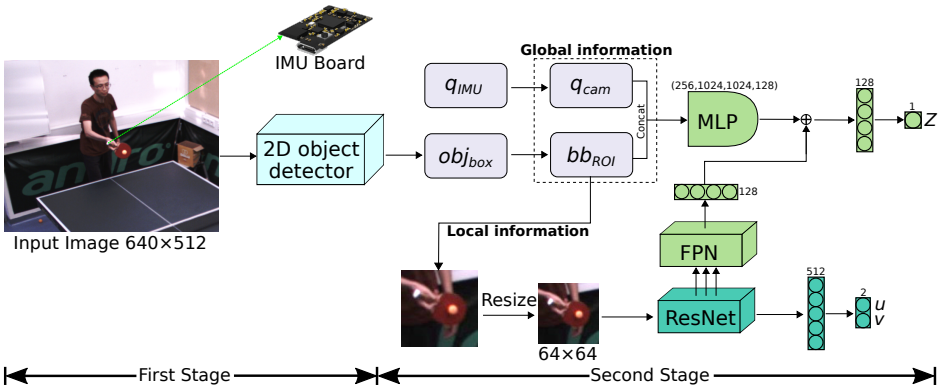


Fig. 2: CNN architecture for the racket position estimation during testing in our scenario. The rotation q_{IMU} is read from a wireless IMU as a 4D quaternion in the IMU frame. It is transformed to the camera frame as q_{cam}. The images with 640×512 pixels are first fed into a pre-trained 2D object detector in order to find the racket bounding box $obj_{box}$and its position $[x_c, y_c, h, w]$ in pixels. A new region of interest bb_{ROI}, $[x'_{min}, y'_{min}, h', w']$, is computed to compensate the 2D object detection error by Eq. (2). Then quaternions and bb_{ROI} together with the image crops are fed into different network layers in order to extract the global and local features, respectively. The last fully-connected layers output the racket depth Z and the 2D projection point $[u, v]$ of the racket 3D centroid. Finally, X and Y positions can be reconstructed with Eq. (1).

any state-of-art method along the development of the 2D object detection in the future.

The outputs of our architecture are the depth component Z and the local 2D projection point $[u, v]$ of the racket 3D centroid. Then we can indirectly derive the entire 3D position $[X, Y, Z]$ with the equation below:

$$X = \frac{(x'_{min} + u - c_x)\, Z}{f_x}, \quad Y = \frac{(y'_{min} + v - c_x)\, Z}{f_y} \tag{1}$$

where x'_{min}, y'_{min} are the left upper corner in bb_{ROI}. f_x, f_y are the focal lengths in pixels, $[c_x, c_y]$ is a principal point. Here $[u, v]$ is different from $[x_c,\ y_c]$ which is provided from the object detector, since the later one is not the exact centroid but the center of the detected bounding box. This will affect the $[X, Y]$ a lot when having a large depth Z (from 2.6 m to 5.3 m in our case). Therefore, the position regression problem is decomposed into the following two sub-tasks.

3.2 Racket Centroid Extraction

In order to detect the racket in images, we employ a self-pretrained YOLOv4 [2] model, which is a very fast and accurate one-stage object detector. It can generate a 4-D vector obj_{box} localizing the racket as a 2D bounding box. The obj_{box} is composed of the rectangle center x_c, y_c, height h and width w in image coordinates. To tolerate detection errors and make the subsequent estimation more robust and accurate, we dynamically adjust the obj_{box} to a new region of interest $bb_{ROI} = [x'_{min}, y'_{min}, h', w']$ during training. The bb_{ROI} is computed by the following equations:

$$\begin{cases} s = max(h, w) \\ N = randint(-\alpha s, \alpha s) \\ (h', w') = (\beta s + s, h') \\ (x'_c, y'_c) = (x_c, y_c) + N \\ (x'_{min}, y'_{min}) = (x'_c, y'_c) - 0.5(h', w') \end{cases} \tag{2}$$

where s is the maximum value in h and w. α and β are coefficients to control the center noise N and corner offsets, which are equal to 0.2 and 1.5, respectively. N is a 2D vector of integers, randomly chosen from $-\alpha s$ to αs during training and evaluation, while is set to zero during testing. The resulting obj_{box} has a square size and keeps the same aspect ratio as before.

