Sensor Fusion and Stroke Learning in Robotic Table Tennis

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. Yapeng Gao
aus Shanxi, China

Tübingen
2022

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

Tag der mündlichen Qualifikation:	03.06.2022
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Andreas Zell
2. Berichterstatter:	Prof. Dr. Andreas Schilling

For my family and friends

Abstract

Research on robotic table tennis is attractive for studying diverse algorithms in many fields, such as object detection, robot learning, and sensor fusion, as table tennis is full of challenges in terms of speed and spin. In this thesis, we focus on optimal stroke learning with sensor fusion for a KUKA industrial manipulator. Four high-speed cameras and an IMU are used for object pose detection. To learn an optimal stroke for the robot, a novel policy gradient approach is proposed.

Firstly, we develop a multi-camera calibration approach for wide-baseline camera pairs. The initial intrinsic and extrinsic transformations are computed using the classic calibration methods, resulting in a 3D position error of 15.0 mm for four cameras (11.0 mm for each stereo pair) in our test dataset. A novel loss function is proposed to post-optimize them with a new set of pattern images from each camera. The final accuracy is 3.2 mm for stereo cameras and 2.5 mm for four cameras. To efficiently use those cameras, we divide them into two stereo-camera pairs for the ball and racket detection, respectively. With the well-calibrated cameras, the 3D position of the ball can be triangulated when the pixel positions of the ball center are determined with two different approaches: color thresholding and two layers CNN.

Secondly, we propose an optimal stroke learning approach for teaching the robot to play table tennis. A realistic simulation environment is built for the ball's dynamics and the robot's kinematics. The learning strategy is decomposed into two stages: the ball hitting state prediction and the optimal stroke learning. Based on the controllable and applicable actions in our robot, a multi-dimensional reward function and Q-value model are proposed. The comparison with other RL methods is performed using an evaluation dataset of 1000 balls in simulation. An efficient retraining approach is proposed to close the sim-to-real gap. The testing experiments in reality show that the robot can successfully return the ball to the desired target with an error of around 24.9 cm and a success rate of 98% in three different scenarios.

Instead of training the policy in simulation, another option is initializing it with the actions of a human player and the corresponding state of the ball. To get the human actions, we directly detect the racket from images and estimate its 6D pose using two proposed approaches: traditional image processing with two cameras and deep learning by fusing one camera and an IMU. The experiment shows the latter method outperforms the former in terms of robustness for both the black and red sides of the racket. The former method is 1.9 cm better in position (2.8 cm versus 4.7 cm), but much slower in speed when the detection head is replaced with YOLOv4. Finally, a behavior cloning experiment is performed to reveal the potential of this work.

Kurzfassung

Die Forschung am Roboter-Tischtennis ist attraktiv für die Untersuchung diverser Algorithmen in vielen Bereichen, wie z. B. Objekterkennung, Roboterlernen und Sensorfusion, da Tischtennis voller Herausforderungen steckt, im Bezug auf Geschwindigkeit und Spin. In dieser Arbeit konzentrieren wir uns auf das Lernen optimaler Schläge mit Sensorfusion für einen KUKA Industriemanipulator. Vier Hochgeschwindigkeitskameras und eine IMU werden zur Erkennung der Objektlage verwendet. Um einen optimalen Schlag am Roboter zu lernen, wird ein neuartiger Policy-Gradient-Ansatz vorgeschlagen.

Zunächst entwickeln wir eine Multi-Kamera-Kalibrierung für Kamerapaare mit weiter Basislinie. Erste intrinsische und extrinsische Transformationen werden mit den klassischen Kalibrierungsmethoden berechnet, was zu einem 3D-Positionsfehler von 15,0 mm für vier Kameras (11,0 mm für jedes Stereopaar) in unserem Testdatensatz führt. Es wird eine neuartige Loss-Funktion vorgeschlagen, um mit einem neuen Satz von Musterbildern von jeder Kamera nachzuoptimieren. Die endgültige Genauigkeit beträgt 3,2 mm für Stereokameras und 2,5 mm für vier Kameras. Um diese Kameras effizient zu nutzen, teilen wir sie in zwei Stereokamera-Paare für die Ball- bzw. Schlägererkennung auf. Mit den kalibrierten Kameras kann die 3D-Position des Balls trianguliert werden, nachdem die Pixelpositionen der Balls mit zwei verschiedenen Ansätzen bestimmt wurden: durch Farbschwellenwertverfahren oder mit einem zweischichtigen CNN.

Zweitens schlagen wir einen optimalen Schlag-Lernansatz vor, um dem Roboter das Tischtennisspielen beizubringen. Es wird eine realistische Simulationsumgebung für die Dynamik des Balls und die Kinematik des Roboters entwickel. Die Lernstrategie wird in zwei Phasen unterteilt: die Vorhersage des Balltreffpunkts und das Lernen des optimalen Schlags. Basierend auf den steuerbaren und anwendbaren Aktionen in unserem Roboter werden eine mehrdimensionale Belohnungsfunktion und ein Q-Value-Modell vorgeschlagen. Der Vergleich mit anderen RL-Methoden wird anhand eines Evaluierungsdatensatzes von 1000 Bällen in der Simulation durchgeführt. Es wird ein effizienter Transfer-Learning-Ansatz vorgeschlagen, um die Lücke zwischen Simulation und Realität zu schließen. Die Testexperimente in der Realität zeigen, dass der Roboter den Ball mit einem Fehler von etwa 24,9 cm und einer Erfolgsrate von 98% in drei verschiedenen Szenarien erfolgreich zum gewünschten Ziel zurückschlagen kann.

Anstatt die Bewegungen in der Simulation zu trainieren, ist eine andere Option, sie anhand von Aktionen eines menschlichen Spielers und dem entsprechenden Zustand des Balls zu initialisieren. Um die menschlichen Aktionen zu erhalten, detektieren wir den Schläger direkt aus Bildern und schätzen seine 6D-Pose mit zwei vorgeschlagenen Ansätzen: traditionelle Bildverarbeitung mit zwei Kameras und Deep Learning durch

Fusion einer Kamera und einer IMU. Das Experiment zeigt, dass die letztere Methode die erstere in Bezug auf die Robustheit sowohl für die schwarze als auch für die rote Seite des Schlägers übertrifft. Die erstere Methode ist um 1,9 cm besser in der Position (2,8 cm gegenüber 4,7 cm), aber viel langsamer in der Geschwindigkeit, wenn der Erkennungskopf durch YOLOv4 ersetzt wird. Abschließend wird ein Imitations-Experiment durchgeführt, bei dem der Roboter die Bewegungen eines menschlichen Spielers nachahmt, um das Potenzial dieser Arbeit aufzuzeigen.

Acknowledgments

First I am deeply grateful to Prof. Andreas Zell for his guidance and supervision throughout my research over the years. I am especially grateful for the opportunity to join the table tennis robot project. Thank you for funding my research in the fifth year. I would also like to thank Prof. Andreas Schilling for his evaluations and suggestions of my research. I have learnt much from his lecture "Bildverarbeitung".

I am very grateful to my project partner and the co-author of my publications: Jonas Tebbe. I have benefited a lot from working with him in coding, writing, correcting, and discussing. I also thank other colleagues in this group: Chenhao Yang, Ya Wang, and Nuri Benbarka for the valuable discussion when I meet problems. Special thanks to Vita Serbakova, Klaus Beyreuther, and Uli Ulmer for helping me in my work.

I acknowledge the China Scholarship Council (CSC) for funding my research at the chair of Cognitive Systems in the University of Tübingen.

Finally, I am extremely grateful to my family for their love and support, especially my mother and my girlfriend Hong Gao.

Contents

Chapter 1

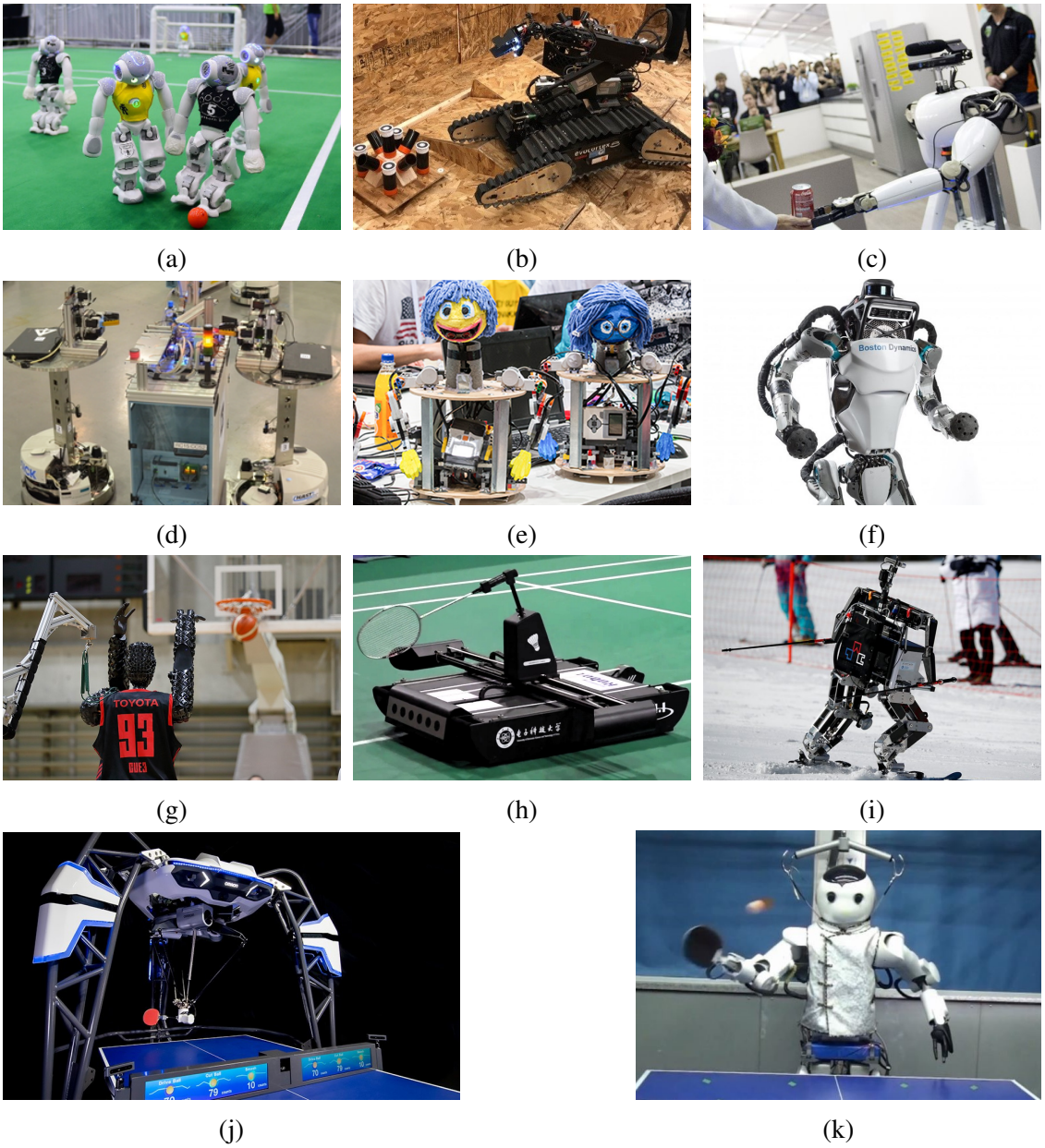
Introduction

1.1 Motivation

Robotics has become an important research area in many applications. With the increased capabilities of artificial intelligence (AI), robotic systems have brought significant changes to the world. For example, developing a robot that can play sports is a highly appealing field, which promotes the integration between robotics and AI through the human-machine or machine-machine interaction.

The Robot Soccer World Cup (RoboCup, Kitano *et al.* (1997)), an annual international robotics competition, was convened for soccer sport and recently has developed into four other major domains of competition that include Rescue League, Home, Logistics League, and Junior. These are shown in Fig. 1.1a–1.1e. With its advanced control system and state-of-the-art hardware, the recent Atlas humanoid robot (BostonDynamics, 2013) (see Fig. 1.1f), can not only perform jumps and spins, but also complete 360-degree twist and backflip like a gymnast. In Fig.1.1g, a basketball robot, CUE3 (Toyota, 2019), is shown, built by Toyota. It can achieve a 100 percent success rate with 2,020 successful shots. A mobile robot called Robomintoner (UESTC, 2016) is designed for the badminton game by integrating a racket as its hand, shown in Fig. 1.1h. To capitalize on the attention during the 2018 Winter Olympics, the host organization designed a robot skiing tournament with the requirement that all competitors should have a human-like body shape (see Fig. 1.1i).

Apart from the aforementioned sport robots, research on robotic table tennis is attractive for studying different algorithms in many fields, like object detection, robot learning, or sensor fusion, as table tennis faces some challenges in terms of speed and spin. Various robots integrated with different sensors have been proposed in recent years. Based on the concept of harmony between humans and machines, OMRON (Asai *et al.*, 2019) developed a table tennis robot that can accurately return the ball to the opponent table. A racket is mounted on a 6-axis industrial parallel link robot as its end-effector. Two servo systems are equipped to drive the robot and the racket. A position controller is used to update the racket position at 1 ms intervals. To measure the 3D ball position, they install two Quad-VGA industrial cameras that can run at 80 fps. All the hardware systems are synchronized by a machine automation controller. The future trajectory of the ball is



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k)

Figure 1.1: Some advanced robots designed for competitions and sports. (a) The humanoid robot NAO created by SoftBank Robotics. They are used in the RoboCup Standard Platform League. (b) The rescue robot that needs to autonomously find the simulated victims hidden in unstructured environments. (c) A service robot that is capable of autonomously accomplishing the required tasks. (d) Multi-robot collaborative assembly for industrial. (e) Dressed mobile robots in the RoboCup Junior OnStage challenge. They are designed by the junior students and can perform dance, storytelling, theatre, or other innovative actions. (f) Atlas humanoid robot with 28 compact mobile hydraulic joints from Boston Dynamics. (g)–(i) Sport robots created for basketball, badminton, skiing, respectively. (j) and (k) Robots designed for table tennis.

predicted based on an aerodynamic force model. In addition, three cameras are adopted during their latest exhibition to detect the human player's motion, as shown in Fig. 1.1j. The resulting distance error between the desired and the predicted landing position is 22.5 cm. Xiong *et al.* (2012) developed two identical 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 (Fig. 1.1k). The 3D ball position is estimated from two cameras. The spin of the ball can be extracted from a pan-tilt camera, which has a telephoto lens to detect the logo on the ball; see Zhang *et al.* (2014). All cameras can operate at 120fps with a resolution of 640×480.

Instead of designing a dedicated robot, some other researchers directly employ a robotic manipulator that facilitates the experimental environment setup. Mülling *et al.* (2011) mounted a 7-DOF Barrett WAM™ arm in a hanging position from the ceiling. The racket can be driven at high speed and acceleration. Similar to it, a 7-DOF robot arm is hung at the front of the table in Li *et al.* (2012). A PID controller is used to operate the racket in joint space. Considering the weight and the workspace of the robot, Liu *et al.* (2013) fix a 7-DOF Mitsubishi PA10 robot on the floor. Mahjourian *et al.* (2018) deploy a 5-DOF lightweight robot arm and a 1-DOF prismatic joint at the base of the robot assembly to provide more flexible control for the racket. In this work, we employ a 6-DOF KUKA industrial robot arm that is mounted on the floor and is aligned with the centerline of the table, as shown in Fig. 1.2.

While each robot in the above systems is different from others, they are facing some common challenges that are shown in Table 1.1. "✓" means this subject will be studied in this thesis. "•" means that it was implemented by an other member in our group and will not be presented in details. "−" denotes that we have not successfully implemented it yet. This thesis will mostly focus on three of those challenges that motivate our work in the following chapters.

Camera calibration is the fundamental part that significantly influences the further steps. Multiple cameras positioned in a particular way are employed to improve the calibration accuracy or to recognize different objects (Mülling *et al.*, 2011; Xiong *et al.*, 2012; Kawakami *et al.*, 2021). Intuitively, a human player can estimate the ball's hitting state by analyzing the pose of the opponent and the flying ball. Therefore, object pose estimation is necessary for understanding and predicting the state of each object, especially for the table tennis ball. When interacting with the robot, the racket motion of the human player provides more information that can estimate the ball spin and help the robot learn from human demonstrations. As a result, we mainly focus on the 6D pose detection for the racket instead of the human joints. The Reflexxes library (Kröger, 2011) is integrated to control the robot in Cartesian Space. To play table tennis with more intelligence, an end-to-end approach can be applied to the robot directly based on the corresponding state of the incoming ball. However, the training process is also extremely expensive for a non-spinning ball (Büchler *et al.*, 2020). Instead, we separate the learning into two stages: the hitting state prediction of the ball and the optimal stroke

Figure 1.2: Our robotic table tennis system with a 6-DOF KUKA robot arm. The racket is placed at the end effector in the penhold style. Four high-speed cameras and 12 LED panel lights are fixed to the ceiling.

Table 1.1: Current challenges for robotic table tennis.

	Subjects	Ours
Camera Calibration	Stereo Cameras	✓
	Multiple Cameras	✓
Object Pose Estimation	Ball Position	✓
	Human Pose	✓
	Racket Pose	✓
Ball Trajectory Prediction	3D Position	•
	Linear velocity	•
	Spin	•
Robot Control	Joint Space	–
	Cartesian Space	•
Robot Learning	Reinforcement Learning	✓
	Imitation Learning	–

learning based on the prediction. This two-stage algorithm can substantially accelerate the training process and is capable of dealing with different spin balls.

1.2 Contributions

The work in this thesis mainly addresses the development of robust and real-time approaches for ball and racket pose detection using various sensors and the learning of the optimal stroke motion for the robot. The specific contributions included in each of the four technical chapters are described as follows:

Chapter 3:

- A multi-camera calibration approach with a novel loss function is proposed, which can be solved by the sparse bundle adjustment approach.
- Based on the traditional image processing technique and the prior knowledge of the color and shape of the ball, an unsupervised method is developed to extract its 2D pixel position.
- Since a few other objects, such as the racket, the human hand and the KUKA robot, have similar colors and influence the result of the ball detection, we propose a CNN-based approach to segment the ball from the background.
- A dataset with the manually labeled annotations is presented. The experiment shows that our calibration method achieves high accuracy in millimeters. The CNN-based approach outperforms the unsupervised one in terms of accuracy.

Large parts of this work is based on the article published in:

1. Tebbe, J., Gao, Y., Sastre-Rienietz, M., & Zell, A. (2018). A table tennis robot system using an industrial kuka robot arm. In German Conference on Pattern Recognition (pp. 33-45). Springer.

Chapter 4:

- A simulation environment for robotic table tennis is designed to replicate the reality as closely as possible and provide a suitable platform for various algorithms comparison.
- Considering the physical and dynamic restriction of the real KUKA robot, we propose a novel approach to learn the optimal stroke based on reinforcement learning (RL).
- 10,000 different simulated ball trajectories are generated with a wide range of spin for training and a further 1000 balls for evaluation. Two new metrics are introduced to give an accurate and clear comparison.

- To apply the trained model in a new environment, we update the model with 10 new epochs subsequently, which is sufficient to bridge the cross-domain gap. The experiment, in reality, shows that the proposed method can achieve high performance.

Large parts of this work is based on the article submitted in:

2. Gao, Y., Tebbe, J., & Zell, A. (2021). Optimal Stroke Learning with Policy Gradient Approach for Robotic Table Tennis. arXiv preprint arXiv:2109.03100.

Chapter 5:

- By applying an unsupervised method similar to the ball detection, the racket contour in the red side can be extracted from two monocular cameras. The racket 3D position can be triangulated from the contour centers.
- To estimate the racket orientation, an efficient feature extraction approach is developed using the detected contours. Then the racket surface can be reconstructed from the feature pairs matched based on the racket size. The racket orientation is the normal unit vector of this surface.
- With a discrete Kalman filter, the racket pose can be tracked smoothly. A dataset is created by mounting a racket on the robot. The experiment shows the resulting position error is less than 13 mm, and the orientation error is under 15° for the red racket surface.

Large parts of this work is based on the articles published in:

3. Gao, Y., Tebbe, J., Krismer, J., & Zell, A. (2019). 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). IEEE.
4. Gao, Y., Tebbe, J., Krismer, J., & Zell, A. (2019). Real-time 6D Racket Pose Estimation and Classification for Table Tennis Robots. International Journal of Robotic Computing. 23-39. 10.35708/RC1868-126249.

Chapter 6:

- To robustly detect the 6D racket pose, we mount an inertial measurement unit (IMU) to the bottom of a racket handle. The orientation of the racket can be measured directly from the IMU at 100 HZ via Bluetooth.
- An object detector is utilized to extract the bounding box around the racket in images. By fusing the cropped racket image with the IMU, we can estimate the 3D racket position by the proposed position detector for both the red and black racket side.
- Two datasets are generated for training and evaluation. Several existing methods are compared with ours on the dataset. The resulting position error is around 4.7

cm at a range of 6 m. To reveal the potential of this work, we conduct one behavior cloning experiment in which the robot directly mimics the movements of a human player.

Large parts of this work is based on the article published in:

5. Gao, Y., Tebbe, J., & Zell, A. (2021). Robust Stroke Recognition via Vision and IMU in Robotic Table Tennis. In International Conference on Artificial Neural Networks (ICANN) (pp. 189-196). Springer.

Chapter 2

Background

This chapter provides fundamentals that are helpful to understand this thesis. First, we introduce our table tennis robot system that includes a KUKA Agilus R900 sixx robot, two laser scanners, and four industrial cameras. The robot is operated by a KR C4 compact controller via a data cable and a motor cable. One host PC is used to handle the data from all the sensors and transmit the command to the controller. We then explain the pinhole camera model that describes the camera intrinsic parameters and the extrinsic transformation related to the world coordinate system. In this thesis, we define the world coordinate system the same as the one in the robot. To estimate the object pose, we introduce two kinds of approaches: a single stage (end-to-end) and a two-stage method, which gives a faster detection process. Finally, deep reinforcement learning (DRL) algorithms for robotic learning are presented.