Finally, it is scaled to the size of 64×64 as the input for the ResNet (see Fig. 2). This *Dynamic Resize* technique is based on the Dynamic Zoom In (DZI) in [11]. In contrast to the DZI that enlarges the crops, here we simply shrink them, since the texture-less surfaces on the racket contain many similar features and it has little influence to the centroid regression. An example with a synthetic image for training is shown in Fig. 3. Then a ResNet18 [9] is deployed to extract the deep features, followed by two dense layers with 512 and 2 units, respectively as shown at the bottom of Fig. 2.

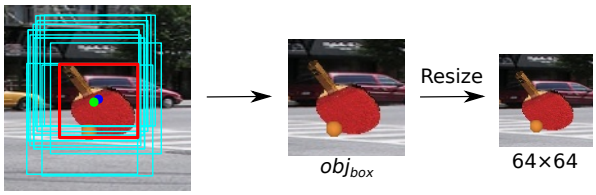


Fig. 3: An example for *Dynamic Resize* during training. **Left**: The detected bounding box (red) from YOLOv4 and the dynamically computed *ROI* candidates (cyan). The bounding box center $[x_c,\ y_c]$ and the racket centroid $[u, v]$ are marked as blue and green circle, respectively **Middle**: the randomly selected bb_{ROI} for training. **Right**: the final resized crop.

3.3 Depth Regression

Next, we propose a novel deep fusion approach for the depth Z regression. Intuitively, if we know the bounding box positions in images, the racket 3D position could be estimated by the given camera intrinsics $[f_x, f_y, c_x, c_y]$. However, these positions will change with different orientations and especially if there are occlusions or truncations. To avoid these problems, [25] runs a RetinaNet [12] on the input images and concatenates the generated RoIAlign features and bounding box information as joint features, which are used for translation regression. RoIAlign features are only for predicting rotation. It is a one-stage vehicle pose method, and not sufficiently fast and accurate for sports.

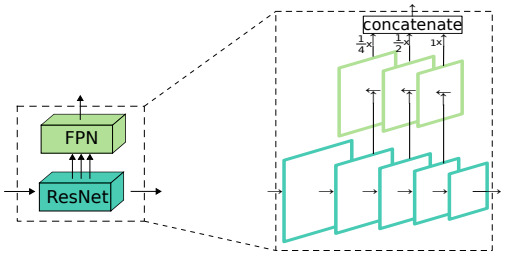


Fig. 4: ResNet-FPN (Feature Pyramid Network).

Inspired from it, we consider the combination of the rotation value q_{cam} and bb_{ROI} as the global features, which are fed into a 4-layer MLP network with 256, 1024, 1024, 128 units separately. The local features indicating the racket local pixel position, size and occlusions, are concatenated with the global network (via $\bigoplus$ in Fig. 2). A Resnet-FPN network (in Fig. 4) is used for extracting local features, since it includes multi-scale features and can recover the scale ratio information when resizing the *ROI* crops to 64×64. Finally, the depth Z is retrieved as output of a 128-D dense layer.

To train the whole networks, we design a joint position loss function $\mathcal{L}_{\text{pos}}$ to optimize the centroid detection and depth regression as follows:

$$\mathcal{L}_{\text{pos}} = \gamma_1 \cdot |Z - \hat{Z}| + \gamma_2 \cdot \left\| C_{pos} - \hat{C}_{pos} \right\|_1 \tag{3}$$

where Z and $\hat{Z}$ are representing the estimated and ground-truth depth. C_{pos} and $\hat{C}_{pos}$ are the estimated and true centroid pixel positions. γ_1 and γ_2 are used to balance the different errors.

4 Experiments

4.1 Dataset

To train the proposed model, we create a synthetic dataset which can be labeled automatically. A racket CAD model is first reconstructed from a real racket with the free, open-source reconstruction software Meshroom [24], based on the structure from motion (SfM) technique. This results in a reconstructed 3D mesh in Fig. 5 left. Then post-processing is used to remove the background, fill the holes, smooth the surface, blend vertex color, scale the model size, and change the coordinates in Meshlab [7]. The final high-quality 3D model is shown in Fig. 5 right.

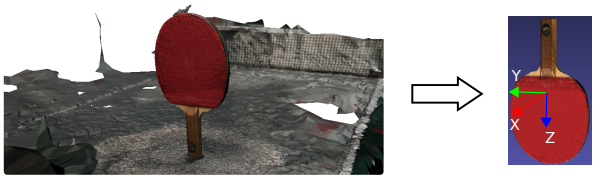
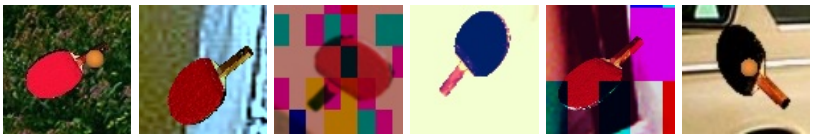


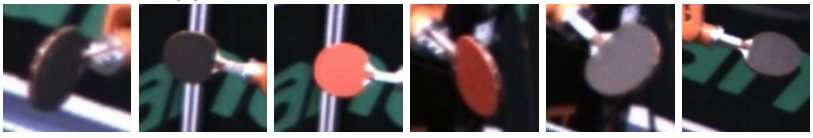
Fig. 5: **Left**: Reconstructed mesh with background from multiple views using Meshroom software. **Right**: the final CAD model with its coordinates.

By using domain randomization (DR) [22], we can generate a set of synthetic images as well as their 6D pose. The racket CAD model is placed in a simulated scene at random positions and rotations. Then, each one is projected into the image plane as the foreground, with a known bounding box. The images from the Pascal VOC dataset are embedded as the background. Each synthetic image is rendered with a random light source position and diffuse reflection. Other techniques, like Gaussian noise, motion blur, ping pong ball and occlusions, are included to reduce the "reality gap". A few examples are presented in Fig. 6(a). Meanwhile, the annotations, including the bounding box positions, racket centroids in pixels and the racket 6D pose, are collected from the simulated 3D scene as the ground truth tags, which are then used to train the object detector and the position regression model, respectively. 50,000 training patterns are collected finally. The resulting range of the depth Z is $[2.6m, 5.3m]$.

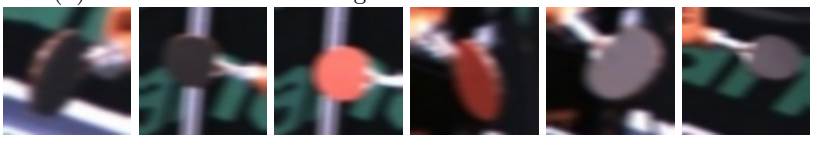
For evaluation dataset collection, a usual way that we tried was mounting multiple reflection markers on the racket and then capturing the human player motions with an OptiTrack systems. However, the markers must be placed at the surface in a critical requirement, which would result in many occlusions in images. Therefore, one convenient method is to make use of our KUKA robot that has a racket at the end-effector. This racket differs slightly from the rendered CAD model such that this can also test the robustness against multiple rackets. Another stationary camera opposite to the robot is used to take the images. By moving the robot to given positions and rotations, we collect an evaluation dataset of 208 images (Fig. 6(b)). To obtain the correct pose with respect to the camera coordinate frame, we first calculated the transformation matrix between the robot and the camera by the hand-eye calibration method [21]. The resulting range of the depth Z is from 2.8 m to 5.2 m. To simulate a fast moving racket, we manually apply motion blur (Fig. 6(c)) with a 7×7 kernel on each image for the following comparisons. Due to the high frame rates and fast shutter speed of the cameras, motion blur is actually imperceptible in our case.