2.1 Robotic Table Tennis System

2.1.1 The KUKA Robot

The whole hardware setup is illustrated in Fig. 2.1. A six-axis industrial robot, the KUKA Agilus KR 6 R900, is employed to play table tennis by mounting a racket at the end-effector. It is firmly fixated to the floor with a square steel plate, about 0.5 m away from the table. The KR C4 compact (Fig. 2.1④) can receive the motion commands from a host PC and control the robot via the data cable and motor cable. With the KUKA RobotSensorInterface (RSI) package, we can operate the robot at 250 Hz (4 ms cycle). All the motion commands are generated in Cartesian space on the host PC and subsequently transformed to joint space by the KR C4 controller. The world coordinate system is illustrated in Fig. 2.1 left, which is identical to the robot's.

When observing an incoming ball, we are going to find the desired position and the linear velocity of the racket at the hitting time and drive the robot to reach this target. The motion path is planned by the Reflexxes library (Kröger, 2011) that can control the pose and velocity of the end effector within 1 ms cycles. When given an arbitrary initial state of motion, kinematic motion constraints, and the desired target state, a new state of motion can be calculated for the current control cycle, as shown in Fig. 2.2.

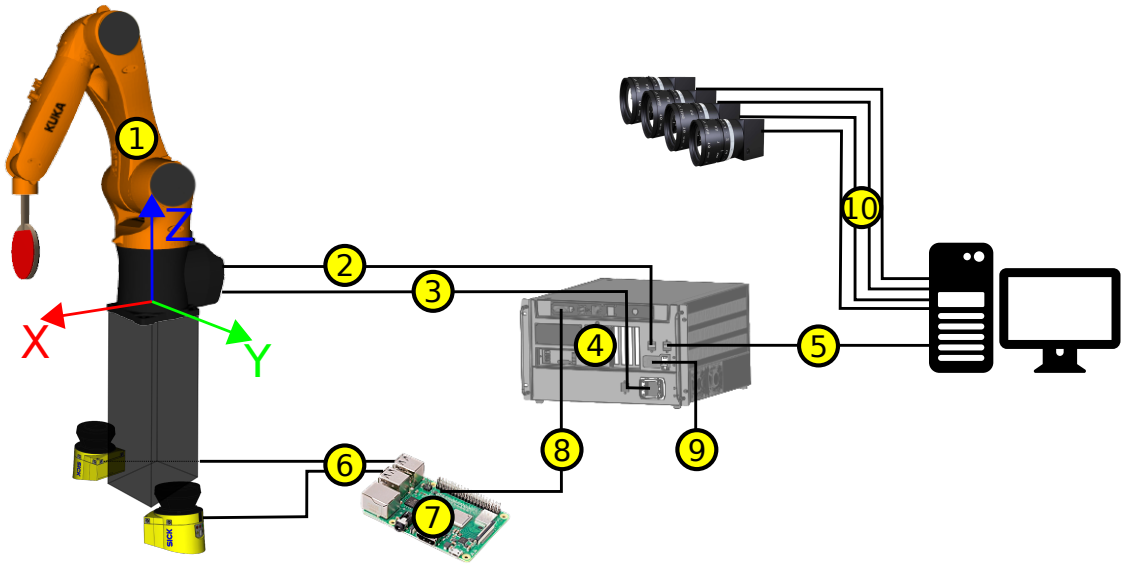


Figure 2.1: The hardware setup for the table tennis robot in the home position. ①: KUKA Agilus KR 6 R900 sixx manipulator. ②: data cable. ③: motor cable. ④: KR C4 compact controller. ⑤: motion commands transmission via ethernet interface from the host PC. ⑥: two SICK laser scanner connecting cables. ⑦: Raspberry Pi 3 controller. ⑧: external safety interface cable. ⑨: power supply connection. ⑩: USB3.0 cables for four cameras. The world coordinate system is identical to the robot's, which is illustrated on the left.

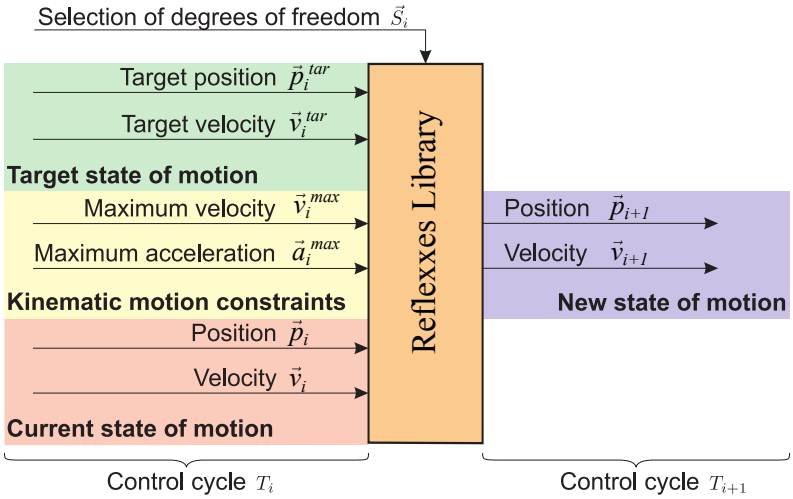


Figure 2.2: Input and output values of the Reflexxes library within one control cycle. The states of motion can be processed either in Cartesian space or in joint space. In this work we adopt the former one. Source: Kröger (2011).

To prevent the access and protect the area around the robot, we mount two SICK S300 Standard laser scanners on the floor as a security system in which each provides a 270° scanning angle (see Fig. 2.3). Thus, the security system can perform a full 360°scan for the protective fields. The response time is 80 ms and the protective field range is set to 2.5 m. To avoid the interference of a moving ball, we discard all small objects whose diameter is less than 0.05 m. The stationary table legs and the walls in the room are filtered out from the 3D point clouds when starting the scanners. For security reasons, an additional computer, Raspberry Pi 3 (Fig. 2.1⑦), runs independently of the host PC and is responsible for processing the captured point clouds. If one object gets close to the robot, the computer will send a STOP command to the KR C4 compact via the external safety interface cable (Fig. 2.1⑧). Once the robot is stopped, we have to manually reset the error in the Raspberry Pi 3 and then continue the robot program.

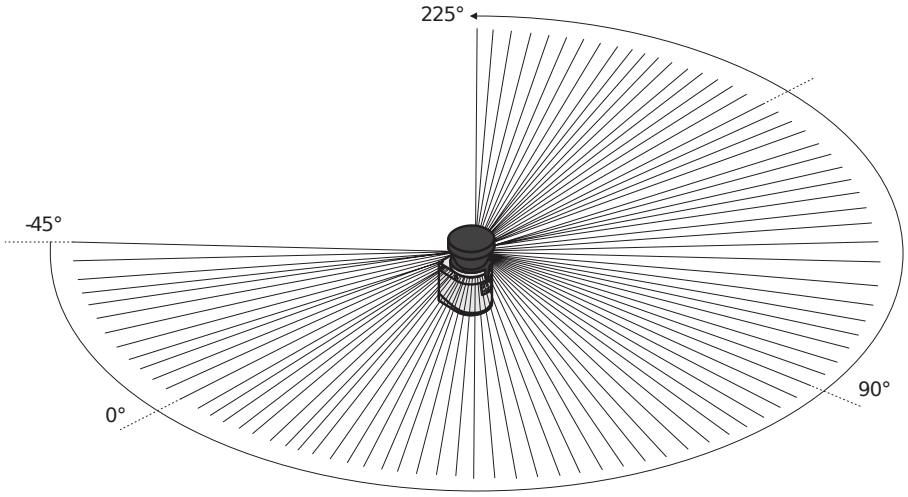


Figure 2.3: The protective field of a SICK S300 Standard laser scanner. The first beam of a scan begins at $-45°$ in relation to the rear side at 225°. Source: SICK (2020).

2.1.2 Cameras

Four FLIR Chameleon®3 color cameras, which are mounted in the corners of the ceiling, are used for object pose detection. These cameras are connected to the host PC via USB3.0 cables (Fig. 2.1⑩). One of them is configured as a primary camera which can trigger the other three replica cameras asynchronously via GPIO pins. The image resolution is 1280×1024 with a frame rate of 150 frames per second (FPS). Each camera has a 16MB buffer where the images can be stored temporarily. Here we store the latest four images into the buffer to be more robust against data loss. Then these images can be transmitted to the host PC for object detection. In this thesis, we split the four cameras into two pairs that can be used for different topics: ball position detection and racket pose detection.

2.2 Camera Calibration

A camera sensor can convert light photons to digital signals, which maps the 3D world to a 2D image in pixels. In order to investigate the camera geometry, a pinhole camera model has been developed to provide a geometric transformation between the realistic world and the image plane (Potmesil and Chakravarty (1982)). This will be discussed in Section 2.2.1. However, the monocular camera can only indicate an ambiguous 3D position when the 2D pixel is reprojected into 3D world. The usual way to handle it is to use stereo cameras, as described in Section 2.2.2

2.2.1 Monocular Camera Calibration

The pinhole model is illustrated in Fig. 2.4, where an orange ball with coordinates $P_w = (X_w, Y_w, Z_w)^T$ is projected into the image plane, forming the corresponding pixel $p = (u, v)$. The distance between the camera coordinate origin c and the image plane is defined as the physical focal length f. In order to describe the model with rectangular pixels, f is separated into two focal lengths f_x and f_y, which are introduced based on the width and height of one pixel (Kaehler and Bradski (2016)). The principal point (c_x, c_y) is usually near the image center. The projection of the ball P_w in world coordinates to the camera can be summarized as follows:

$$s \cdot p = A[R \mid t]P_w \tag{2.1}$$

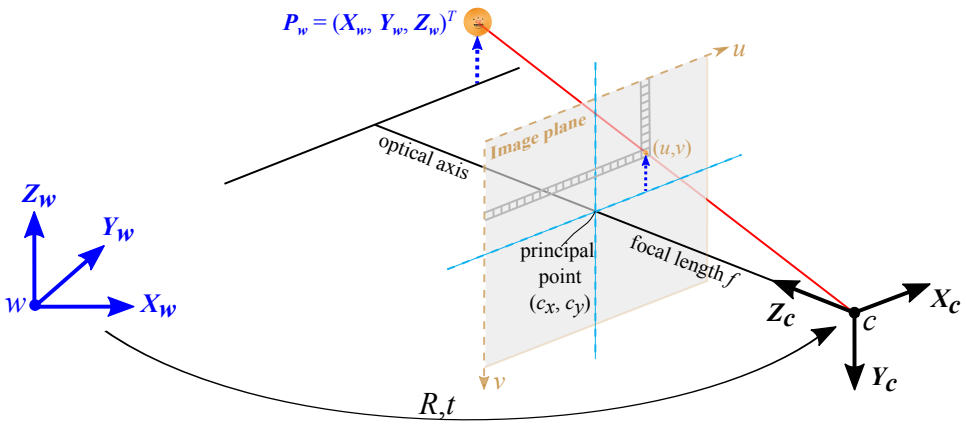


Figure 2.4: The pinhole camera model with an orange ping pong ball at position $(X_w, Y_w, Z_w)^T$ in the world coordinate system. $[R \mid t]$ is the transformation matrix from the world coordinate system w to the camera coordinate system c. The ball position P_c in camera coordinates is $(X_c, Y_c, Z_c)^T$.

where

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$[R|t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \tag{2.3}$$

$$p = (u, v, 1)^T, \quad P_w = (X_w, Y_w, Z_w, 1)^T. \tag{2.4}$$

A denotes the camera intrinsic matrix (Heikkila and Silvén (1997)). $[R \mid t]$ is the extrinsic matrix divided into a rotation R and a translation t between world coordinates w and camera coordinates c. p and P_w are converted to homogeneous vectors. s is the arbitrary scaling of the projective transformation and not part of the camera model. The ball position P_c in camera coordinates is $(X_c, Y_c, Z_c)^T$ which is the dot product of the transformation matrix $[R \mid t]$ and the P_w. If $Z_c \neq 0$, the Eq. 2.1 then can be written as the following:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x \cdot x' + c_x \\ f_y \cdot y' + c_y \end{bmatrix} \tag{2.5}$$

where x' and y' are equal to X_c/Z_c and Y_c/Z_c, respectively.

However, it is difficult to manufacture an ideal parabolic lens without any distortions. Therefore, Fryer and Brown (1986) introduced two models to correct the P_c. The radial distortions occur because of the lens shape, which is parameterized as the radial parameters $[k_1, k_2, k_3]$. The tangential distortions arise since the lens is not exactly parallel to the image plane, which includes two parameters p_1 and p_2. Therefore, the final model can be corrected as:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x \cdot x'' + c_x \\ f_y \cdot y'' + c_y \end{bmatrix} \tag{2.6}$$

where

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x' y' + p_2 \left(r^2 + 2x'^2\right) \\ y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1 \left(r^2 + 2y'^2\right) + 2p_2 x' y' \end{bmatrix} \tag{2.7}$$

with

$$r^2 = x'^2 + y'^2. \tag{2.8}$$

The aim of monocular camera calibration is to estimate the intrinsics $[f_x, f_y, c_x, c_y]$ and the distortions $[k_1, k_2, k_3, p_1, p_2]$. A chessboard (Zhang *et al.*, 1999) is usually chosen to compute the relative pose $[R \mid t]$ with the solvePnP method. The Levenberg-Marquardt (LM) optimization is used to minimize the reprojection error between the measured pixels and the projected ones in the image.

2.2.2 Stereo Camera Calibration

Once the intrinsics for monocular cameras are known, we can set up a pair of cameras to reconstruct the 3D position of an object in the world space. Fig. 2.5 shows the stereo camera geometry for a ball with coordinates P_w that is projected onto the left and right cameras as p_l and p_r, respectively. The camera baseline (the line jointing the camera coordinate origins) intersects each image plane at the epipoles. The epipolar line is the straight line passing through the p_l (p_r) and the epipole on the image plane. Each transformation matrix T is composed of a rotation R and a translation t. Then the point P_w relative to the left and right cameras can be computed by the following equations:

$$P_l = {}^{l}R_w \cdot P_w + {}^{l}t_w, \quad P_r = {}^{r}R_w \cdot P_w + {}^{r}t_w. \tag{2.9}$$

P_l and P_r are related by:

$$P_r = {}^{r}R_l \cdot P_l + {}^{r}t_l. \tag{2.10}$$

When performing the stereo camera calibration with a chessboard, the transformations $[{}^{l}R_w \mid {}^{l}t_w]$ and $[{}^{r}R_w \mid {}^{r}t_w]$ can be solved by the PnP method. Therefore, the output of the stereo calibration is the transformation $[{}^{r}R_l \mid {}^{r}t_l]$ between two cameras, which can be derived by:

$${}^{r}R_l = {}^{r}R_w \cdot {}^{l}R_w^T \tag{2.11}$$

$${}^{r}t_l = {}^{r}t_w - {}^{r}R_l \cdot {}^{l}t_w. \tag{2.12}$$

Similar to the monocular calibration, the LM optimization is also adopted to minimize the reprojection error for all the points from both cameras. The OpenCV library (Bradski, 2000) is utilized for both monocular and stereo calibrations.

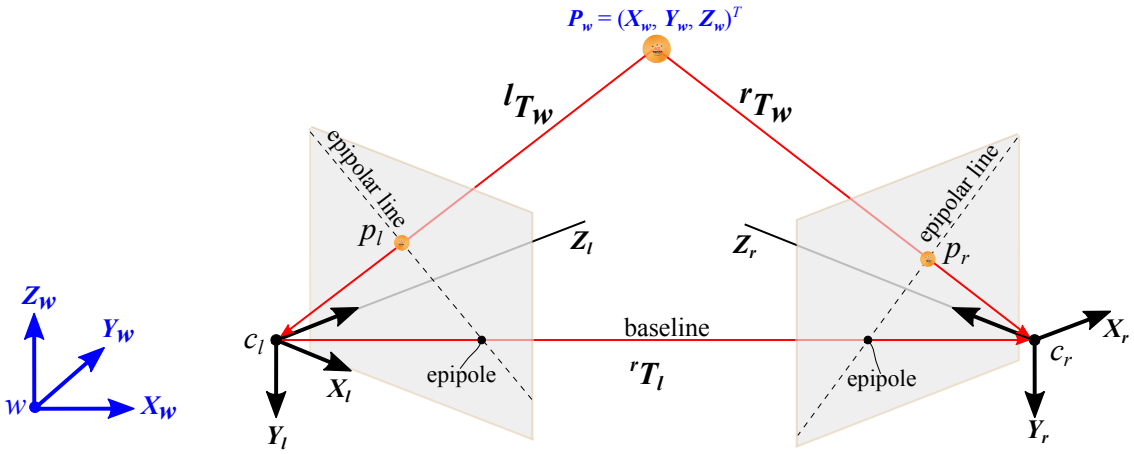


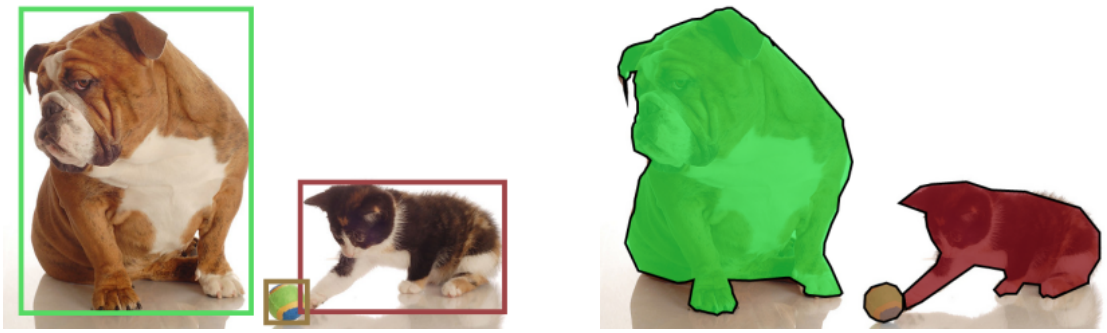
Figure 2.5: The stereo camera geometry with a ball. The transformation matrix ${}^{r}T_l$ from the left camera c_l to right camera c_r can be computed by the stereo calibration with a chessboard. With the calibrated stereo camera, we can reconstruct the 3D point in camera coordinates from pixels (p_l and p_r in this case) by triangulation.

2.3 Deep Learning for Visual Object Recognition

Recognizing an object and estimating its pose in the real world is a challenging task in robotics, such as object grasping (Tremblay *et al.*, 2018), industrial parts assembly (Kyrarini *et al.*, 2019), and bin picking (Kalra *et al.*, 2020). In this section, we will review recent vision-based work on 2D object recognition and 6D pose estimation.

2.3.1 2D Object Recognition

Object recognition is usually used to describe a collection of related computer vision topics in digital images and videos. Based on the different application fields, object recognition can be distinguished into two topics: object detection and image segmentation, as shown in Fig. 2.6.



(a) Object detection (b) Image segmentation

Figure 2.6: An annotated sample from the COCO dataset (Lin *et al.*, 2014), illustrating the difference between object detection and image segmentation. Three objects (*dog*, *ball*, and *cat*) are marked with different colors.

Object Detection

Object detection takes an image as input and outputs one or more bounding boxes with the corresponding class labels, which means it is a combination of object localization and image classification. Fig. 2.6a shows an example where three bounding boxes are tinted with different colors for multiple object detection. To accomplish the object detection, Girshick *et al.* (2014) developed an R-CNN (Regions with CNN features) neural network architecture. 2,000 region proposals were first generated using the selective search algorithm (Uijlings *et al.*, 2013) for the input image. For each region proposal, a convolutional neural network (CNN) was employed to produce a deep feature vector that was then fed into a support vector machine (SVM) to classify the presence of the object. The minimal bounding box with its label was finally decided by the greedy non-maximum suppression (NMS) technique. To accelerate the detection speed, they fed the

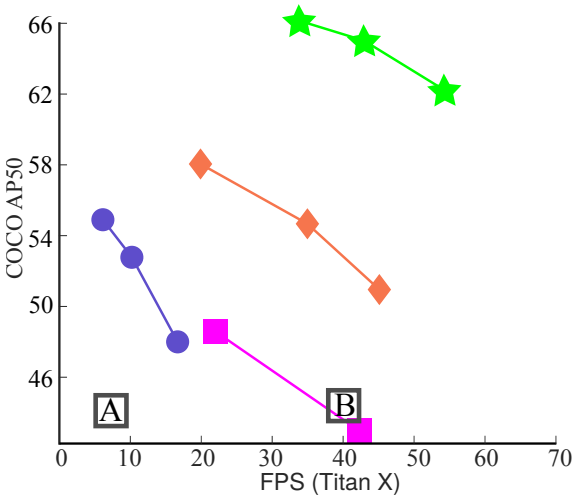
input image into a CNN module to first generate a deep feature map that was then divided into 2,000 regions. Each region proposal was reshaped into a fixed size using an RoI pooling layer. Instead of the SVM, a softmax layer was used to predict the class of the proposed region. This improved version was named Fast R-CNN (Girshick, 2015). However, the selective search algorithm was very slow and could generate some wrong candidate region proposals. Therefore, Ren *et al.* (2015) employed a separate region proposals network (RPN) to learn to produce the region proposals, which is called Faster R-CNN. An experiment showed it was $\sim$25x and $\sim$250x faster than Fast R-CNN and R-CNN, respectively.