(a) Training dataset from synthetic images



(b) Evaluation dataset using another racket in our KUKA robot



(c) Applying motion blur with a 7×7 kernel

Fig. 6: Cropped examples for training and evaluation of the racket position estimation.

4.2 Training and Inference

As shown in Fig. 2, we need to train two separate models one by one for the different stages. To make the first stage (Yolov4) faster, we resize the network input to 512x512 and change the trained model from darknet [2] to the tkDNN [13] framework. The activation function used in the second stage is ReLU [1]. The last two 128-D dense layers for depth Z regression are activated by leaky

ReLU, with a negative slope 0.02. The outputs Z and $[u, v]$ are activated by the logistic sigmoid function. All the inputs are normalized for better performance. To avoid overfitting, we freeze the parameters in the first 4 residual blocks of the ResNet during the beginning 40 epochs. The other hyperparameters are given in the table below:

Table 1: Hyperparameters separately for different models.

	optimizer	epochs	batch size	learning rate
2D object detector	Adam	100	16	3e-3
Position regression	RAdam	100	4	1e-4

The training is processed by a host computer with an NVIDIA RTX 2080Ti GPU, a 3.0GHz Intel i7-97000 CPU and 32GB RAM. Each bounding box is extracted by Yolov4 in 7.8 ms, then the depth Z and the centroid $[u, v]$ can be regressed in 1.7 ms. The overall inference rate is around 100 Hz.

4.3 Evaluation

The mAP (mean Average Precision) by Yolov4 is 86.9% for an IoU threshold of 0.5 in the evaluation dataset. To evaluate the position estimation accuracy, we use two metrics: position error E_{trans}, and $\leq$5cm. In the $\leq$5cm metric, a pose is considered correct if the position error is within 5cm. Due to some other approaches having large position errors, we extend $\leq$5cm to a third metric: $\leq$10cm.

Table 2: Evaluation for racket position estimation.

	Sensors	E_{trans}	$\leq$5cm	$\leq$10cm
Zhang et al. [27]	camera,IMU,marker	-	-	-
Gao* et al. [8]	stereo cameras	2.8 cm	91.8%	100.0%
AAEs [20]	single camera	39.1 cm	6.7%	17.3.0%
CDPN [11]	single camera	36.6 cm	7.5%	21.8%
R. Staszak [19]	single camera	23.5 cm	10.6%	25.0%
OUR (no FPN)	single camera, IMU	6.8 cm	48.6%	85.1%
OUR (with motion blur)		5.2 cm	60.1%	93.2%
OUR		**4.7 cm**	**65.0%**	**95.5%**

In Table 2, we compare our method with current research in which different sensors are used. Zhang et al. [27] did not show the position error, since they did

not have a dataset for evaluation and their method is not compatible with our dataset. The remaining methods are trained and evaluated in our dataset. In order to use stereo cameras in Gao et al. [8], we expand the evaluation dataset by the second well-calibrated camera. Instead of using the color thresholding method to detect the red surface, we extract the racket center either on the red side or on the black side by our centroid regression model. The * indicates it is used with modifications. The resulting performance is the best one. However, it will take twice as much time as ours' and can not extract the rotation value robustly and accurately. Moreover, it needs more effort to pre-calculate the transformation matrix between these two cameras. To get a fair comparison, we replace the rotation head with the true value and only use the translation head in [11, 19, 20]. Among them, [20] and [11] obtain the 3D position under two assumptions: the bounding box size is linearly affected only with respect to the depth Z, and is therefore never changed when having the same Z. These assumptions lead to a large position error when the object is far away from the camera (6m distance in our case). Although [19] has a bit better results, they still did not take the global pixel positions of the bounding box into consideration. In comparison, our method achieves a more robust performance with the second best accuracy.