Instead of splitting the process into two stages, one-stage detectors discard the region proposal stage and directly map the original image to the output over a dense sampling of possible locations. This makes detection potentially much faster but might decrease the performance slightly. YOLO (You Only Look Once, Redmon *et al.* (2016)) was the first unified detector model. The input image was divided into an $S \times S$ grid. Each grid cell predicted B bounding boxes, confidence score p for each box, and C class probabilities. Each bounding box consisted of 4 parameters: the center (x, y), the height h, and the width w. 24 convolutional layers followed by 2 fully connected layers were designed to process the input. The output was a $S \times S \times (B * 5 + C)$ tensor in which the minimal bounding boxes and labels were chosen by thresholding the confidence scores. The SSD (Single Shot MultiBox Detector) model (Liu *et al.*, 2016) improved YOLO in speed for high-accuracy detection. The core of SSD was to extract feature maps from different scaled convolutional layers and then predict offsets of predefined anchor boxes on each map. However, Lin *et al.* (2017) found that the extreme foreground-background class imbalance affected the detection accuracy tremendously. To address this problem, the novel Focal Loss function was introduced by reshaping the standard cross entropy-loss such that it down-weighted the loss assigned to well-classified examples. To evaluate the effectiveness of the Focal Loss, they designed and trained a simple dense detector called RetinaNet.

The comparison of the aforementioned methods on speed and AP50 performance is illustrated in Fig. 2.7 left. Instead YOLOv1, three YOLO variants, YOLOv2 (Redmon and Farhadi, 2017), YOLOv3 (Redmon and Farhadi, 2018), YOLOv4 (Bochkovskiy *et al.*, 2020)) are tested on the COCO dataset. VGG-16 (Simonyan and Zisserman, 2014) and ResNet-50 (He *et al.*, 2016) are used in SSD and RetinaNet as their backbones, respectively. The comparison table within different input sizes is shown in Fig. 2.7 right, which is sourced from Lin *et al.* (2017) and Bochkovskiy *et al.* (2020). All models are evaluated on an NVIDIA Titan X GPU.

Image Segmentation

Object detection builds a bounding box corresponding to each class in images, which is rather vague for object shapes. For example, an autonomous driving system needs to identify the road and building in order to understand the scene better. Therefore,



Method	Size	FPS	AP50
[A] Faster R-CNN	448	7.0	45.3
[B] YOLOv2	544	40.0	44.0
SSD	300	43.0	43.1
	512	22.0	48.5
RetinaNet	400	15.6	47.8
	600	10.2	53.2
	800	6.5	55.0
YOLOv3	320	45.5	51.5
	416	34.5	55.3
	608	19.6	57.9
YOLOv4	416	54.0	62.8
	512	43.0	64.9
	608	33.0	65.7

Figure 2.7: The comparison of different object detection models on an NVIDIA Titan X.

image segmentation is used to label every pixel in an image such that a collection of regions of pixels shares certain features (i.e., color and shape). Fig. 2.8 shows a pixel-level semantic segmentation example from the Cityscapes dataset (Cordts *et al.*, 2016). It includes 8 semantic groups, such as human, vehicle, construction, and sky. When considering higher-level object instance information, different classes in one group will be classified. For example, multiple *person* classes in the *human* group will be identified as different instances individually.

The first deep learning model for semantic image segmentation was proposed by Long *et al.* (2015). They designed a fully convolutional network (FCN) based on VGG-16 by

Figure 2.8: A pixel-level semantic segmentation example which shows an urban street scene in Tübingen, which is from the Cityscapes dataset (Cordts *et al.*, 2016). Overlayed colors encode different semantic groups. For example, a *human* group including the person and rider, is labeled as the red color in this case.

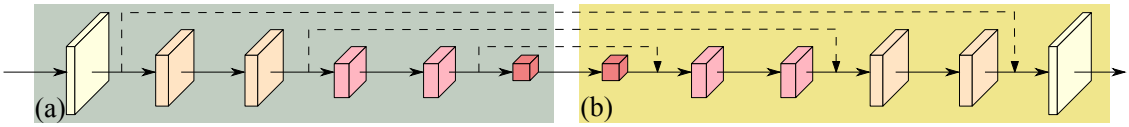


Figure 2.9: The encoder-decoder structure used in DeconvNet (Noh *et al.*, 2015) and SegNet (Badrinarayanan *et al.*, 2017). (a) An encoder to form the high-to-low convolutions in series. (b) A decoder to recover low-to-high representations in series.

replacing all dense layers with an upsampling layer. As a result, the output was a spatial segmentation map related to the designed classes. However, this model ignored the global context information, which caused failure for large or small objects. To address it, Noh *et al.* (2015) developed a deconvolution network (DeconvNet) in order to learn to construct the segmentation map. A deep feature vector was extracted by an encoder (here VGG-16) and then was fed into the deconvolution network. This encoder-decoder model (shown in Fig. 2.9) significantly outperformed FCN on the PASCAL VOC 2012 dataset (Everingham *et al.*, 2012) with an accuracy of 72.5%. Badrinarayanan *et al.* (2017) proposed a SegNet model by removing all the dense layers in DeconvNet. This improvement accelerated the segmentation speed and made the model easier to train because it was fully convolutional and involved fewer parameters. For object instance segmentation, He *et al.* (2017) extended the Faster R-CNN and presented a Mask R-CNN that detected objects efficiently in an image while simultaneously generating a high-quality segmentation mask for each instance. They also designed a joint loss function that combined the losses of the bounding box, class, and instance mask together.

The experiment shown their model outperformed other recent approaches, such as OpenPose (Cao *et al.*, 2019), DANet (Fu *et al.*, 2019), and CenterNet (Duan *et al.*, 2019) for human pose estimation, semantic segmentation, and object detection, respectively. Recently, Wang *et al.* (2020) developed a high-resolution network (HRNet) that maintained high-resolution representations during the whole process. Instead of connecting high-to-low (encoder) and low-to-high (decoder) convolutions in series, they used four scaled-resolution convolutions to generate the semantic masks in parallel, shown in Fig. 2.10.

2.3.2 6D Pose Estimation

Object recognition provides only 2D information in an image, which can not describe the real 3D world. Pose estimation is a task that can represent the 6D object pose, namely 3D translation and 3D rotation, in the camera coordinates or world coordinates. It is crucial for real-world applications, like autonomous navigation, robotic grasping, and augmented reality. Fig. 2.11 illustrates the pipeline for 6D object pose estimation given an input RGB/RGB-D image.

One of the first CNN models, BB8 (Rad and Lepetit, 2017), was developed to predict

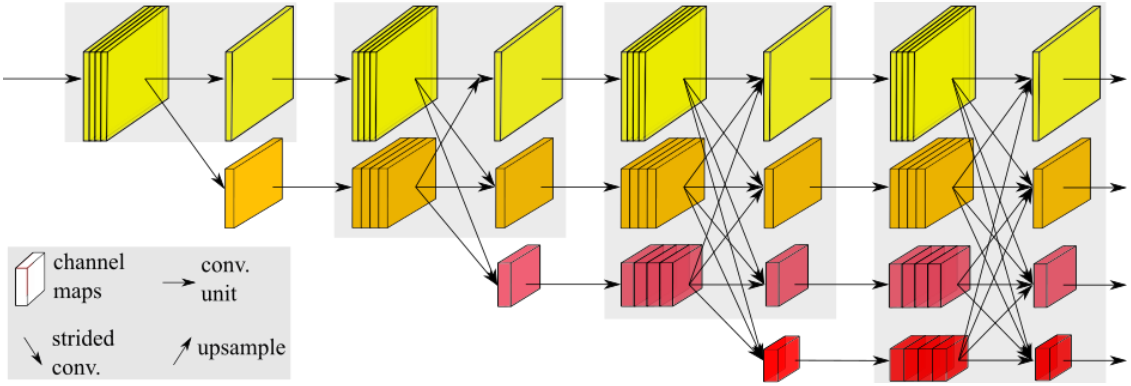


Figure 2.10: The main architecture used in the high-resolution network (HRNet). The outputs are merged depending on different tasks, such as human pose estimation, semantic segmentation, and object detection.

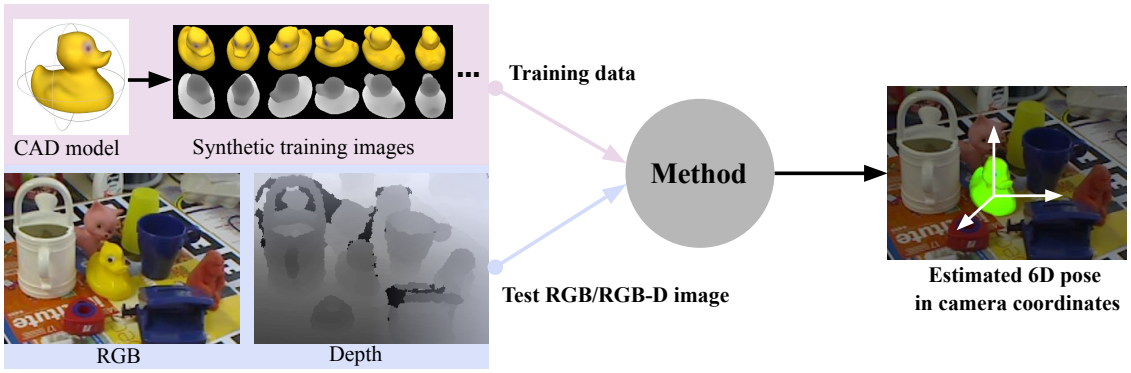


Figure 2.11: The pipeline for 6D object pose estimation given an input RGB/RGB-D image, which is from Hodan *et al.* (2018). Generally, the training dataset is created based on the 3D CAD models. The designed method can predict the 6D pose of any instance in the camera coordinates.

the 6D object pose. The 2D object centers were first identified based on the VGG net. Then the crops around the centers were fed into another VGG net in order to regress the 2D projections of 8 corners of the object 3D bounding boxes. With the prior information about the 3D sizes of the objects, the 6D pose can be obtained from the 2D-3D correspondences using a PnP algorithm. An additional step was required to refine the estimated pose. To simplify and accelerate the estimation steps, Tekin *et al.* (2018) proposed a single-shot approach that directly detected the 2D projections of the 3D bounding box vertices without any posterior refinement. Another holistic model, PoseCNN, was proposed in Xiang *et al.* (2017). It decoupled the pose estimation problem into three tasks: semantic labeling, 3D translation estimation, and 3D rotation regression. A new training loss function, ShapeMatch-Loss, was introduced for symmetric objects. However, these methods are prone to fail in the case of occluded and truncated objects. To address this problem, Peng *et al.* (2019) designed a pixel-wise voting network (PVNet) to produce a prediction for each pixel densely. Instead of using 8 corners of the 3D bounding box as keypoints, they robustly extracted K keypoints on the object surface with the farthest point sampling (FPS) algorithm. For each pixel, an object label and a set of direction vectors from this pixel to keypoints were predicted by PVNet. A RANSAC-based voting method was then used to generate hypotheses of 2D locations for each keypoint. The 6D pose could finally be driven by an uncertainty-driven PnP algorithm. Inspired from it, Song *et al.* (2020) introduced a HybridPose model, which utilized a hybrid intermediate representation to combine different geometric information, including keypoints, edge vectors, and symmetry correspondences. The experiment showed it outperformed PVNet on LineMOD (Hinterstoisser *et al.*, 2012) and Occlusion LineMOD (Brachmann *et al.*, 2014) datasets.

In addition, a low-cost RGB-D sensor can provide an extra depth channel for the RGB image, which is helpful for indoor applications. Wang *et al.* (2019a) presented an end-to-end model, DenseFusion, to estimate the object pose directly from RGB-D images. Based on an object segmentation model, two masks could be cropped from the RGB image and the corresponding depth map. Then they fed them into an encoder-decoder net and a PointNet (Qi *et al.*, 2017), respectively, to get a color feature embedding and a geometric feature embedding. After fusing them pixel-wisely, the final 6D pose could be generated by a pose predictor. A robotic grasping experiment showed 73% of the grasps were successfully completed. He *et al.* (2020) extended PVNet to a PVN3D model that was able to handle an input RGB-D image. Instead of extracting 2D keypoints, they designed a deep 3D keypoints Hough voting network to regress the keypoints offsets. PVN3D achieved state-of-the-art performance on the LineMOD and YCB-Video (Xiang *et al.*, 2017) datasets.

2.4 Deep Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning that involves sequential decision making by trial and error. In contrast to supervised learning and unsupervised learning, the RL algorithm helps an agent learn an optimal policy that is capable of deciding which action can maximize the cumulative rewards by interacting with an environment incrementally. Thus, RL does not need supervised labels or predefined categories. Deep RL incorporates deep learning into RL, which allows agents to make decisions directly from high-dimensional inputs (i.e., images) without any manual engineered features.

A significant advance in deep RL is the AlphaGo program (Silver *et al.*, 2016) developed by Google DeepMind. AlphaGo was the first computer program to defeat Go world champions in different global arenas and was arguably the greatest Go player in 2016. It used two deep neural networks: a policy network that outputs move probabilities and a value network that estimates the winning probability from the current position. AlphaGo learned the game by playing with professional human players, which was restricted by the limits of human knowledge. Therefore, DeepMind revealed AlphaGo Zero (Silver *et al.*, 2017) that can be trained by playing against itself without any human knowledge. Fig. 2.12 illustrates the DNN Architecture for AlphaGo Zero. The input is a $19 \times 19 \times 17$ image stack that consists of 8 binary feature planes for black stones in the previous 8 turns, further 8 planes for white stones, and 1 plane representing the color to play. 40 residual blocks (CNNs) are used to encode the inputs as deep features. Then they are passed into two separate heads for predicting the move probabilities and estimating the probability of the winning from the current position, simultaneously. Finally, a Monte Carlo tree search (MCTS) technique is used to select more substantial moves than the original move probabilities. AlphaGo Zero achieved superhuman performance and won $100 : 0$ against AlphaGo.

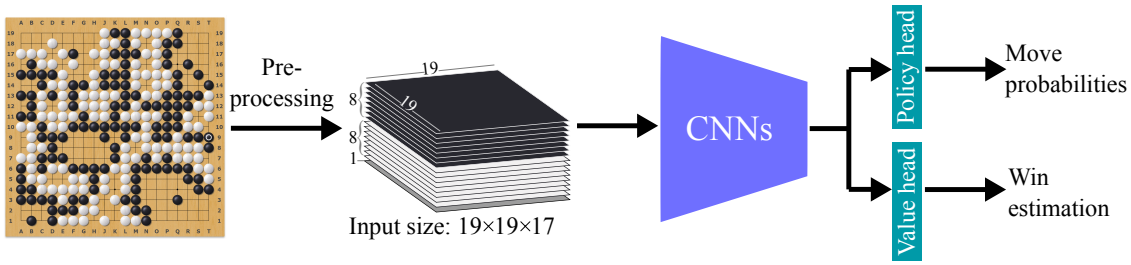


Figure 2.12: The AlphaGo Zero DNN Architecture for RL.

2.4.1 Markov Decision Processes

The typical RL problem can be formalized as a discrete-time control process where the agent and environment continually interact to achieve a goal (Sutton and Barto, 2018).

As illustrated in Fig. 2.13, at each time step t, the agent

(a) observes the environment's **state** $s_t \in \mathcal{S}$.

(b) selects an **action** $a_t \in \mathcal{A}$ based on a policy π.

(c) receives a scalar **reward** $r_t \in \mathcal{R}$ and transitions to **next state** s_{t+1}, according to the environment model.

In addition to the policy π, each state and action is passed into a value function $Q(s,a)$ to predict the expected amount of future rewards. π and $Q(s,a)$ are known as the **actor** and **critic**, which are what we want to learn in RL.

By assuming that the subsequent states and rewards only depend on the current state and action (Markov property), not the history, the RL problem is then reframed as Markov decision processes (MDPs). This assumption can be formalized as:

$$P(s_{t+1} \mid s_t, a_t) = P(s_{t+1} \mid s_0, a_0, \ldots, , s_t, a_t) \tag{2.13}$$

$$P(r_t \mid s_t, a_t) = P(r_t \mid s_0, a_0, \ldots, , s_t, a_t). \tag{2.14}$$

The MDPs and agent thereby give rise to a T-step trajectory:

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_{T-1}, a_{T-1}, r_{T-1}). \tag{2.15}$$

This trajectory will terminate when the goal is reached, or the limited number of steps is exceeded, which is called **one episode** in RL. The return R_t starting from time t is computed as the infinite-horizon discounted accumulated rewards:

$$R_t = r_t + \gamma^1 r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{2.16}$$

where γ is in the range of $(0,1]$. It is a discount factor to penalize future rewards. Then the value function in the state s for a state-action pair can be defined as:

$$Q^{\pi}(s,a) = \mathbb{E}_{a\sim\pi}[R_t \mid s_t = s, a_t = a]. \tag{2.17}$$

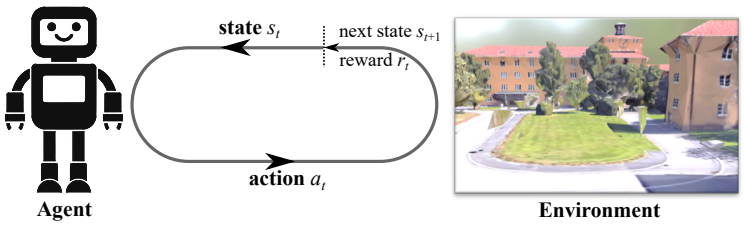


Figure 2.13: The agent–environment interaction in a Markov decision process.

The action a can be computed either from a stochastic policy $\pi(\cdot \mid s)$ or a deterministic one $\pi(s)$. Based on Bellman's expectation equations, the above equation can be rewritten into:

$$Q^{\pi}(s,a) = \mathbb{E}_{s' \sim \mathcal{P}} \left[r + \gamma \mathbb{E}_{a' \sim \pi} \left[Q^{\pi} \left(s', a' \right) \right] \mid s_t = s, a_t = a \right] \tag{2.18}$$

where s' and a' are the next state and action sampled from the environment's transitions and π, respectively. The idea behind Bellman's equations is decomposing the value function into the immediate reward plus the discounted values in the next step.

The optimal value function $Q^*(s,a)$ that gives the expected return if always acting according to the optimal policy, can be expressed by:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a). \tag{2.19}$$

Then the optimal action $a^*(s)$ can be obtained from it via:

$$a^*(s) = \arg\max_{a} Q^*(s,a) \tag{2.20}$$

which means the optimal policy will choose the action a that maximizes the expected return (or Q-value) in state s. We finally summarize the MDP as a 5-tuple:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma) \tag{2.21}$$

where $\mathcal{P}$ is the probability of transitioning the state from s to s' under action a. In this thesis, we will only discuss model-free RL algorithms that do not use the $\mathcal{P}$, since the environment model in most cases is unknown. The rest are states $\mathcal{S}$, actions $\mathcal{A}$, rewards $\mathcal{R}$, and discount factor γ.

2.4.2 Deep Q-Network

Q-learning (Watkins and Dayan, 1992) is a classic RL algorithm to learn the optimal policy π^* with the value function $Q^{\pi}(s,a)$. According to Eq. 2.18 and 2.19, the optimal value function $Q^*(s,a)$ can be rewritten as:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{P}} \left[r + \gamma \max_{a'} Q^* \left(s', a' \right) \right] \tag{2.22}$$

which means the maximum return starting from state s and action a, is the sum of the immediate reward r and the discounted maximum return from the next states s'. A basic idea behind RL is considering the Bellman equation as a value iteration update:

$$Q_{i+1}(s,a) = \mathbb{E}_{s' \sim \mathcal{P}} \left[r + \gamma \max_{a'} Q_i \left(s', a' \right) \right]. \tag{2.23}$$

Sutton and Barto (1998) have shown that this converges to the optimal value function, $Q_i \to Q^*$ as $i \to \infty$. The original Q-learning algorithm is depicted in Algorithm 1 where $Q(s,a)$ is typically approximated by a linear function or a non-linear function, the epsilon-greedy exploration strategy follows the greedy strategy with probability $1-\varepsilon$, and selects a random action with probability ε from all possible desecrate actions. Q-learning is an *off-policy* RL algorithm, since the current policy is different from the one used to generate the next action. This gives Q-learning the ability to reuse the past samples.

Algorithm 1: Q-learning for updating Q-value

Input: initial $Q(s,a)$, learning tate α, and discount factor γ
Output: Optimal value function $Q^*(s,a)$

```
for each episode do
    Initilize state s
    for each step of episode do
        if s is not terminal then
            Choose action a ← arg max_a Q(s,a), using epsilon-greedy strategy
            Apply action a, observe the reward r and new state s'
            Q(s,a) ← Q(s,a) + α[r + γ max_a' Q(s',a') − Q(s,a)]
            s ← s'
        else
            Q(s,a) ← Q(s,a)
        end
    end
end
```

A neural network was first used with Q-learning in Mnih *et al.* (2013). They proposed a deep Q-network (DQN) with weights ϕ to estimate $Q(s,a)$, i.e. $Q(s,a;\phi) \approx Q^*(s,a)$, and trained it by minimizing the following loss function at each iteration step i:

$$\mathcal{L}_i(\phi_i) = \mathbb{E}_{s,a,r,s',\sim\rho(\cdot)}\left[\left(y_i - Q(s,a;\phi_i)\right)^2\right] \tag{2.24}$$

where

$$y_i = r + \gamma \max_{a'} Q\left(s',a';\phi_{i-1}\right) \tag{2.25}$$

is known as the temporal difference (TD) target and $y_i - Q$ as the TD error. $\rho(\cdot)$ donates the behavior distribution over transitions s,a,r,s'.

However, the non-linear function approximator will make RL unstable and prone to diverging during training. DQN addressed this problem by using two techniques:

- **Experience Replay**: At each step of the episode, the transitions s,a,r,s' are stored into a *replay buffer*. Then instead of using the newest transitions to update Q-

network, DQN used a batch of transitions that were randomly sampled from the replay buffer. This technique can reuse each transition and avoid the strong correlations between consecutive samples.

- **Target Network**: The Q-network is periodically copied and kept frozen as the optimization target $Q(s', a'; \phi_{i-1})$. This technique removes short-term oscillations and makes training more stable.

The full algorithm is presented in Algorithm 2. An experiment has shown that DQN achieved better performance than an expert human player on 3 computer games in Atari 2600, which provides various RL agents with RGB images as inputs. Fig. 2.14a to 2.14c show these 3 games including Pong, Breakout, and Enduro. DQN achieved close to human performance on Beam Rider (Fig. 2.14d). The average total rewards for various games are included in Table 2.1, where **Random** means all the actions are sampled randomly from a Gaussian distribution, and **DQN Best** is the best performance in all episodes.

Algorithm 2: Deep Q-network with Experience Replay

Input: random weights ϕ, discount factor γ, replay buffer $\mathcal{D}$
Output: Optimal value function $Q^*(s, a; \phi)$

for *episode=0, M* **do**
  Initilize state s_0
  for $t = 0$, T **do**
    Choose action $a_t \leftarrow \arg\max_a Q(s_t, a; \phi)$, using epsilon-greedy strategy
    Apply action a_t, observe the reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in buffer $\mathcal{D}$
    Sample a random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
    if $t < T - 1$ **then**
      $y_j = r_j + \gamma \max_{a'} \hat{Q}\left(s_{j+1}, a'; \phi\right)$
    else
      $y_j = r_j$
    end
    Update ϕ by minimizing the loss function:
      $$\mathcal{L}_j(\phi) = \mathbb{E}\left[\left(y_j - Q\left(s_j, a_j; \phi\right)\right)^2\right]$$
    Reset target network $\hat{Q} \leftarrow Q$ periodically
  end
end

(a) Pong (b) Breakout (c) Enduro (d) Beam Rider

Figure 2.14: Screen shots from four Atari 2600 Games. DQN surpassed expert human players on (a), (b), and (c). It achieved close to human performance on (d).

Table 2.1: The average reward comparison between DQN and human performance.

	Pong	Breakout	Enduro	Beam Rider
Random	-20.4	1.2	0	354
Human	-3	31	368	**7456**
DQN	20	168	470	4092
DQN Best	**21**	**225**	**661**	5184

2.4.3 Policy Gradient Theorem

By computing the Q-value (critic) for each discrete action, DQN can find the optimal action with the best value, which belongs to the **value-based** RL methods. In contrast, **policy-based** methods directly build a policy function (actor), mapping states to actions, and have better convergence properties. They can be used for environments with either discrete or continuous actions. There are two common kinds of policy gradient methods (Sutton and Barto, 2018) to compute the actions:

Stochastic Policy Gradient

A stochastic policy can be denoted by π_θ:

$$a_t \sim \pi_\theta \left(\cdot \mid s_t \right) \tag{2.26}$$

which is the probability distribution for action a_t to take from state s_t.

For a T-step trajectory τ in Eq. 2.15, the first state s_0 is randomly sampled from a distribution ρ_0:

$$s_0 \sim \rho_0(\cdot). \tag{2.27}$$

The probability for this trajectory is:

$$P(\tau \mid \theta) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1} \mid s_t, a_t)\, \pi_\theta(a_t \mid s_t). \tag{2.28}$$

The expected return in τ is then:

$$J(\theta) = \int_\tau P(\tau \mid \theta) R(\tau) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta}[R(\tau)] \tag{2.29}$$

where

$$R(\tau) = \sum_{t=0}^{T-1} \gamma^t r_t. \tag{2.30}$$

According to OpenAI (2018), the policy gradient $\nabla_\theta J(\pi_\theta)$ can be derived by the following steps:

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] \\ &= \nabla_\theta \int_\tau P(\tau \mid \theta) R(\tau) \\ &= \int_\tau \nabla_\theta P(\tau \mid \theta) R(\tau) \\ &= \int_\tau P(\tau \mid \theta) \nabla_\theta \log P(\tau \mid \theta) R(\tau) \\ &= \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [\nabla_\theta \log P(\tau \mid \theta) R(\tau)]. \end{aligned} \tag{2.31}$$

Since

$$\begin{aligned} \nabla_\theta \log P(\tau \mid \theta) &= \nabla_\theta \log p_0(s_0) + \sum_{t=0}^{T-1} (\nabla_\theta \log P(s_{t+1} \mid s_t, a_t) + \nabla_\theta \log \pi_\theta(a_t \mid s_t)) \\ &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t), \end{aligned} \tag{2.32}$$

thus the stochastic policy gradient (SPG) is finalized as:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \Phi_t \right] \tag{2.33}$$

where Φ_t is equal to $R(\tau)$, or can be one of the following variants:

$$\begin{aligned}
&\Phi_t = \sum_{t'=t}^{T-1} R\left(s_{t'}, a_{t'}, s_{t'+1}\right) && \text{or} \\
&\Phi_t = \sum_{t'=t}^{T} R\left(s_{t'}, a_{t'}, s_{t'+1}\right) - b\left(s_t\right) \quad \text{(with a baseline } b\text{)} && \text{or} \\
&\Phi_t = V_\phi\left(s_t\right) \quad \text{(}V\text{-value function with weights } \phi\text{)} && \text{or} \\
&\Phi_t = Q_\phi\left(s_t, a_t\right) \quad \text{(}Q\text{-value function)} && \text{or} \\
&\Phi_t = A_\phi\left(s_t, a_t\right) = Q_\phi\left(s_t, a_t\right) - V_\phi\left(s_t\right) \quad \text{(Advantage function).}
\end{aligned} \tag{2.34}$$

If a value function, such as $V_\phi\left(s_t\right), Q_\phi\left(s_t, a_t\right), A_\phi\left(s_t, a_t\right)$, is used, the policy gradient method is then represented as an **actor-critic** model.

Deterministic Policy Gradient

A deterministic action can be calculated by:

$$a_t = \pi_\theta\left(s_t\right). \tag{2.35}$$

Supposing the expectation from state s is approximated to a Q-value, $J(\theta)$ over a trajectory τ can be written as:

$$J(\theta) = \mathop{\mathbb{E}}_{s\sim\mathcal{S}}\left[Q_\phi\left(s, \pi_\theta(s)\right)\right], \tag{2.36}$$

because each action is deterministic and thereby the action probability is equal to 1, according to Silver *et al.* (2014). The deterministic policy gradient (DPG) can be given as:

$$\nabla_\theta J\left(\theta\right) = \mathop{\mathbb{E}}_{s\sim\mathcal{S}}\left[\nabla_\theta Q_\phi(s,a)\big|_{a=\pi_\theta(s)}\right] \tag{2.37}$$

or

$$\nabla_\theta J\left(\theta\right) = \mathop{\mathbb{E}}_{s\sim\mathcal{S}}\left[\nabla_\theta \pi_\theta(s) \cdot \nabla_a Q_\phi(s,a)\big|_{a=\pi_\theta(s)}\right] \tag{2.38}$$

by expanding the gradient of the Q-value.

According to Eq. 2.20, the optimal action $a^*(s)$ can be found by computing the Q-value for each possible action, individually, for the discrete action spaces. However, if the actions are in continuous spaces, the problem will become highly non-trivial. It is unacceptable to perform infinite actions in the environment. Therefore, Lillicrap *et al.* (2015) proposed a Deep Deterministic Policy Gradient (DDPG) algorithm to concurrently learn a Q function and a policy in continuous action spaces. Instead of calculating the maxima over actions in the target (Eq. 2.25), DDPG approximates it by using a target policy network $Q_{\phi targ}$:

$$\max_{a'} Q\left(s', a'; \phi\right) \approx Q_{\phi targ}(s', \pi_{\theta targ}(s')). \tag{2.39}$$

Then Eq. 2.25 becomes:

$$y_i = r + \gamma(1-d)Q_{\phi targ}(s', \pi_{\theta targ}(s')) \tag{2.40}$$

where $d = 1$ when s' is a terminal state, otherwise $d = 0$. The loss function for updating the Q-value is defined as:

$$\mathcal{L}(\phi, \mathcal{D}) = \mathbb{E}_{s,a,r,s',d,\sim\mathcal{D}}\left[\left(Q_\phi(s,a) - y_i\right)^2\right] \tag{2.41}$$

where $\mathcal{D}$ is the experience buffer. The whole process for DDPG is detailed in Algorithm 3.

Algorithm 3: DDPG algorith with Experience Replay

Input: random weights θ, ϕ, discount factor γ, replay buffer $\mathcal{D}$, target network weights $\theta_{\text{targ}} \leftarrow \theta, \phi_{\text{targ}} \leftarrow \phi$, factor ρ
Output: Optimal action function $\pi^*_\theta(s)$

for *episode=0, M* **do**
  for $t = 0$, T **do**
    Choose action $a = \text{clip}\left(\pi_\theta(s) + \varepsilon, a_{\text{low}}, a_{\text{high}}\right)$, where $\varepsilon \sim \mathcal{N}$
    Apply action a, observe the reward r and new state s'
    Store transition (s, a, r, s', d) in buffer $\mathcal{D}$
    If $d = 1$, reset the environment
    if *it is time to update* **then**
      Sample a random minibatch of transitions $(s_j, a_j, r_j, s_{j+1}, d_j)$ from $\mathcal{D}$
      Compute targets:
        $y_j = r_j + \gamma(1-d_j)Q_{\phi targ}(s_{j+1}, \pi_{\theta targ}(s_{j+1}))$
      Update ϕ by minimizing the Q loss function:
        $\mathcal{L}_j(\phi) = \mathbb{E}\left[\left(Q_\phi\left(s_j, a_j\right) - y_j\right)^2\right]$
      Update π by minimizing the policy loss:
        $\mathcal{L}_j(\theta) = -\mathbb{E}\left[Q_\phi\left(s, \pi_\theta(s)\right)\right]$
      Update target networks:
        $\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$
        $\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta$
    end
  end
end

However, DDPG sometimes tends to overestimate the Q-value because it exploits the errors in the Q-function. Twin Delayed DDPG (TD3) (Fujimoto *et al.*, 2018), a variant of DDPG, addresses this problem by using two Q-functions and updating the policy with delay. It is used as the backbone for robot stroke learning in Chapter 4.

Chapter 3

Multi-camera Calibration and 3D Ball Position Detection

In this chapter, we develop a calibration approach for multiple stationary cameras in Section 3.1. The evaluation dataset is generated by using the KUKA Agilus R900 sixx robot. To extract the ball pixel positions from images, we propose two approaches: the traditional image processing that makes use of the color thresholding technique and deep learning that segments the ball contours based on a manually labeled dataset in Section 3.2. Finally, we briefly introduce the ball trajectory prediction work by Tebbe *et al.* (2018, 2020) in Section 3.3.

Large parts of this work have been pre-published in Tebbe *et al.* (2018).

3.1 Calibration Approaches

To achieve high accuracy for the object pose detection, stepwise calibration is needed, which includes the following three steps:

Monocular camera calibration

The goal is to estimate the intrinsics:

$$[f_x, f_y, c_x, c_y] \tag{3.1}$$

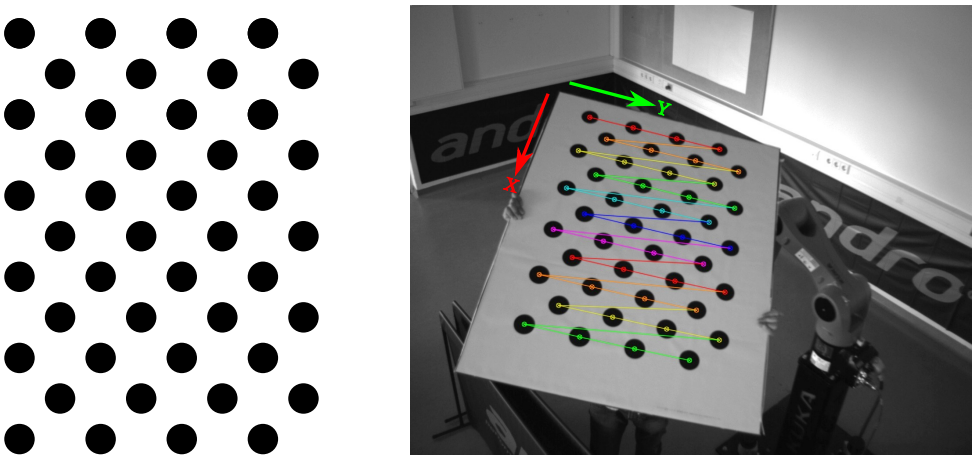
and the distortion vector:

$$[k_1, k_2, k_3, p_1, p_2] \tag{3.2}$$

for each monocular camera, according to Section 2.2.1.

In general, asymmetric circle-grids give better performance than the classic black-white chessboard, both in terms of the quality of final results as well as the stability of those results between multiple runs (Kaehler and Bradski, 2016). For these reasons, an asymmetrical circle pattern (Fig.3.1a) is printed on A0 paper and is glued to a rigid board for camera calibrations. It has a size of 4×11, and the spacing between two circles is 8.0 cm. As shown in Fig. 3.1b, a hand-designed coordinate system is located on the board

where the top-left circle center is the origin, and the X and Y are along the directions of the pattern rows and columns. The Z value for each circle is equal to zero. Given each circle's position in both image coordinates and the 3D world, we can then compute the intrinsics with Eq. 2.6.



(a) Circle grid pattern (b) Monocular calibration example

Figure 3.1: Monocular camera calibration. (a) an asymmetrical circle pattern with the size of 4×11. It is printed on A0 paper and is glued to a rigid board. The spacing between two circles is 8.0 cm. (b) a calibration example where each circle center is extracted and displayed in colors by OpenCV (Bradski, 2000). The top-left and bottom-right centers are the origin and the end of this pattern, respectively.

Stereo Camera Calibration

Since there are four (one primary and three replica) cameras at each ceiling corner (see Section 2.1.2), we need to compute three sets of transformations between primary and replica cameras. Fig. 3.2 shows two snapshots from the left (primary) and right (replica) cameras. After collecting a number of snapshots, we can get the relative rotation matrix ${}^{r}R_{l}$ and translation vector ${}^{r}t_{l}$ by Eq. 2.11 and 2.12.

Multi-camera Calibration

To post-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 (Fig. 3.3). Then, we employ a modified sparse bundle adjustment approach (Triggs *et al.*, 1999), which minimizes the following error function:

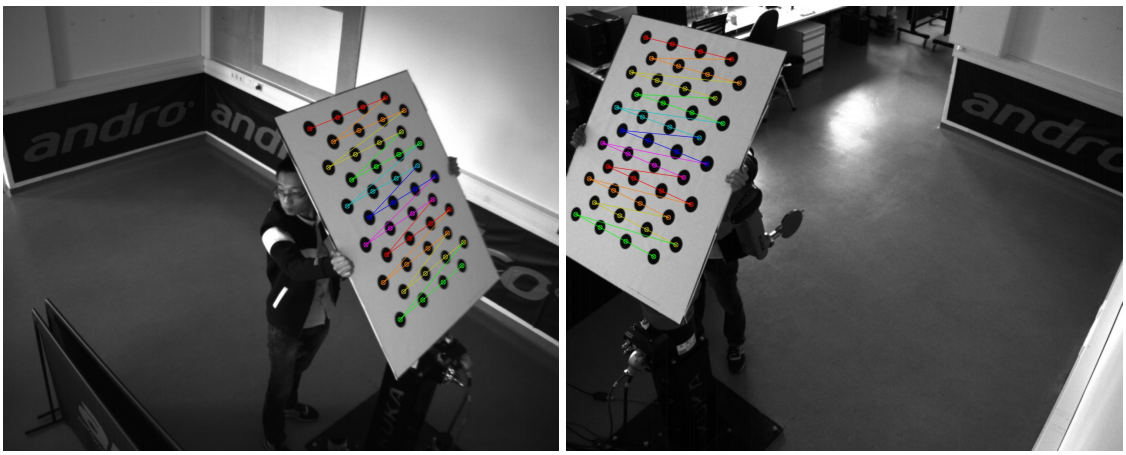


Figure 3.2: Stereo calibration example with the goal to compute the transformation matrix for a pair of monocular cameras. Left image is from the primary camera, right one from a replica camera.

$$E(P,X) = \sum_{i=1}^{m} \sum_{j=1}^{n} \|P_j X_i - x_{ij}\|^2 + \lambda \sum_{i=1}^{m} |\|X_{i+1} - X_i\| - D|^2 \tag{3.3}$$

where the first term is normally used in the standard multi-view calibration algorithm. 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 (Agarwal *et al.*, 2012).

The system's accuracy is evaluated by mounting a table tennis ball on the end-effector of the robot and comparing it 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 point sets. An example is illustrated in Fig. 3.4.

The two systems operate in different coordinate systems, and a coordinate transformation is estimated using the two 3D point sets according to Arun *et al.* (1987). The camera calibration errors are shown in Table 3.1, which includes three tests using both two and four cameras. Adopting our proposed multi-camera calibration we achieve an error of 2.5 mm for four cameras, 3.2 mm for two cameras. To efficiently use the cameras, we divide them into two stereo-camera pairs for the ball and racket detection, respectively.

Table 3.1: Calibration errors comparison in mm.

	Stereo Calibration	Multi-View Calibration	**Ours**
Two Cameras	11.0	4.9	**3.2**
Four Cameras	15.0	4.7	**2.5**

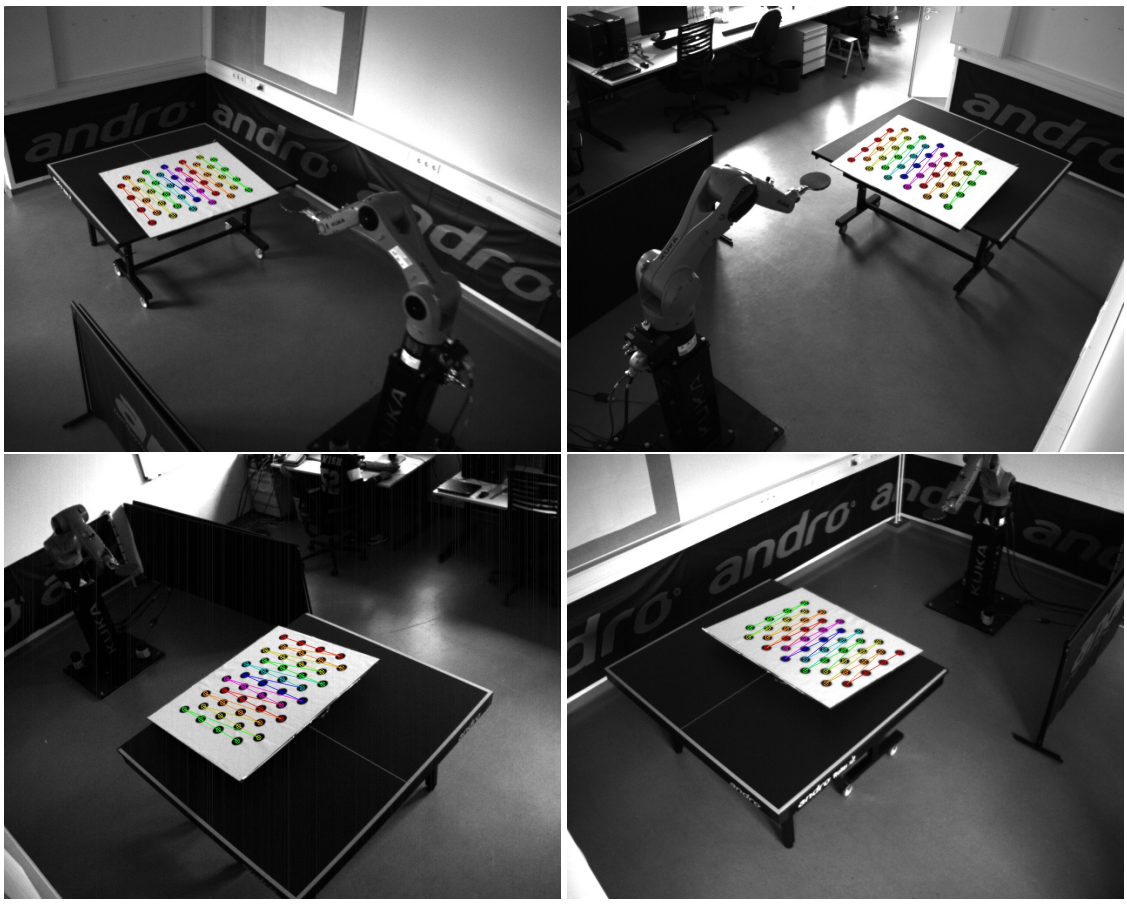


Figure 3.3: Multi-camera calibration example with the goal to refine the transformation matrices. Four images are simultaneously recorded as one set from every camera. 10 sets are finally utilized in the multi-camera calibration part.