Furthermore, two additional experiments, with motion blur (in Fig. 6(c)) and without FPN layers, are performed to simulate a moving racket and do an ablation study, respectively. Fig. 7 shows four examples with different movements. To demonstrate this work, we apply the human movements to our KUKA robot with coordinate transformation. The robot uses the penhold grip while playing since it is more flexible and controllable than the shakehand style in our scenario, as shown in the video `https://youtu.be/U2YPh_ZwQxQ`

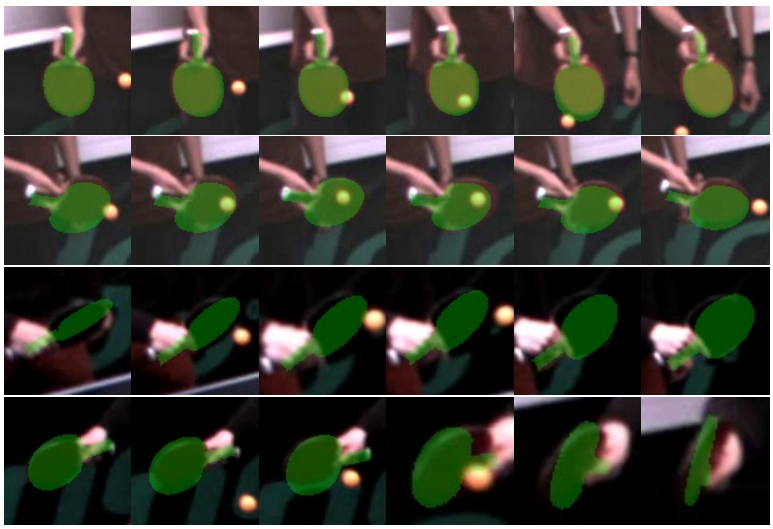


Fig. 7: Four stroke movements for the racket pose estimation.

5 Conclusion

In this paper, we proposed a novel approach for stroke recognition via a camera and IMU. We generated several datasets for training and evaluation. The experiment has shown the proposed method gives a robust performance. With the main goal of improving the capabilities of our table tennis robot in mind, we are planning to apply our approaches to human stroke examples and make the table tennis robot hit the ball by imitating the human movements. In addition, we could also predict the ball's flying trajectory by analyzing the racket pose, since our approach can be run at 100 Hz.

However, our approach is going to fail if the detected bounding box is wrong in the first stage. For example, the player's left hand could also be recognized as a racket if there are some circle patterns in the background, as shown in the demo video. In this case, we could utilize the tracking method to identify the coherent relations between frames.

References

1. Agarap, A.F.: Deep learning using rectified linear units (relu). CoRR **abs/1803.08375** (2018), `http://arxiv.org/abs/1803.08375`
2. Alexey Bochkovskiy, Chien-Yao Wang, H.Y.M.L.: Yolov4: Yolov4: Optimal speed and accuracy of object detection. arXiv (2020)
3. Beange, K.: Validation of Wearable Sensor Performance and Placement for the Evaluation of Spine Movement Quality. Ph.D. thesis, University of Ottawa (2019)
4. Bosch : Bosch, `https://www.bosch-sensortec.com/`
5. Bridgeman, L., Volino, M., Guillemaut, J., Hilton, A.: Multi-person 3d pose estimation and tracking in sports. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). pp. 2487–2496 (2019)
6. Cao, Z., Hidalgo, G., Simon, T., Wei, S.E., Sheikh, Y.: OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. In: arXiv preprint arXiv:1812.08008 (2018)
7. Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G.: Meshlab: an open-source mesh processing tool. In: Eurographics Italian chapter conference. vol. 2008, pp. 129–136 (2008)
8. Gao, Y., Tebbe, J., Krismer, J., Zell, A.: Markerless racket pose detection and stroke classification based on stereo vision for table tennis robots. In: 2019 Third IEEE International Conference on Robotic Computing (IRC). pp. 189–196 (Feb 2019). https://doi.org/10.1109/IRC.2019.00036
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016)
10. Kawakami, S., Ikumo, M., Oya, T.: Omron table tennis robot forpheus, `https://www.omron.com/innovation/forpheus.html`
11. Li, Z., Wang, G., Ji, X.: Cdpn: Coordinates-based disentangled pose network for real-time rgb-based 6-dof object pose estimation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (October 2019)
12. Lin, T., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: 2017 IEEE International Conference on Computer Vision (ICCV). pp. 2999–3007 (2017)