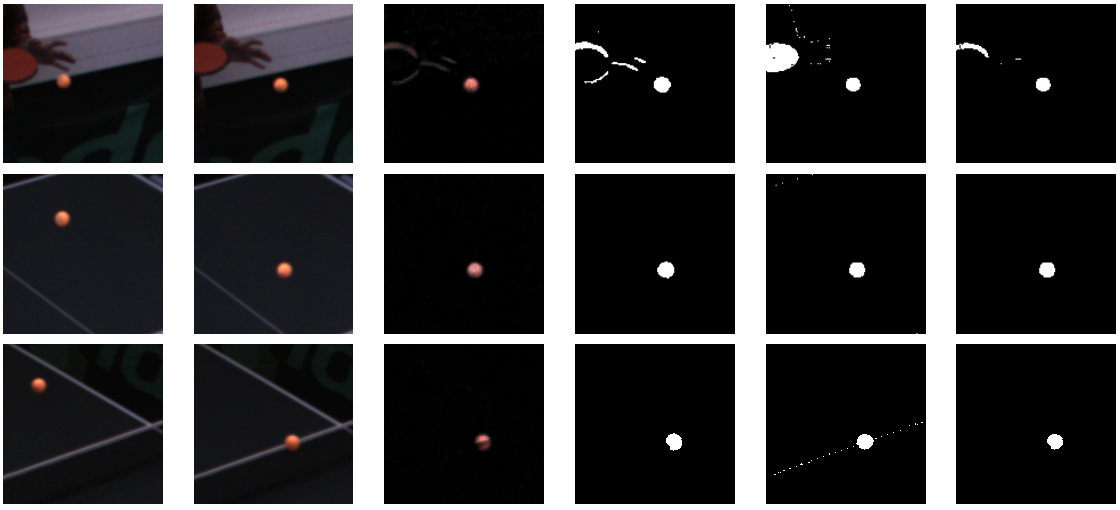
Figure 3.4: An evaluation sample recorded using our KUKA robot. A ping pong ball is attached at the end effector. The ball's positions in image coordinate systems are manually labeled. The 3D ball positions can be read from the robot controller.

3.2 Ball 2D Pixel Position Extraction

It is difficult to estimate the ball position directly from one monocular camera because of its tiny size and texture-less appearance. Therefore, we decouple it into two steps: ball 2D pixel position extraction and 3D position triangulation from two cameras. In order to efficiently detect the ball from images, we first introduce the traditional image processing approaches by fusing the ball's features. Then an alternative approach, deep learning, is used to segment the ball contours based on the deep features.

3.2.1 Traditional Image Processing

The method adopted in this subsection is fusing multiple features including motion, color, area and shape, which are used in Zhang *et al.* (2009); Li *et al.* (2012). 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 3.5 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 the racket and the player's hand. The middle row details a state on the table and in the last row the ball is crossing the net.



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

Figure 3.5: 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).

We want to avoid the crescent shaped ball shown in Zhang *et al.* (2010) 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 3.5(a)-(c). The binary shown in Figure 3.5(d) uses thresholding according to the following equation:

$$Binary_{\mathrm{n}}(u,v) = \begin{cases} 255 & \text{if } L \leqslant HSV_{\mathrm{n}}(u,v) - HSV_{\mathrm{n\text{-}7}}(u,v) \leqslant U \\ 0 & otherwise \end{cases}. \tag{3.4}$$

$HSV_n(u,v)$ 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 3.5(e). By means of computing the bitwise conjunction of (d) and (e) in Figure 3.5(f), we can extract the orange moving objects including balls and possibly skin regions, the moving racket 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:

$$\begin{cases} 10px \leqslant & Area & \leqslant 800px \\ 0.5 \leqslant & AreaExtent & \leqslant 1 \\ 1/1.4 \leqslant & AspectRatio & \leqslant 1.4 \end{cases} \tag{3.5}$$

where *Area* is the contour area extracted from the Figure 3.5(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) with the size of 160×160, will be computed around the ball's center. Then we can track the next ball in this ROI to accelerate the detection speed.

3.2.2 Deep Learning

Although the traditional approach shows some success, it is very tedious to find the appropriate thresholds. And the orange KUKA robot could influence the detected contours, as shown in Fig. 3.6. In recent years, Deep Learning has shown remarkable results in various applications. Therefore, instead of using color thresholding in Fig. 3.5(d) and (e), we try to segment the ball contours based on a CNN model. The following three steps are introduced for collecting datasets, creating CNNs, and training.

Dataset

In order to achieve some diversity in color values, 4036 consecutive images are collected from the two cameras under different lightness conditions, 80% of which is for training

Figure 3.6: An example when the ball is going to hit the racket. Left: the cropped RGB image. Middle: the final contour by the traditional approach. Right: the result segmented by deep learning.

and the rest for testing. Additionally, a testing dataset including 1114 consecutive images is recorded for comparing different algorithms. A free online tool, called Computer Vision Annotation Tool (CVAT, Manovich (2018)), is adopted to mask the round contours of the balls semi-automatically based on a pre-trained DNN model (here we used YOLOv3). Each label is a binary image where 0 means background and 1 means ball. Then we randomly crop each image and its mask into a small size of 160×160 around the ball's position, which actually generates a dataset of ROIs. If there is no ball in an image, a random crop is created.

Model

Because of this simple and binary segmentation problem, it is possible to extract the ball contours only using a shallow network. The developed model is detailed in Table 3.2, including one hidden layer *conv2d_1* and one output layer *conv2d_2*. Both of them are 2D convolution layers with $softmax$ activation and a kernel size of 11×11. The output width w and height h are varied depending on different input sizes. Here it is 160×160. The total number of trainable parameters is $728 + 486 = 1,214$.

Table 3.2: CNN model summary for ball segmentation.

Layer (type)	Output Shape	# Parameters
input (InputLayer)	[(None, w, h, 3)]	0
conv2d_1 (Conv2D)	(None, w, h, 2)	728
conv2d_2 (Conv2D)	(None, w, h, 2)	486

Trainig

The binary cross-entropy loss function provided by TensorFlow (Abadi *et al.*, 2015) is used for training together with the Adam (Kingma and Ba, 2014) optimizer. The inputs are normalized from $[0,255]$ to $[0,1]$. The batch size and epochs are set to 2 and 100. Other hyperparameters are the default values. The training process is illustrated in Fig. 3.9a.

Diffienrent model study

To investigate the number and architecture of layers, we develop three other models for comparison:

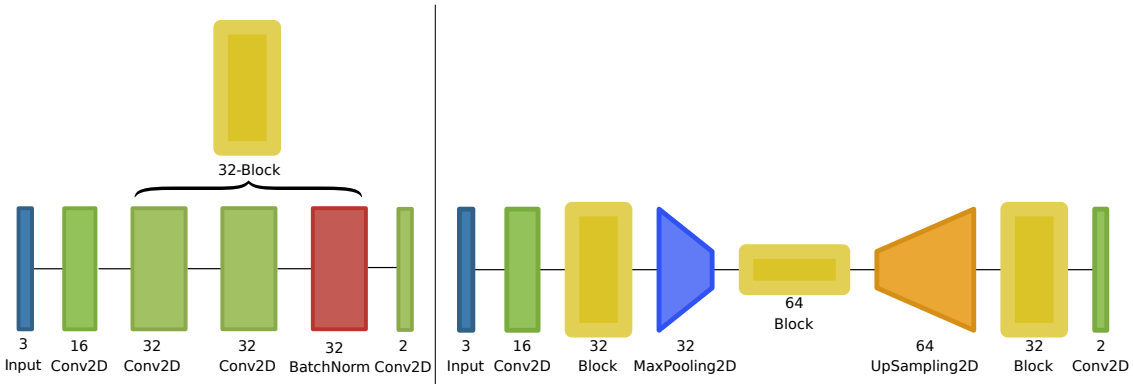
- Five-layer model: The inputs are successively fed into a convolutional layer with 16 filters, one block including two convolutional layers and a batch normalization layer, and the output layer, as shown in Fig. 3.7 left.
- Encoder-Decoder: It is built based on a block system. After the first block, one MaxPooling layer is performed to downscale the feature resolution, followed by the second block. An UpSampling layer gets the resolution back up to the original size. Finally, the third block and the output layer stay unchanged as same as the ones in the five-layer model. The schematic is illustrated in Fig. 3.7 right.
- SegNet: Four blocks are used in both encoder and decoder. An additional residual is held in each block right before MaxPolling, which is then added back after the corresponding UpSampling step. This operation can preserve the details lost in the down- and upsampling process. The whole structure is shown in Fig. 3.8.

All the hidden layers in these three models are activated by the *ReLU* function. A smaller kernel size of 3×3 is applied in each hidden convolutional layer. Other hyperparameters are the same as in the two-layer model. The training loss and test loss are plotted in Fig. 3.9. Overfitting occurs in Fig. 3.9c and 3.9d.

In addition, all models are evaluated on the testing dataset for the inference speed. The results are shown in Table 3.3. Considering the speed-accuracy trade-off, we decide to segment the ball using the two-layer model since it is the fastest one with acceptable accuracy.

Table 3.3: Comparison of each model with the parameter numbers, the lowest test loss, and the inference speed per image. The input size is 160×160.

Model	Total Parameters	Test Loss	Speed (ms)
Two-layer	1,214	0.0034	1.67
Five-layer	14,530	0.0029	2.24
Encoder-Decoder	98,050	0.0020	2.77
SegNet	2,942,082	0.0023	4.48



(a) Five layers model (b) Encoder-Decoder

Figure 3.7: Schematic visualization of the five layers model (left) and the Encoder-Decoder model (right). The five layers model consists of a convolutional layer with 16 filters, one block including two convolutional layers and a batch normalization layer, and an output layer. The Encoder-Decoder model is built based on the three blocks.

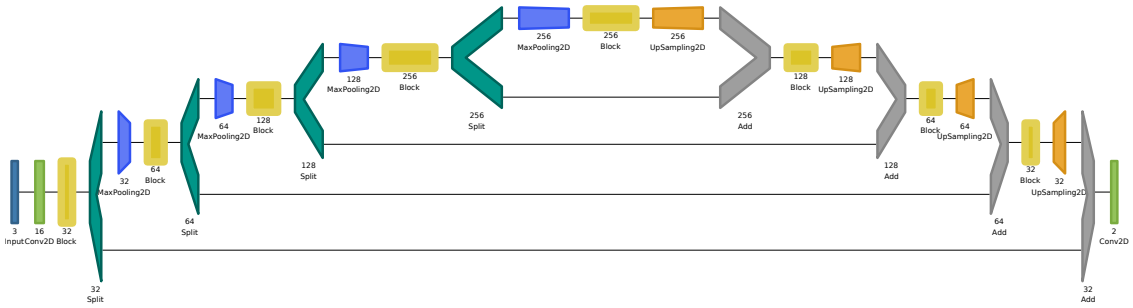
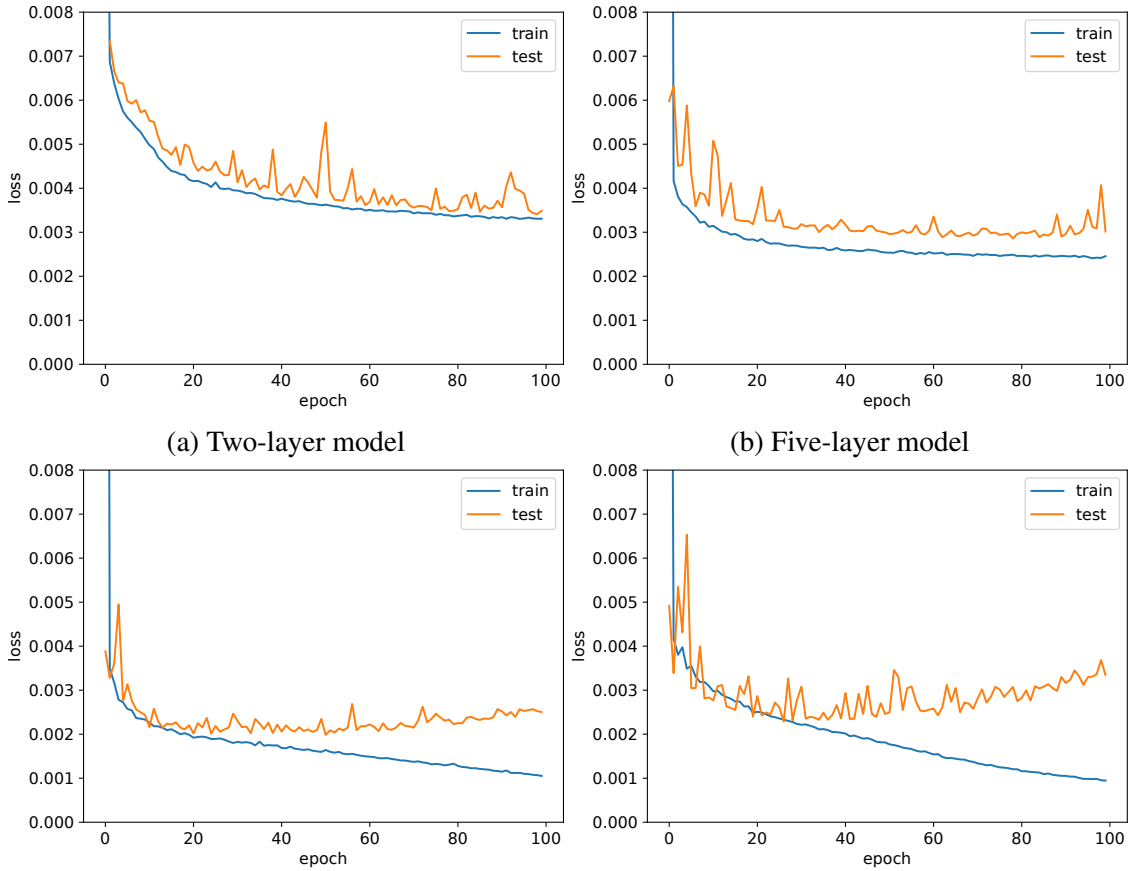


Figure 3.8: Schematic visualization of SegNet. An additional residual is held in each block right before MaxPolling, which is then added back after the corresponding UpSampling step. This operation can preserve the details lost in the down- and upsampling process.



(a) Two-layer model

(b) Five-layer model

(c) Encoder-Decoder

(d) SegNet

Figure 3.9: Training loss and test loss curves for different models. (a)-(d) consist of two-layer, five-layer, an encoder-decoder, and a SegNet architecture, respectively. (a) and (b) illustrate the very similar performance on test loss. Overfitting occurs in (c) and (d).

3.2.3 Comparison

With the ball contours found, the color thresholding and deep learning algorithms can be evaluated in terms of accuracy and speed using the following metrics:

- **IoU**: the Intersection over Union (IoU) metric is used to quantify the percent overlap between the predicted contour and the ground truth:

$$\text{IoU} = \frac{\text{ground truth} \cap \text{prediction}}{\text{ground truth} \cup \text{prediction}}. \tag{3.6}$$

- **Center Error**: average pixel distance error for the contour centers.
- **Speed**: the inference speed per image measured with the testing dataset.

Since our whole project is written in C++, we then convert the deep learning model from TensorFlow to OpenCV DNN module for better compatibility. Table 3.4 shows that deep learning can achieve more accurate ball contours, which results in a higher IoU score than the color thresholding algorithm. The reason is the color threshold values are not adaptive to the lighting in some cases (see Fig. 3.10). However, they have similar center errors that are around 1 pixel, since Eq. 3.5 can post-process the contour candidates and filter out false blobs or lines. For speed testing, it is unsurprising that color thresholding is faster than deep learning. The OpenCV DNN module can slightly accelerate the TensorFlow model in speed (1.67 ms to 1.51 ms per image).

Table 3.4: Comparison of color thresholding and deep learning algorithms in testing dataset. Both of them are written in OpenCV and C++. The input size is 160×160.

Algorithm	IoU	Center Error (pixels)	Speed (ms)
Color Thresholding	69.42%	1.33	**0.67**
Two-layer CNNs	**85.93%**	**0.77**	1.51

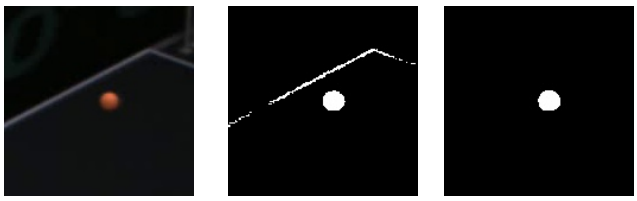


Figure 3.10: An example when the ball is close to the table edge. Left: the cropped RGB image. Middle: the final contour by the traditional approach. Right: the result segmented by deep learning.

If there is no ball in the previous image, we shrink the current image (1280×1024) to half (640×512) so that the segmentation is fast enough at 150 Hz. A ball trajectory is illustrated in Fig. 3.11. When ball pixel positions are known from two or more cameras, we can reconstruct the ball 3D position using triangulation (see Fig. 3.12).

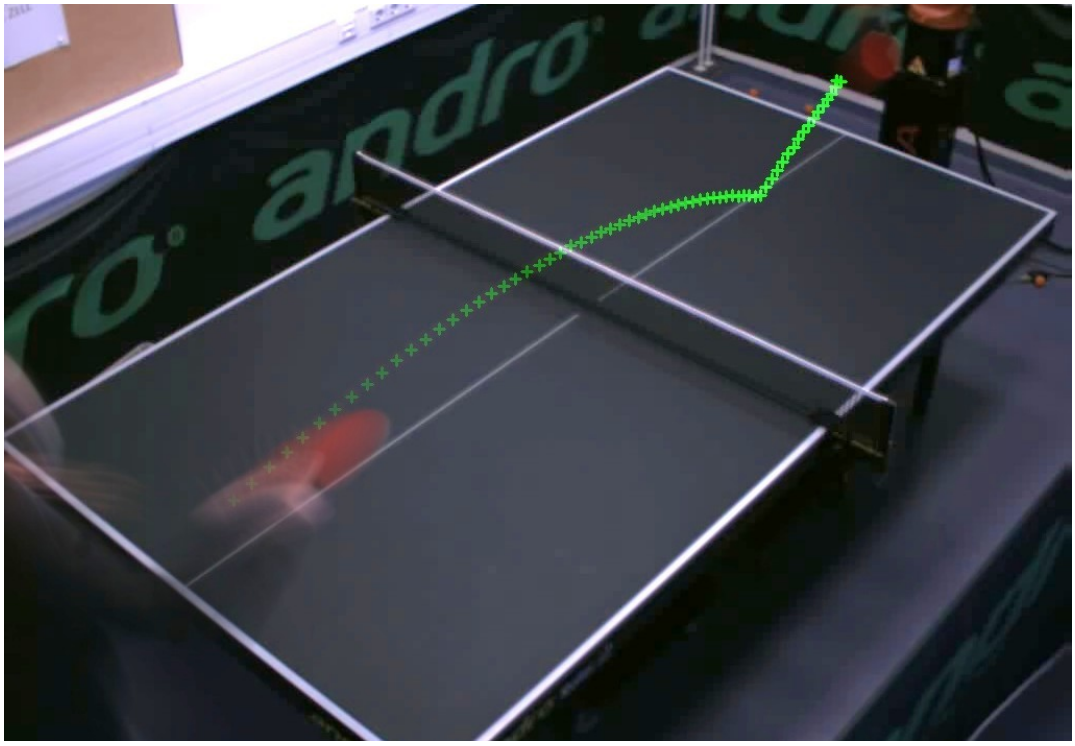


Figure 3.11: A ball trajectory shown in the image.

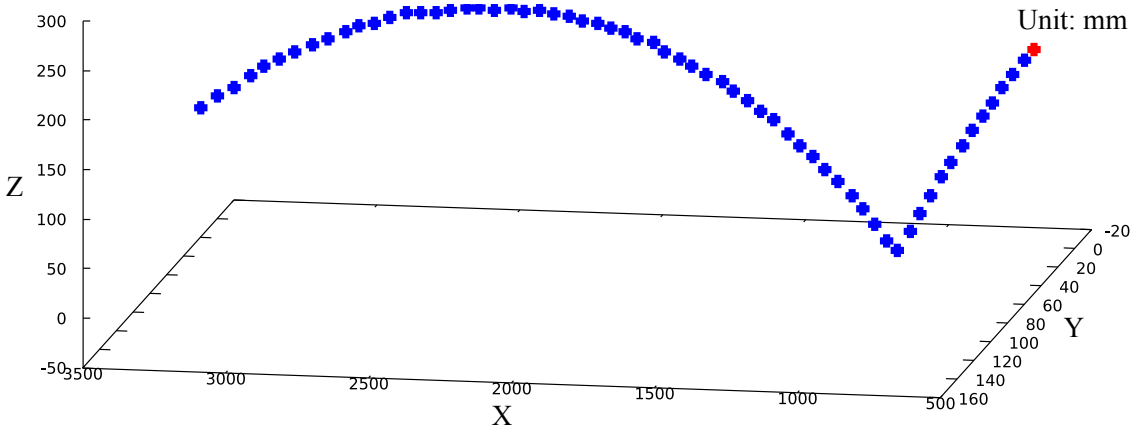


Figure 3.12: A ball 3D trajectory in the KUKA robot coordinates. The red marker is the position where the robot hits the ball.

3.3 Ball Prediction

In this section, we will briefly introduce the ideas for predicting the ball position and spin that are studied in Tebbe *et al.* (2018, 2020). Note that this part only provides guidance for the following chapters and is not the focus of this thesis. When observing the ball's 3D positions in the robot coordinates, we can predict the future trajectory of the ball and the hitting time based on the flying ball model, as shown in Fig. 3.13. An extended Kalman filter (EKF) is adopted to estimate the ball's 3D position p and 3D linear velocity v. A spin detector is used to predict the 3D angular velocity ω. The hitting time is when the ball reaches the desired position along the X axis. Here we set the desired X position to 676 mm in robot coordinates. The evaluation on different spins is shown in Table 3.5. The metric is the distance error between the predicted and the desired bounce positions.

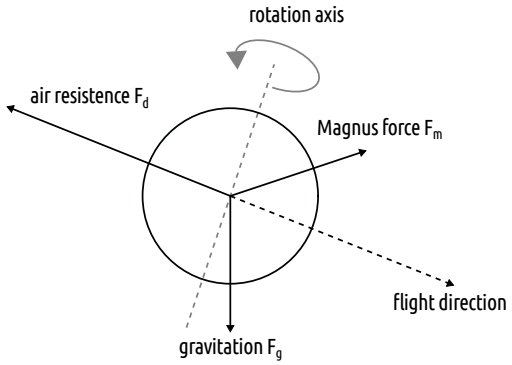


Figure 3.13: Ball flying model: gravitation, air resistance, and Magnus force.