13. M. Verucchi, L. Bartoli, F.B.F.G.P.B., Bertogna, M.: Real-time clustering and lidar-camera fusion on embedded platforms for self-driving cars. In: 2020 Fourth IEEE International Conference on Robotic Computing (IRC) (2020)
14. von Marcard, T., Henschel, R., Black, M.J., Rosenhahn, B., Pons-Moll, G.: Recovering accurate 3d human pose in the wild using imus and a moving camera. In: Computer Vision – ECCV 2018. pp. 614–631. Springer International Publishing, Cham (2018)
15. mbientlab: mbientlab, `https://mbientlab.com/`
16. Pavllo, D., Feichtenhofer, C., Grangier, D., Auli, M.: 3d human pose estimation in video with temporal convolutions and semi-supervised training. In: Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
17. Peng, S., Liu, Y., Huang, Q., Zhou, X., Bao, H.: Pvnet: Pixel-wise voting network for 6dof pose estimation. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019. pp. 4561–4570 (2019). https://doi.org/10.1109/CVPR.2019.00469
18. Simon, T., Joo, H., Matthews, I., Sheikh, Y.: Hand keypoint detection in single images using multiview bootstrapping. In: CVPR (2017)
19. Staszak, R., Belter, D.: Hybrid 6d object pose estimation from the rgb image. In: ICINCO (2019)
20. Sundermeyer, M., Marton, Z.C., Durner, M., Brucker, M., Triebel, R.: Implicit 3d orientation learning for 6d object detection from rgb images. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 699–715 (2018)
21. Tebbe, J., Gao, Y., Sastre-Rienietz, M., Zell, A.: A table tennis robot system using an industrial kuka robot arm. In: Brox, T., Bruhn, A., Fritz, M. (eds.) Pattern Recognition. pp. 33–45. Springer International Publishing, Cham (2019)
22. Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: Domain randomization for transferring deep neural networks from simulation to the real world. In: 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS). pp. 23–30. IEEE (2017)
23. Wang, J., Qiu, K., Peng, H., Fu, J., Zhu, J.: Ai coach: Deep human pose estimation and analysis for personalized athletic training assistance. pp. 374–382 (10 2019). https://doi.org/10.1145/3343031.3350910
24. Wang, Z., Li, J.Z.: Text-enhanced representation learning for knowledge graph. In: Ijcai. pp. 1293–1299 (2016)
25. Wu, D., Zhuang, Z., Xiang, C., Zou, W., Li, X.: 6d-vnet: End-to-end 6dof vehicle pose estimation from monocular rgb images. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). pp. 1238–1247 (2019)
26. Xiang, Y., Schmidt, T., Narayanan, V., Fox, D.: PoseCNN: A convolutional neural network for 6d object pose estimation in cluttered scenes (2018)
27. Zhang, K., Fang, Z., Liu, J., Wu, Z., Tan, M.: Fusion of vision and imu to track the racket trajectory in real time. In: 2017 IEEE International Conference on Mechatronics and Automation (ICMA). pp. 1769–1774 (Aug 2017). https://doi.org/10.1109/ICMA.2017.8016085
28. Zhao, Y., Yang, R., Chevalier, G., Gong, M.: Deep residual bidir-lstm for human activity recognition using wearable sensors. CoRR **abs/1708.08989** (2017), `http://arxiv.org/abs/1708.08989`
29. Zimmermann, C., Welschehold, T., Dornhege, C., Burgard, W., Brox, T.: 3d human pose estimation in rgbd images for robotic task learning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. 1986–1992 (2018)