3.4 Conclusions

In this chapter, we developed a calibration approach for multiple stationary cameras in Section 3.1. The evaluation dataset was generated by using the KUKA Agilus R900 sixx robot. To extract the ball pixel positions from images, we proposed two approaches: the traditional image processing that makes use of the color thresholding technique, and deep learning that segments the ball contours based on a manually labeled dataset in Section 3.2. Finally, we briefly introduced the ball trajectory prediction work by Tebbe *et al.* (2018, 2020) in Section 3.3.

Table 3.5: Results on bounce point prediction for balls served from a ball throwing machine with different settings. For each setting we recorded 10 trajectories, thus there are 90 trajectories in total. With fitted spin, the prediction performance is much better than without spin.

		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

Chapter 4

Optimal Stroke Learning with Policy Gradient Approach for Robotic Table Tennis

Learning to play table tennis is a challenging task for robots, due to the variety of the strokes required. Current advances in deep Reinforcement Learning (RL) have shown potential in learning the optimal strokes. However, the large amount of exploration still limits the applicability when utilizing RL in real scenarios. In this chapter, we first propose a realistic simulation environment where several models are built for the ball's dynamics and the robot's kinematics. Instead of training an end-to-end RL model, we decompose it into two stages: the ball's hitting state prediction and consequently learning the racket strokes from it. A novel policy gradient approach with TD3 backbone is proposed for the second stage. In the experiments, we show that the proposed approach significantly outperforms the existing RL methods in simulation. To cross the domain from simulation to reality, we develop an efficient retraining method and test in three real scenarios with a successful return rate of 98%.

Large parts of this work have been submitted in Gao *et al.* (2021a).

4.1 Introduction

Reinforcement Learning (RL) has recently achieved a variety of successes, especially in autonomous driving (Kendall *et al.*, 2019), gaming (Silver *et al.*, 2017; Berner *et al.*, 2019) and robotics manipulation (Gu *et al.*, 2017; Kalashnikov *et al.*, 2018). In RL one studies an agent in its surroundings. Based on the observed state, the agent can take actions in the environment and subsequently perceive a reward that indicates whether these actions were performed well or not. This is defined as a Markov Decision Process (MDP). The goal of RL is to maximize the expected value of the cumulative reward in one episode. However, a large amount of exploration is usually required to formulate a near-optimal policy of actions. For example, OpenAI (Berner *et al.*, 2019) trained the Dota2 game from raw images over 10,000 years (in-game time) against itself on 256

GPUs and 128,000 CPU cores. Therefore, it is particularly hard to train RL models directly for vehicles or robots in the real world.

To address this problem, one common way is creating a simulation environment to train a model and then transfer it to reality. Peng *et al.* (2018) randomly simulated dynamics parameters (link mass, joint damping, puck friction, etc.) for a 7-DOF fetch robot arm and considered them as a part of the current state. Meanwhile, Gaussian noise was applied on each joint position, velocity, and the gripper position. Based on DDPG (Lillicrap *et al.*, 2015), this dynamics randomization method can cross the reality gap and perform equally well on a real robot. With a similar idea, Andrychowicz *et al.* (2020) trained dexterous in-hand manipulation policies entirely in simulation, by randomizing many of the physical properties, such as the friction coefficients and the object's appearance. A scaled PPO (Schulman *et al.*, 2017) algorithm was developed for training. However, dynamic randomization is only meaningful for the case where multiple interactions with the environment arise in an episode, since the proper dynamics, such as mass, friction, damping of each robot joint, need to be revealed in the early interaction steps in each episode. For robotic table tennis there is only one-time interaction with the racket during one stroke, which means there is no way to reveal the proper dynamics in one episode for our robot. (Gao *et al.*, 2020; Mahjourian *et al.*, 2018; Zhu *et al.*, 2018) demonstrated the potential of RL only in simulation. Büchler *et al.* (2020) developed a hybrid RL system that could run in reality, but it was unable to handle balls with various speeds and spins. Another problem caused by the existing RL methods, like TRPO (Schulman *et al.*, 2015), PPO, DDPG, TD3 (Fujimoto *et al.*, 2018), or SAC (Haarnoja *et al.*, 2018) is the fuzzy one-dimensional reward that cannot precisely express the interaction with the environment for multi-dimensional actions.

Inspired by the aforementioned methods, in this chapter we propose a novel approach for optimal stroke learning in robotic table tennis. Two learning steps, including the training in simulation and the retraining in reality, can be completed in around 1.5 hours for various spin balls. The main contributions of this chapter are as follows:

- We design a realistic simulation in combination with the Gazebo simulator, Robot Operating System (ROS), OpenAI Gym (Brockman *et al.*, 2016) and the RL library Spinning Up. The robot and table tennis ball are controlled by the Gazebo plugin. Different parts can communicate with each other via ROS topics.

- We decompose the learning strategy into two stages: the ball hitting state prediction and the optimal stroke learning, on which we mainly focus in this work. Based on the controllable and applicable actions in our robot, a multi-dimensional reward function and Q-value model are proposed.

- A comparison with other RL methods is performed using an evaluation dataset of 1000 balls in simulation. An efficient retraining approach is proposed to close the sim-to-real gap. The testing experiments in reality show that the robot can

successfully return the ball to the desired target with the error of around 24.9 cm and a successful return rate of 98% in three different scenarios.

4.2 Related Work

4.2.1 Simulation for Robotic Table Tennis

In simulated environments, the robot can freely explore various actions and alleviate the safety concerns during the training steps. Meanwhile, simulation can provide a fair and deterministic environment for comparing the performance of different approaches. In Mahjourian *et al.* (2018), PyBullet was used to create the physical entities. In order to learn from demonstrations, they connected a virtual reality (VR) setup with the simulator and captured the human actions with a instrumented racket. In Koç *et al.* (2018), a simple simulation was built on top of MATLAB. To make use of robotic drivers and devices, Silva *et al.* (2015) implemented a quadrotor model in the Gazebo simulator, which can be easily combined with ROS. Büchler *et al.* (2020) developed a hybrid simulation and real training for muscular robots. They duplicated the incoming ball's state from reality to simulation and then simulated the impact with the racket to estimate the ball's landing position. The real robot was copied to the simulation by overwriting the simulated with the real robot action at every time step. In this way, the simulation and the real scenario can remain partially identical. However, these simulations are not realistic enough, which results in more effort on transferring from simulation to reality. In addition, some of them are not compatible with the existing RL libraries that include some advanced RL algorithms. Therefore, we designed a realistic simulation in combination with the Gazebo, ROS, OpenAI Gym and Spinning Up. The ball's dynamics is determined based on Blank *et al.* (2017) and Tebbe *et al.* (2020). The simulated robot is controlled in Cartesian space.

4.2.2 Reinforcement Learning in Robotic Table Tennis

Recently, deep RL has attracted a lot of interest by researchers in robotic table tennis. Gao *et al.* (2020) developed an end-to-end algorithm to directly learn to control a simulated table tennis robot in joint space. They took the joint position trajectories and ball locations as input and trained a multi-modal model-free policy to learn the velocities for each joint. In Büchler *et al.* (2020), a hybrid simulation and reality system was introduced to train a muscular table tennis robot in joint space. They leveraged PPO (Schulman *et al.*, 2017) as the backbone. To return and smash the ball with a high successful return rate, they developed a dense reward function that depends on the ball position and the robot state. However, these end-to-end approaches are not efficient for training, they needed about 14 hours to train for a set of similar ball trajectories. Mahjourian *et al.*

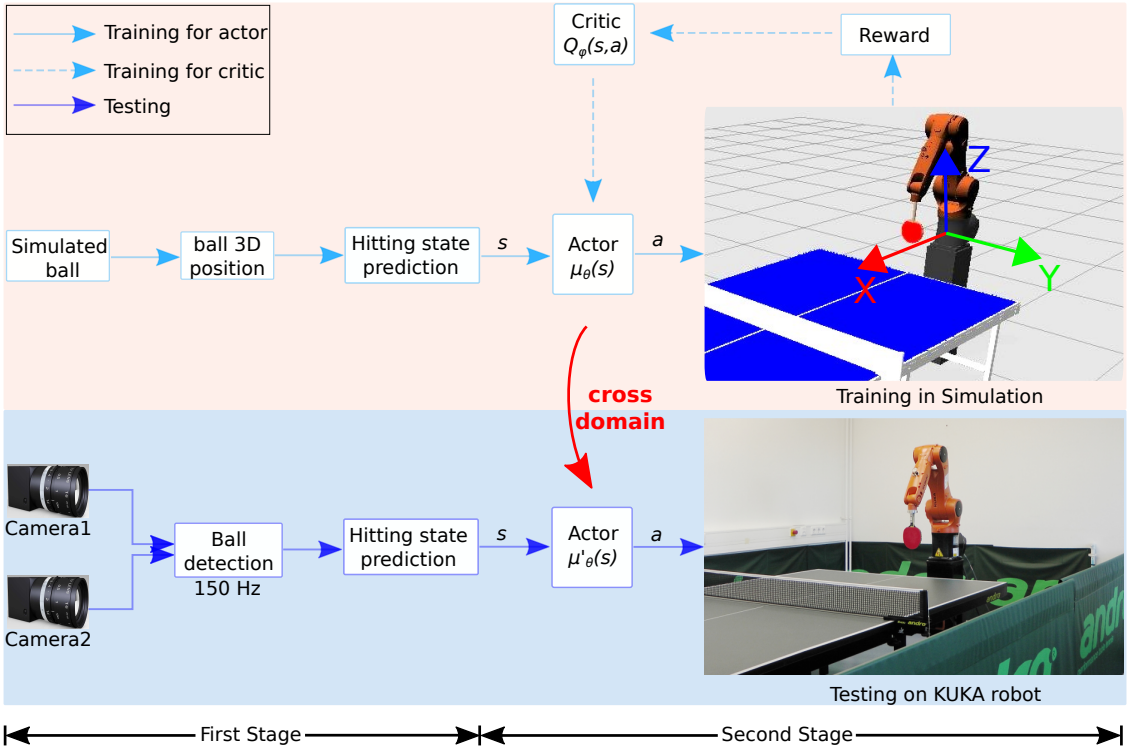


Figure 4.1: The whole framework for training and testing. The first stage is used for predicting the ball's state s at the hitting point at a frequency of 150 Hz. In this chapter we focus on the second stage where a optimal stoke can be learnt based on a novel RL algorithm. The upper part is performed in simulation. A proposed RL model is trained within 10,000 episodes. To cross the "reality gap", we retrain the model in an efficient way. The world coordinate system is identical to the robot's, which is shown in the simulation image.

(2018) incorporated the stroke learning into a hierarchical control system that includes the inverse landing model, analytic racket controller, forward racket mode and forward landing model. Each model was trained separately to make the learning process easier and more efficient. Zhu *et al.* (2018) adopted a two-stage approach for stroke learning. In the first stage, the ball's hitting states (position and velocity) were determined by an extended Kalman filter (EKF) based predictor. Then, these states were fed into DDPG (Lillicrap *et al.*, 2015) as inputs, the outputs were the racket's velocity.

4.3 Methodology

To learn the optimal stroke efficiently and hit the ball to the desired target on the table successfully, we propose a novel framework, as shown in Fig. 4.1. A realistic simulation environment is developed for robot learning and for comparison with other advanced RL algorithms. The ball's hitting state (position, velocity, spin) can be predicted by the approaches in Section 3.3. In the second stage, a novel approach is employed to learn the optimal stroke in simulation, which is conjugated with ROS and OpenAI libraries (see Fig. 4.2).

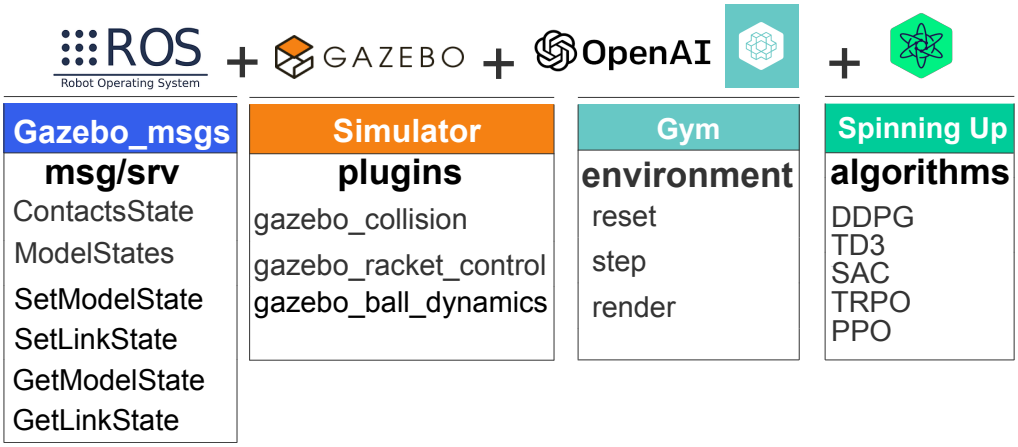


Figure 4.2: The learning architecture, by combining the Gazebo simulator with ROS, Gym, and Spinning Up. We can subscribe and publish the states (pose and velocity) of the ball and the robot from Gazebo with the `Gazebo_msgs` in ROS. These states should be fed into the Gym toolkit that allows us to use the RL algorithms in Spinning Up.

4.3.1 Simulation

A challenge for deep RL is how to safely interact with the environment. In robotic table tennis, it is difficult to explore every possibility since unexpected collisions would

destroy the mechanical robot parts. In addition, the robot has to interact with the environment for a large number of steps to learn a high level policy. To address these problems, we develop a realistic simulation that can provide a convenient scenario for optimal stroke learning as well as a comparison of different algorithms. The pose and velocity of the racket is controlled by the simulator. The dynamics models of the ball are as follows:

Flying Ball Model

Except for the gravitational force F_g, a flying ball is usually influenced by the Magnus force F_m and the air drag F_d (Zhang *et al.*, 2014). As shown in Fig. 4.3, F_m is perpendicular to the spin axis and the flight direction. F_d is opposite to the flight direction.

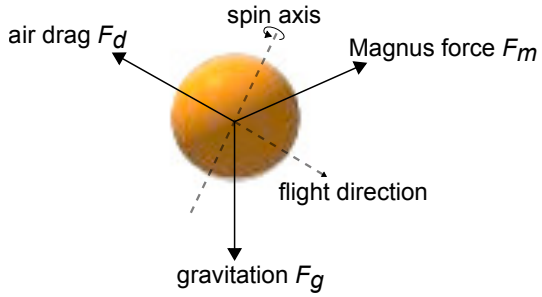


Figure 4.3: Force analysis in a flying ball. A sphere shell of radius r_1 and mass m, with centered spherical cavity of radius r_2, is created as the simulated ball in Gazebo.

These forces can be computed by the following formulas:

$$F_g = (0,0,-mg)^T \tag{4.1}$$

$$F_d = -\frac{1}{2} C_D \rho_a A \|v\| v \tag{4.2}$$

$$F_m = \frac{1}{2} C_M \rho_a A r_1 (\omega \times v) \tag{4.3}$$

where the constants are determined in this work (Tebbe *et al.*, 2018), including the ball's mass $m = 2.7$g, the gravitational constant $g = 9.81\text{m/s}^2$, the drag coefficient $C_D = 0.4$, the air's density $\rho_a = 1.29\text{kg/m}^3$, the lift coefficient $C_M = 0.6$, the ball's radius $r_1 = 20$mm, the cavity radius $r_2 = 19.6$mm, and the ball's cross-section $A = r_1^2\pi$. ω and v are the linear and angular velocity. Given the ball's trajectory in reality, ω and v can be derived for the hitting state prediction by the approaches stated in Section 3.3.

To simulate the accurate dynamics of the ball, we also need the inertia value I calcu-

lated by

$$I = \frac{2}{5} m \left(\frac{r_1^5 - r_2^5}{r_1^3 - r_2^3} \right). \tag{4.4}$$

Bounce Model

In reality, the physical contact between two objects is a very complex thing. To approximate the contact forces between the ball and the table (or the racket), we adopt the Open Dynamics Engine (ODE) which is a popular rigid body dynamics library for robotics. It is already built in the Gazebo simulator. To represent the elastic and frictional impacts on the ball, we compute the restitution coefficient κ_R and the friction coefficient μ similar to Blank *et al.* (2017).

The restitution coefficient κ_R is defined as the ratio of the energy before and after a collision, for example, when the ball bounces off the table. Approximately, it can be solved by a free fall of the ball as follows:

$$\kappa_R^t = \frac{v_2^t - v_2^b}{v_1^b - v_1^t} = \frac{-v_2^b}{v_1^b} = \frac{-\sqrt{2 \cdot g \cdot h_2}}{-\sqrt{2 \cdot g \cdot h_1}} = \sqrt{\frac{h_2}{h_1}} \tag{4.5}$$

where v_1^b and v_1^t are the velocity of the ball and table before impact, v_2^b and v_2^t are after. h_1 and h_2 are the corresponding heights when the ball is not moving. Here the table velocity $v^t = 0$.

The friction coefficient μ is obtained by the setup in Fig. 4.4. Three balls are arranged together in form of a triangle frame. We first put them on the table, and lift the table until they begin to slide. According to the horizontal angle change θ of the table, we can get the friction coefficient μ^t between the table and the ball by

$$\mu^t = \frac{3mg \cdot \sin\theta}{3mg \cdot \cos\theta} = \tan\theta. \tag{4.6}$$

Figure 4.4: Setup for measuring the friction coefficient μ.

We use the same methods to compute the racket's restitution coefficient κ_R^r and friction

coefficient μ^r. The resulting parameter values are in the Table 4.1. Additional required parameters μ_2 and $slip$ are defined as the friction coefficient in second ODE friction pyramid direction and the coefficients of force-dependent-slip (FDS), respectively. They are manually adjusted to fit the reality.

Table 4.1: Collision parameter values when the ball impacts on the table and racket.

	κ_R	μ	μ_2	$slip$
Table	0.97	0.05	0.025	0.01
Racket	0.9	1.0	0.025	0.01

To roughly test the accuracy of the simulation, we utilize a ball throwing machine to launch a topspin ball towards the stationary racket mounted on the robot. The whole trajectory can be recorded as the ground truth with stereo cameras at 150Hz. The starting spin and velocity of the ball are computed by a spin detector tool and a curve fitting approach. Then, we re-serve this ball in our simulation and generate a simulated trajectory shown in Fig. 4.5. The difference between the returned landing positions on the table is about 6.2 cm.

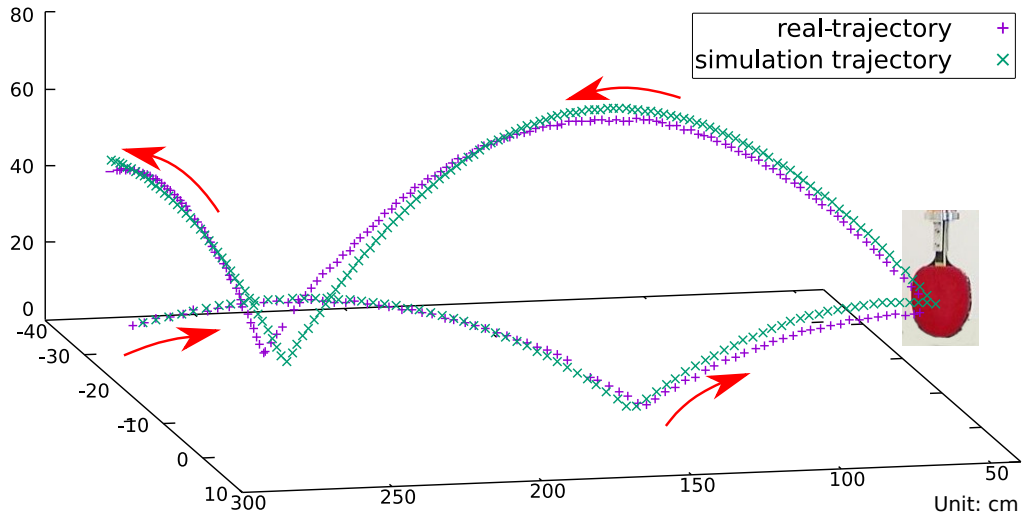


Figure 4.5: Trajectory Comparison between reality and simulation when serving a topspin ball. The difference between the returned landing positions on the table is about 6.2 cm. In this case the racket is stationary since it is difficult to understand and simulate the entire dynamics parameters of a moving racket in simulation. Therefore, the racket's actions applied in simulation are always the true actions without any noise and time latency. This "reality gap" will be closed in the retraining chapter 4.4.3.

4.3.2 Algorithm

With regard to the different types of inputs, there are usually two ways available when using deep RL algorithms in robotic table tennis. The first are one-stage algorithms, which take the ball's state of every step as inputs and learn the racket's pose in an end-to-end way. Others are two-stage algorithms, which first predict the ball's hitting state and then consider it as inputs. The latter one can significantly accelerate the training step and can handle different spin balls. In this thesis we adopt the second way to learn the optimal stroke of the racket based on the state prediction of the ball at the hitting point.

Since there is only a single state vector as input in the second stage, we then parameterize the stroke learning as a bandit problem, where actions have no influence on next states and consequently there are not any delayed rewards in one episode. It is a simple version of an Markov Decision Process (MDP), with

$$M = (S, A, R) \tag{4.7}$$

where S is the set of the observed 11-D states s including the ball 3D position p^b, 3D linear velocity v^b, 3D angular velocity ω^b at the hitting time, and the desired 2D landing target p^{tar} on the table. A is the set of 3D actions a that can be performed on the robot. Due to the restriction of the current mechanical structure and the control system, we can not operate the robot as flexibly as a human can move. Therefore, we only learn to change the robot's linear velocity v_x^r along the x-axis and the orientation angles (β^r, γ^r) around the y and z axes. The racket's target position is the same as the predicted hitting position of the ball. This makes the robot easy to control in the real world. r donates the reward function for computing the immediate reward in each episode.

We utilize a policy $\mu_\theta(s)$ as the actor network, which can output the actions a with respect to the current state s as shown in Fig. 4.6 left. To evaluate the actions, a critic network $Q_\phi(s,a)$ is used, which takes as input both the states and actions and outputs a Q-value, as shown in Fig. 4.6 right. θ and ϕ are the neural network weights. The goal is to lean a deterministic policy $\mu_\theta(s)$, which provides an action that maximizes $Q_\phi(s,a)$. According to the DDPG algorithm, the critic and the actor can be updated, respectively, by minimizing the losses:

$$\mathcal{L}(\phi, \mathcal{D}) = \underset{(s,a,r)\sim\mathcal{D}}{\mathrm{E}} \left[\left(Q_\phi(s,a) - r \right)^2 \right] \tag{4.8}$$

$$\mathcal{L}(\theta, \mathcal{D}) = -\underset{s\sim D}{\mathrm{E}} \left[Q_\phi \left(s, \mu_\theta(s) \right) \right] \tag{4.9}$$

where D is the experience replay buffer for storing s, a, r. The reward r is the feedback from the environment, which we will discuss in more detail later.

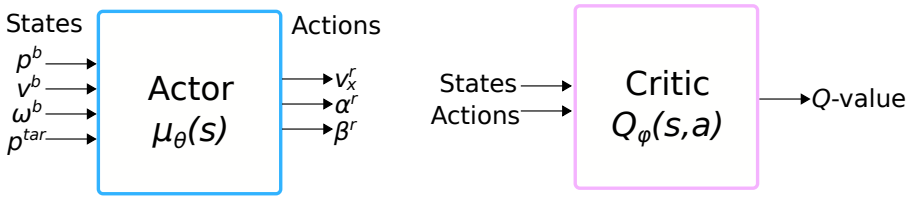


Figure 4.6: Classic Actor-Critic algorithms. Instead of a 1D Q-value, we propose a 3D Q-value to train the corresponding 3D actions.

To accelerate the training step and boost the resulting performance, we apply the following changes to the classic actor-critic algorithms during training:

Exploration

For continuous action spaces several exploration policies are used in deterministic environments. The *random* strategy selects the actions randomly from a Gaussian distribution, the $epsilon-greedy$ strategy takes the random actions occasionally with probability ε and uses the output from the current actor $\mu_\theta(s)$ with probability $1-\varepsilon$. In Xu *et al.* (2018) an additional stochastic policy is deployed to learn how to explore. We noticed that the actor $\mu_\theta(s)$ did not give the action with the maximum Q-value in the earlier training step because of the large loss error. Therefore, we generate the actions a by

$$a = \underset{\mu_\theta(s)+\mathcal{N}}{\operatorname{argmax}} \| Q_\phi(s, \mu_\theta(s)+\mathcal{N}) \| \tag{4.10}$$

where $\mathcal{N}$ is a Gaussian noise.

Reward shaping

In Zhu *et al.* (2018), they develop a reward function that depends on the ball's height h^b across the net and the real landing position p^{real} on the table when the ball is returned. A coefficient value is used to balance the h^b and p^{real}. Then a tricky part is how to select this value. To address this problem, we separate the reward into three vector components: r_h for the height, r_x and r_y for the landing position. Each reward function then is normalized to [0,1] by the following equations:

$$r_x = e^{-|p_x^{real} - p_x^{tar}|} \tag{4.11}$$

$$r_y = e^{-|p_y^{real} - p_y^{tar}|} \tag{4.12}$$

$$r_h = e^{-|h^b - 0.173|} \tag{4.13}$$

$$r = [r_x, r_y, r_h] \quad \text{if } success \text{ else } \overrightarrow{0} \tag{4.14}$$

where p_x and p_y are the landing position in meters along the x and y axes, 0.173 is the net height in meters, e is the natural exponential operation. When a ball is successfully returned to the opposing table, *success* is set to true.

3D Q-value

Normally, the Q-value is a scalar which is expected to be maximized. To make use of the above rewards, we replace the last layer in the critic network from 1D to 3D. This results in a 3D Q-value $[Q_x, Q_y, Q_h]$, which can precisely indicate the quality of the actions.

In addition, for the actor-critic model we adopt the Twin Delayed DDPG (TD3) algorithm as the backbone. The critic is changed to:

$$Q_\phi(s,a) = \begin{cases} Q_{\phi_1}(s,a), \text{ if } \|Q_{\phi_1}(s,a)\| < \|Q_{\phi_2}(s,a)\| \\ Q_{\phi_2}(s,a), \text{ otherwise} \end{cases} . \tag{4.15}$$

The whole training process is depicted in Algorithm 4, where the loss functions $\mathcal{L}(\phi_i, \mathcal{B})$ and $\mathcal{L}(\theta, \mathcal{B})$ are used to update the critic and actor, respectively.

4.4 Experiments

4.4.1 Training and Testing

With random balls in simulation, we can obtain a number of unique states s for each episode. To generalize the trained model, these serves are sampled from a wide range of values. 1,000 serves are collected for a fair evaluation. To bridge the "reality gap", we first apply some Gaussian noise to each ball's 3D position in simulation. Instead of using the true states of the ball in simulation, we then predict the states at the hitting point with the methods in Section 3.3. The predicted hitting position is actually where the simulated racket should move to. This can replicate the real situation and make the trained model more realistic for the real world. The final state range is shown in Table 4.2, which includes the desired landing target (p_x^{tar}, p_y^{tar}), the ball position (p_x^b, p_y^b, p_z^b), linear velocity (v_x^b, v_y^b, v_z^b), and angular velocity $(\omega_x^b, \omega_y^b, \omega_z^b)$ at the hitting point. These states are subsequently normalized as inputs for training. The p_x^b is fixed to form a virtual hitting plane in the first stage.

By considering the robot's mechanical setup, we restrict the robot's linear velocity v_x^r to a range from 0m/s to 2m/s. The orientation angles β^r, γ^r are from $-50°$ to $50°$. The third angle α^r around the x-axis is calculated by

$$\alpha^r = k \cdot \frac{p_y^b}{0.5 \cdot w^t} \tag{4.16}$$

Algorithm 4: Policy Gradient Training with TD3 backbone

Input: Initial actor weights θ, two critic weights ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$, number of episodes T, Gaussian noise $\mathcal{N}$

Output: Optimal policy $\mu_\theta^*(s)$

for *episode=0, T* **do**
  Observe the state s and generate the action a by
    $\text{argmax}_{\mu_\theta(s)+\mathcal{N}} \| Q_\phi(s, \mu_\theta(s) + \mathcal{N} \|)$
  Apply a in the environment and get the reward r
  Store (s, a, r) in the replay buffer $\mathcal{D}$
  Reset the environment
  if *it is time to update* **then**
    Sample a random minibatch $\mathcal{B}$ from $\mathcal{D}$
    for *i=0,1* **do**
      Update the critic by minimizing the loss:
        $$\mathcal{L}(\phi_i, \mathcal{B}) = \underset{(s,a,r)\sim\mathcal{B}}{\mathrm{E}} \left[\| Q_{\phi_i}(s,a) - r \|^2 \right]$$
    end
    if *it is time to update actor* **then**
      Update the actor by minimizing the loss:
        $$\mathcal{L}(\theta, \mathcal{B}) = -\underset{s\sim B}{\mathrm{E}} \| Q_\phi\left(s, \mu_\theta(s)\right) \|$$
    end
  end
end

Table 4.2: state range at the hitting point for training and evaluation.

	training	evaluation
p_x^{tar}	2.55m	
p_y^{tar}	0.0m	
p_x^b	0.675m	
p_y^b	[-0.60m, 0.63m]	[-0.68m, 0.68m]
p_z^b	[-0.01m, 0.34m]	[-0.01m, 0.34m]
v_x^b	[-6.00m/s, -1.35m/s]	[-5.94m/s, -2.52m/s]
v_y^b	[-1.95m/s, 2.16m/s]	[-1.29m/s, 2.02m/s]
v_z^b	[-3.47m/s, 3.15m/s]	[-3.40m/s, 2.60m/s]
ω_x^b	[-127.67rad/s, 110.88rad/s]	[-95.08rad/s, 111.53rad/s]
ω_y^b	[-299.99rad/s, 299.81rad/s]	[-299.62rad/s, 299.73rad/s]
ω_z^b	[-193.81rad/s, 189.65rad/s]	[-189.05rad/s, 189.47rad/s]
Episodes T	10000	1000

where w^t is the table width, k is a coefficient. In this way, the robot will generate a human-like stroke. In principle, the angle α^r will not influence the impact with the ball.

In Equation 4.10, The added action noise $\mathcal{N}$ for exploration is a mean-zero Gaussian distribution with a standard deviation of 0.1. The replay buffer $\mathcal{D}$ has a size of 5,000. The number of training episodes T is 10,000. Other hyperparameters used for actor-critic are given in Table 4.3. The output actions from the actor are scaled to the valid range and then applied to the simulation. These hyperparameters are tuned manually in order to achieve the best performance.

Compared to other environments that require millions of interactions in OpenAI Gym, our model is able to converge after 30 epochs, which took about 1 hour of training. In addition, 1000 episodes were run for evaluation after each epoch. The resulting rewards and the corresponding 3D Q-value are plotted in Fig. 4.7. It is observed that the testing rewards reach a stable level starting from the 20^{th} epoch, although the Q-values have not yet converged to the maximum values.

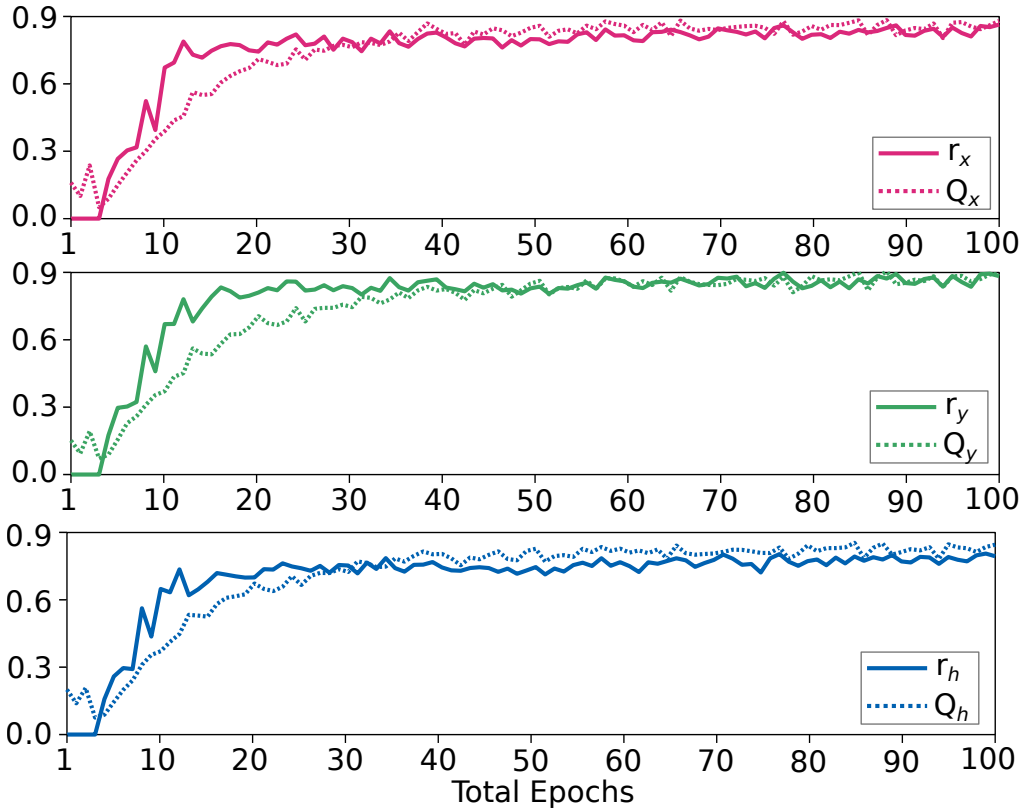


Figure 4.7: The testing rewards $[r_x, r_y, r_h]$ and the 3D Q-value $[Q_x, Q_y, Q_h]$ in simulation for the ball's landing position x, y, and the height h when crossing the net.

Table 4.3: hyperparameters for training in simulation and retraining in reality.

	Actor/Critic	
	Training	Retraining
batch size	512	50
epochs	100	-
episodes per epoch	100	20
learning rate	1e-4	5e-5
optimizer	Adam	
layers	[256,256,3]	
activation	ReLU	
output activation	tanh/linear	

4.4.2 Evaluation

A usual metric for the evaluation is the distance error between the real and the desired landing position (Büchler *et al.*, 2020; Mahjourian *et al.*, 2018; Zhu *et al.*, 2018). However, this metric can not reflect a failed return, for example, if the landing position is not on the table, it will be difficult to calculate the distance error. Therefore, in this thesis, we introduce a new metric: distance error ε_d computed by:

$$r_d = e^{-||p^{real} - p^{tar}||_2} \quad \text{if } success \text{ else } 0 \tag{4.17}$$

$$\varepsilon_d = -\ln\left(\frac{1}{T}\sum_{n=1}^{T} r_d^n\right) \tag{4.18}$$

where the r_d^n is the reward for the landing distance error of the n^{th} ball. It is equal to 0 when the ball fails to land on the opposite table. The number of episodes T is equal to 1000 for evaluation. The second metric ε_h, for the ball flying height error across the net, is calculated in the same way. The third metric is the success rate of returning to the opposite table. To give a fair comparison and evaluation, we adopt 1000 balls resulting in a large range for the states shown in Table 4.2 right.

Since the existing RL algorithms are only allowed to use a 1D Q-value, we then create a 1D reward function similar to Zhu *et al.* (2018) by

$$r_{eval} = e^{-k(||p^{real} - p^{tar}|| + |h^b - 0.173|)} \tag{4.19}$$

where k is a scalar coefficient, which is set to 0.5 in this thesis. This new reward function is only adopted to train the existing RL algorithms including TRPO, PPO, SAC, DDPG, and TD3. As a result of the different reward functions used for evaluation, we compute the distance error ε_D, the height error ε_h and the successful return rate, respectively, which are shown in Table 4.4. The unit of these errors is converted from meters to centimeters for better visualization. The proposed approach, argmax exploration plus 3D Q-value together with TD3 backbone, achieves better performance than the DDPG backbone. Other three approaches, TRPO, PPO, and SAC, learn the optimal stroke using a stochastic policy, which leads to much higher errors and lower return rate.

4.4.3 Retraining in Reality

Although we have built a high-fidelity simulation by manually measuring the coefficients and applying random noise to the ball, the real robot has higher dynamics and complicated factors that can not be accurately measured and included. For example, the robot can not always reach the exact goal at the desired velocity in time, and the performance of the racket will decrease over time with regular use. Therefore, it is necessary to retrain the model in reality. Unlike the methods (Peng *et al.*, 2018; Andrychowicz *et al.*, 2020) that randomize the dynamics parameters in simulation, we adopt an efficient way

Table 4.4: Evaluation for different Algorithms.

Algorithms	ε_d	ε_h	return rate
TRPO	47.0cm	31.0cm	84.8%
PPO	44.2cm	30.8cm	87.1%
SAC	43.5cm	29.0cm	89.2%
DDPG	25.6cm	22.1cm	95.6%
DDPG+argmax	23.0cm	22.3cm	97.4%
DDPG+argmax+3D Q-value	21.3cm	21.7cm	97.9%
TD3	25.2cm	22.3cm	97.2%
TD3+argmax	22.2cm	21.2cm	97.7%
TD3+argmax+3D Q-value	**20.3cm**	**21.2cm**	**98.5%**

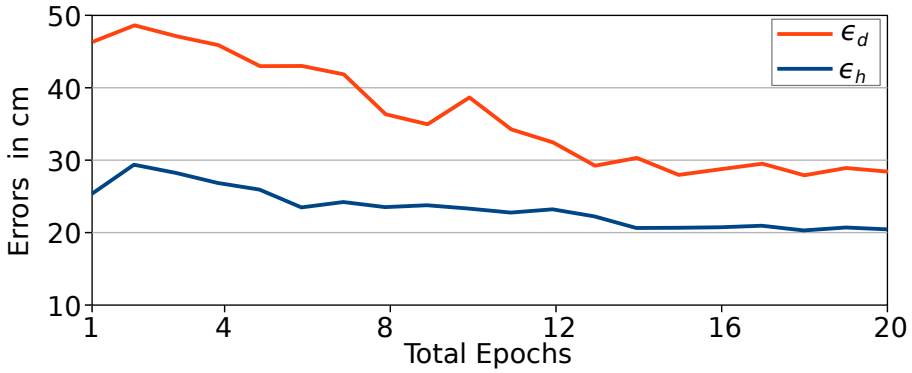
to directly retrain the whole model in our real robot.

To find the best hyper-parameters for retraining, we first change the racket's restitution coefficient κ_R^r and friction coefficient μ^r in simulation. In this way, we can replicate the situation between two different rackets in reality. Based on the pretrained actor-critic model, we then fine-tune all model parameters in the new simulation with various batch sizes, episodes per epoch, and learning rates. The best hyper-parameters found in simulation are shown in Table 4.3 right. The learning rate is one-half of the one in the training step. The number of epochs is different for each new environment.

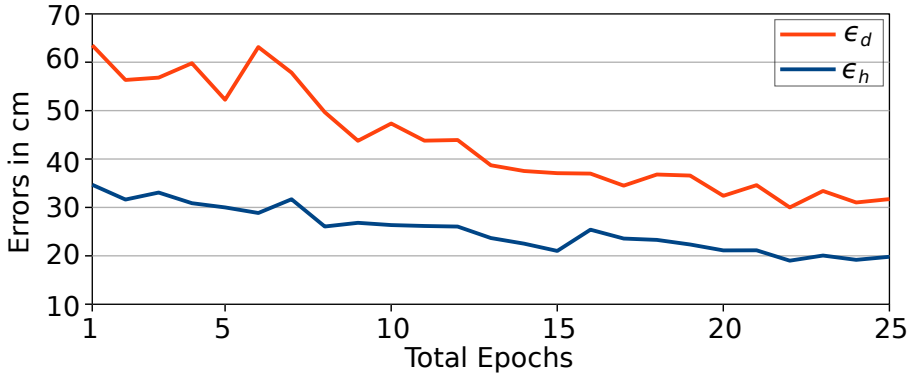
A ball throwing machine, TTmatic 404A, is utilized to provide a range of balls with sidespin, topsin, and backspin. At the moment our robot can only handle sidespin and topspin, as the backspin ball causes too high acceleration in a robot joint. This could be solved in the future. The Reflexxes motion library (Kröger, 2010) is used for robot trajectory planning in Cartesian space. Each epoch includes both sidespin and topspin balls during retraining. The state range for retraining and testing at the hitting point is shown in Table 4.5. Here, the model is retrained with 20 epochs in 0.5 hours to ensure that it achieves convergence. The hitting position p_x^b along the x axis is fixed to 0.675m. The resulting retraining process, including the landing distance error ε_d and the height error ε_h, is plotted in Fig. 4.8a. To investigate the probability for an unknown racket, we also retrain a new model for a second racket whose dynamics are completely different from the first one. The Fig. 4.8b illustrates the second retraining process, which requires more epochs to converge and has similar performance as the first racket.

Table 4.5: State range at the hitting point for retraining and testing.

	retraining/testing in machine	testing with human
p_y^b	[-0.55m, 0.64m]	[-0.65m, 0.43m]
p_z^b	[0.085m, 0.34m]	[0.06m, 0.0.33m]
v_x^b	[-5.20m/s, -3.5m/s]	[-5.6m/s, -2.9m/s]
v_y^b	[-1.05m/s, 2.35m/s]	[-2.38m/s, 1.25m/s]
v_z^b	[-0.78m/s, 3.92m/s]	[-0.4m/s, 2.48m/s]
ω_x^b	[-32.94rad/s, 52.68rad/s]	[-33.00rad/s, 78.48rad/s]
ω_y^b	[-210.52rad/s, 5.33rad/s]	[-182.72rad/s, -55.28rad/s]
ω_z^b	[-157.65rad/s, 34.51rad/s]	[-66.68rad/s, 52.62rad/s]



(a) The first racket



(b) The second racket

Figure 4.8: Retraining process using the ball throwing machine for the original racket (a) and another coefficient-unknown racket (b). ε_d and ε_h in the units of cm are the landing distance error and the height error when ball crossing the net.

4.4.4 Testing in Reality

Various research has shown the achievements of hitting performance on real robots (Mülling *et al.*, 2013; Asai *et al.*, 2019; Büchler *et al.*, 2020). However, they either just attempted a few simple serves or provided unclear scenarios about the state range of the incoming ball. To perform a complete test, we conduct the experiments in three scenarios where the complexity rises gradually. First a human player (Player1) serves the ball with different starting positions at the front of the table. In this way, the hitting position can be fully covered along the y-axis. The second player with senior skills (Player2) then plays a long game rally to test the continuous performance of the robot. The state range for these two scenarios is shown in the third column of the Table 4.5. Finally, we use a ball throwing machine (Machine) to generate different balls with various spins and speeds. It is difficult to fairly compare our performance with other work since the scenarios, like the robot, racket, ball state, and human player, are totally different as well as the evaluation metrics. We then give the Table 4.6 by directly using the data in Mülling *et al.* (2013); Asai *et al.* (2019) or by manually computing for Büchler *et al.* (2020). The average ε_d and ε_h for our three scenarios are 24.9 cm and 22.0cm, with a standard deviation of 9.0 cm and 4.6 cm, respectively. Playing performance, including some failure cases, can be found in a video[1].

Table 4.6: Testing in reality.

Scenarios	episodes	target distance error	height error	return rate
Büchler *et al.* (2020)	107	76.9cm	-	75%
Mülling *et al.* (2013)	30	46.0cm	-	97%
Asai *et al.* (2019)	100	22.5cm	-	-
Player1	40	20.3cm	22.2cm	
Player2	40	25.6cm	23.5cm	98%
Machine	40	28.8cm	20.2cm	

4.5 Conclusions

In this chapter, we first designed a realistic simulation for a table tennis robot. To learn the optimal stroke movement for the robot, we proposed a new policy gradient approach with TD3 backbone. Different algorithms were fairly evaluated in simulation using 1000 balls with a large range of spins and speeds. To cross the domain from simulation to

[1] https://youtu.be/SNnqtGLmX4Y

reality, a retraining approach was proposed for the original racket and another coefficient-unknown racket. The testing results in three complicated scenarios outperform other research with a return rate of 98%. However, the robot will fail if the incoming ball is too high or too slow, since the target can not be reached at a fast enough speed. Also, the robot will not have sufficient reaction time if the ball is too fast (e.g. 10m/s).

Chapter 5

Racket Pose Detection and Stroke Classification based on Stereo Vision

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 chapter, we propose a real-time 3D 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 neural network with two hidden layers 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 stroke classification accuracy is 98%, the same as the human pose estimation method with Convolutional Pose Machines (CPMs).

Large parts of this work have been pre-published in Gao *et al.* (2019a) and Gao *et al.* (2019b).

5.1 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 (Mülling *et al.*, 2011), racket 3D pose detection (Zhang *et al.*, 2017), flying ball position estimation (Lampert and Peters, 2012) and robot trajectory planning (Huang *et al.*, 2015). 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 position will be estimated by combing ball position and spin together using a curve fitting algorithm (Li *et al.*, 2012) or an extended Kalman filter (Zhang *et al.*, 2015). 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 and Saito (2004) 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.* (2018) 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.* (2013) 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.* (2015) 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.* (2017) fused inertial measurement unit (IMU) data with the method (Chen *et al.*, 2013) 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 chapter, 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. The system is shown in Figure 5.1. 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 Section 3.2.1. 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.

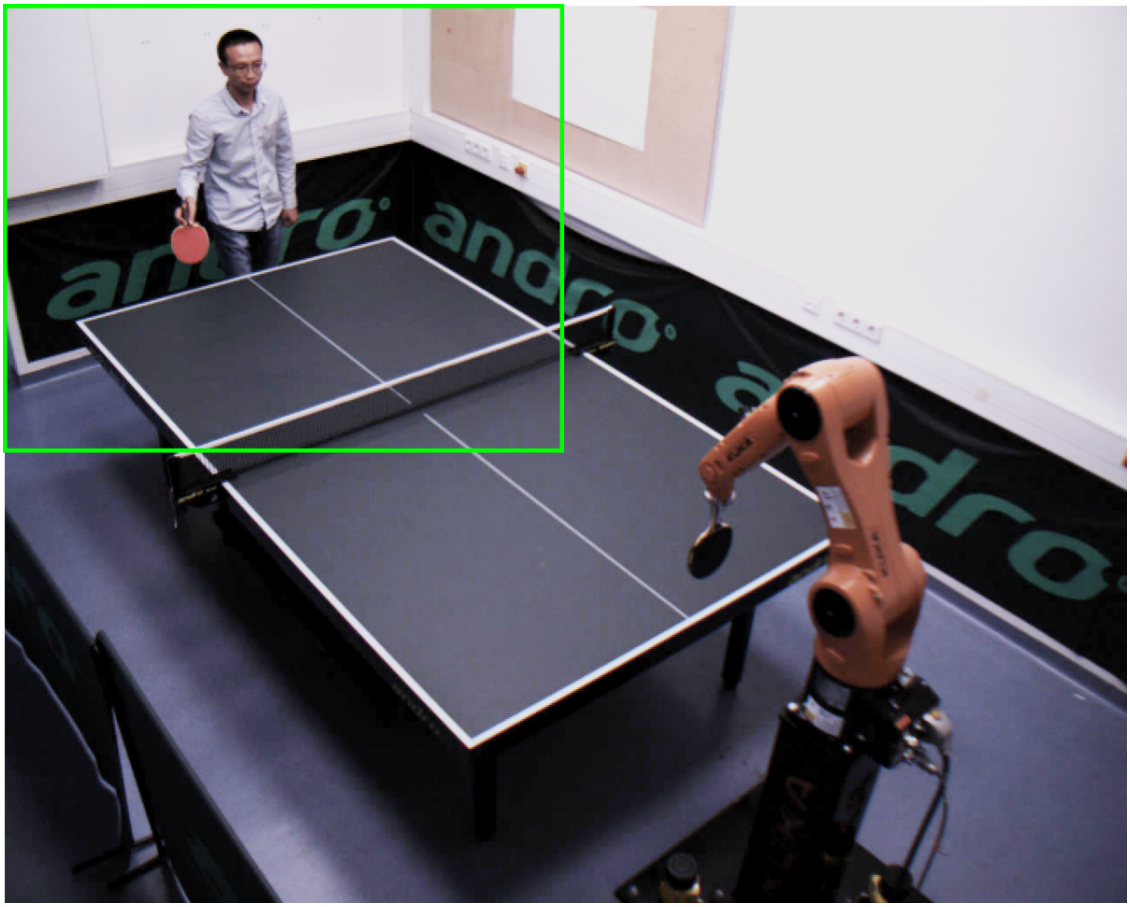


Figure 5.1: Table tennis robot system with KUKA Agilus robot. There are four PointGrey Chameleon3 cameras mounted on the ceiling corners far away from each other to have a large field-of-view, where two cameras opposite to the human are used to detect the racket and another pair is for table tennis ball detection. A table tennis racket is rigidly fixed at the end effector of the robot in a type of penhold grip. The robot coordinate system is set as the world coordinate system and the center of the racket is defined as the tool coordinate center. Each full resolution image (1280×1024) is first cropped into the size of 640×512 to accelerate the following detection process. One case from the left camera is shown in the green rectangle.

The subsequent part of this chapter has the following structure: Section 5.2 introduces related work. The proposed method is presented in Section 5.3. The evaluation and comparison are examined in Section 5.4.

5.2 Related Work

5.2.1 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 Micusik and Wildenauer (2015), 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.

5.2.2 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 (Lane and Thacker, 1998) . 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 (Tareen and Saleem, 2018). Fig. 5.2 shows an example for the racket stereo matching. 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) (Hirschmuller, 2005), 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 CNN models, consisting of embedding, matching, regularization and refinement modules (Tulyakov *et al.*, 2018). However, they currently cannot yet fulfill real-time requirements.

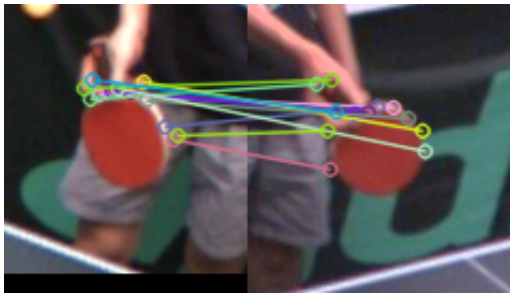


Figure 5.2: Stereo matching example using the ORB keypoint detector and a brute-force descriptor. Almost every matching pair is wrong because the racket surface is low-texture.

5.2.3 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 and Situmeang (2017) 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.* (2015) 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 and Dong (2015) 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 an 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.

5.3 Approach

5.3.1 Racket Detection

To lower the impact on lighting variations, we choose the HSV color space instead of RGB and adopt the color thresholding algorithm similar to Section 3.2.1 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. 5.3 illustrates the pipeline of racket detection in the right camera, which includes four steps.

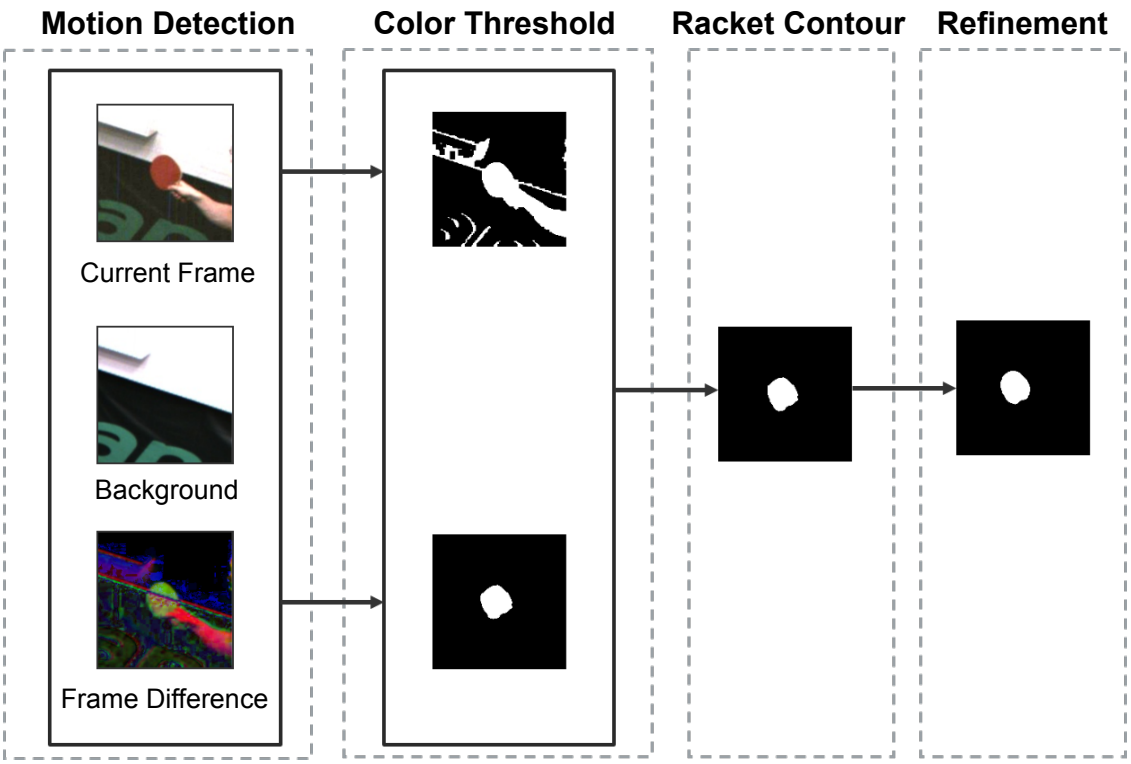


Figure 5.3: Racket detection process with dynamic window, which is for the right image in this case. *Motion detection*: subtract the background from current frame. *Color Threshold*: compute the binary image from HSV space. *Racket Contour*: bitwise AND operation from previous step. *Refinement*: bucket fill results.

We primarily find the moving objects using a static frame difference method by subtracting the background from the 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:

$$\begin{cases} 200px \leqslant & Area & \leqslant 3000px \\ 0.3 \leqslant & AreaExtent & \leqslant 1 \\ 0.3 \leqslant & AspectRatio & \leqslant 3 \end{cases} \tag{5.1}$$

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 conditions 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:

$$C(x,y)_H - L_H \leq C'(x,y)_H \leq C(x,y)_H + U_H \tag{5.2}$$

$$C(x,y)_S - L_S \leq C'(x,y)_S \leq C(x,y)_S + U_S \tag{5.3}$$

$$C(x,y)_V - L_V \leq C'(x,y)_V \leq C(x,y)_V + U_V \tag{5.4}$$

where H,S,V are the components from the 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 the maximal lower or upper color difference between the seed point and one of its neighbors. Fig. 5.3 shows that bucket fill yields better results than color thresholding.

5.3.2 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.

Feature Extraction

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

Figure 5.4: Feature (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, feature 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 Bradski (2000) 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:

$$D = \frac{N_{overlapping}}{N_{edge}} \tag{5.5}$$

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 5.1. 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. 5.4.

Table 5.1: Comparison of ellipse fitting models.

Methods	LS	AMS	Direct
Degree	53.77%	56.60%	58.33

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. 5.5 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:

$$\begin{aligned} P_1 &= Triangulate(P_L, P_R) \\ P_2 &= Triangulate(P_L, P'_R) \\ 75 \leq |Pos - Center| &\leq 90 \quad Pos \in [P_1, P_2] \end{aligned} \tag{5.6}$$

where 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 ($\left| \|Pos - Center\| - \frac{75+90}{2} \right|$).

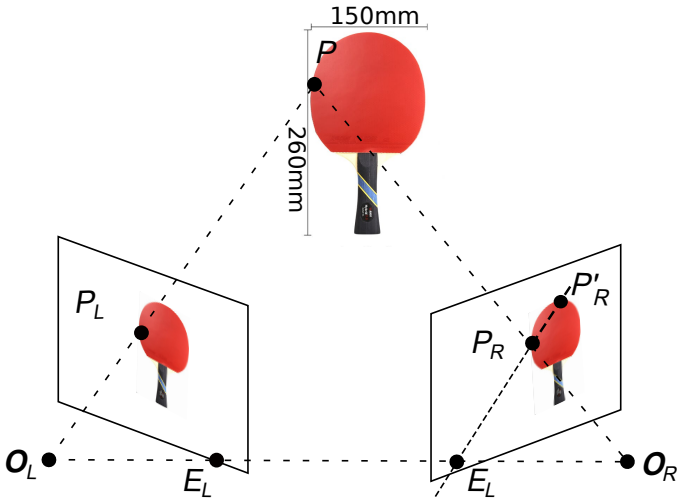


Figure 5.5: Finding the potential candidates P_R and P'_R in the right camera corresponding to P_L. O_L and O_R are the optical centers of the cameras lenses. The epipolar line in the right camera passes through the epipole E_R, the image points P_R and P'_R.

Outliers Removal

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

estimated by:

$$s \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{5.7}$$

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:

$$\sum_i \left(x_i' - \widehat{x_i}\right)^2 + \left(y_i' - \widehat{y_i}\right)^2 \tag{5.8}$$

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 coordinates.

However, using the whole pairs for matrix estimation will lead to a poor result. We utilize the Random SAmple Consensus (RANSAC) (Fischler and Bolles, 1981) 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. 5.6 including 3 penhold and 3 shakehand types. Each corresponding features pair in the left and right cameras is labeled with the same color to be distinguished clearly.

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:

$$\begin{bmatrix} x_0 & y_0 & 1 \\ y_1 & y_1 & 1 \\ & \cdots & \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} z_0 \\ z_1 \\ \cdots \\ z_n \end{bmatrix}. \tag{5.9}$$

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

$$A_{n\times 3} = U_n S_{n\times 3} V_{3\times 3}^T \tag{5.10}$$

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. 5.7. The stereo matching needs around 6.1 ms. The total time for racket

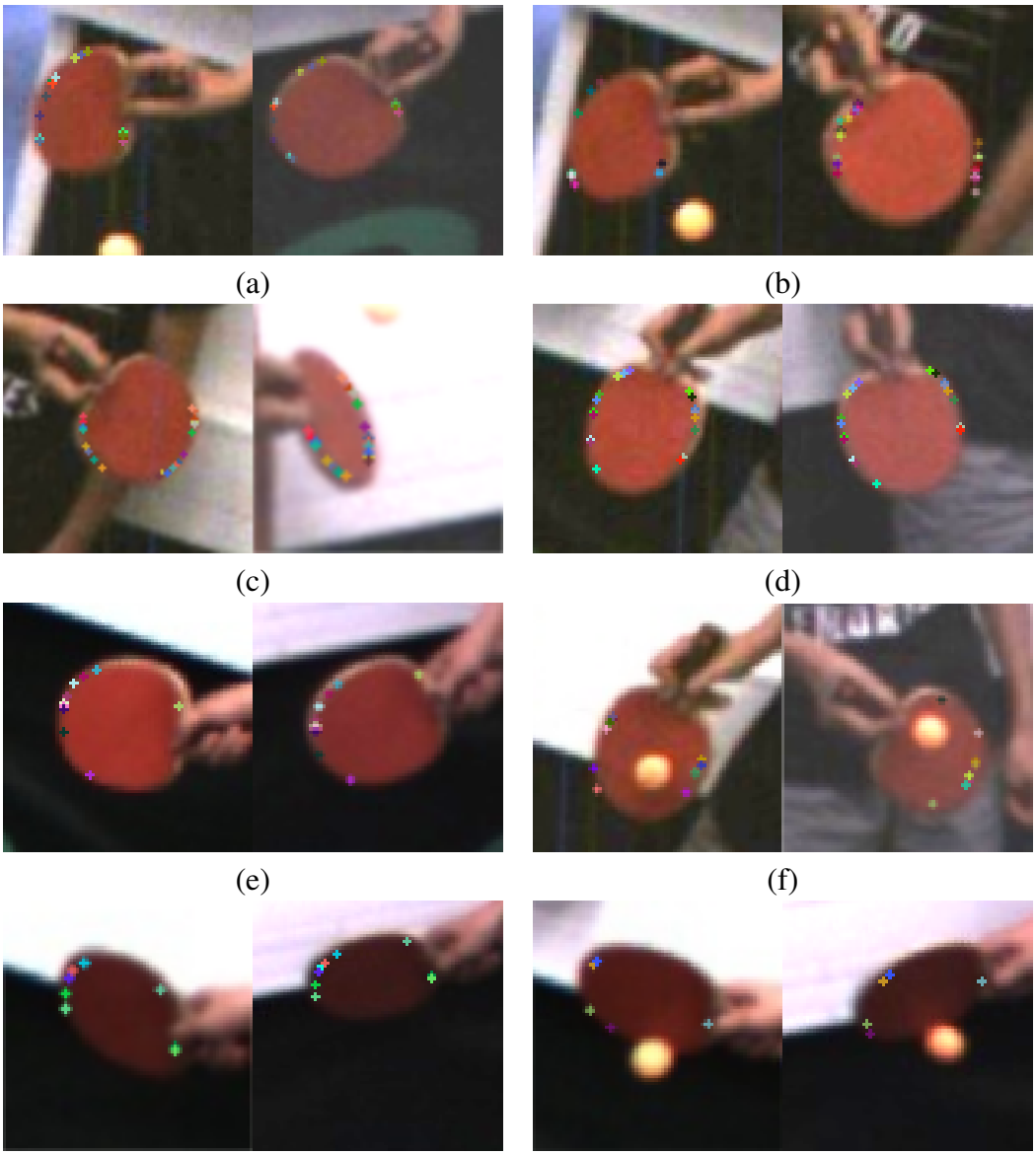


Figure 5.6: Feature matching results including six examples in the left and right cameras. The grip type in (a)-(f) is penhold grip. (g)-(h) use the shakehand grip. The corresponding pairs are labeled with the same color in order to clearly distinguish the correct pairs.

pose estimate needs about 7.0 ms, which means we can estimate the racket 6D pose at 150 FPS.

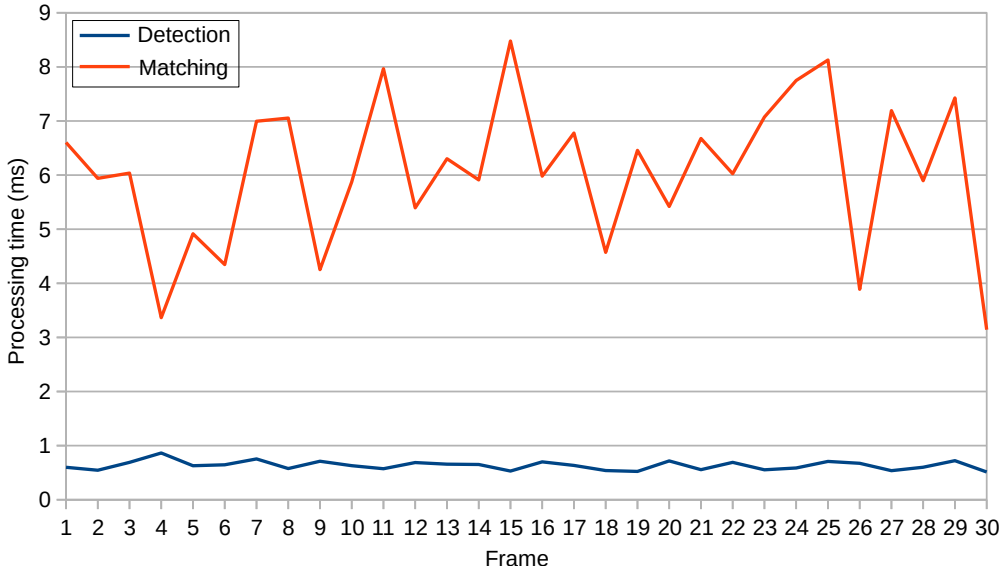


Figure 5.7: Processing time for racket detection and matching. The matching speed is impacted by the number of keypoints.

5.3.3 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 thesis, we employ a discrete Kalman filter that is very efficient and powerful for estimating the pose $[x, y, z, \alpha, \beta, \gamma]^T$. We define the racket state X_t with 15 variables:

$$X_t = [x_t, y_t, z_t, \dot{x}_t, \dot{y}_t, \dot{z}_t, \ddot{x}_t, \ddot{y}_t, \ddot{z}_t, \alpha_t, \beta_t, \gamma_t, \dot{\alpha}_t, \dot{\beta}_t, \dot{\gamma}_t]. \tag{5.11}$$

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

$$\begin{aligned} p_{t+1} &= p_t + \dot{p}_t * \Delta t + \tfrac{1}{2}\ddot{p}_t * \Delta t^2 \\ \dot{p}_{t+1} &= \dot{p}_t + p_t * \Delta t \\ \theta_{t+1} &= \theta_t + \dot{\theta}_t * \Delta t \end{aligned} \tag{5.12}$$

where $\boldsymbol{p} \in [x, y, z]$ and $\theta \in [\alpha, \beta, \gamma]$. Then, we can project the next state and error covariance ahead from current time and update them with the current measurement. From Fig. 5.8 and 5.9, we note that in 30 frames, the estimated pose appears considerably smoother than the original one without Kalman filter.

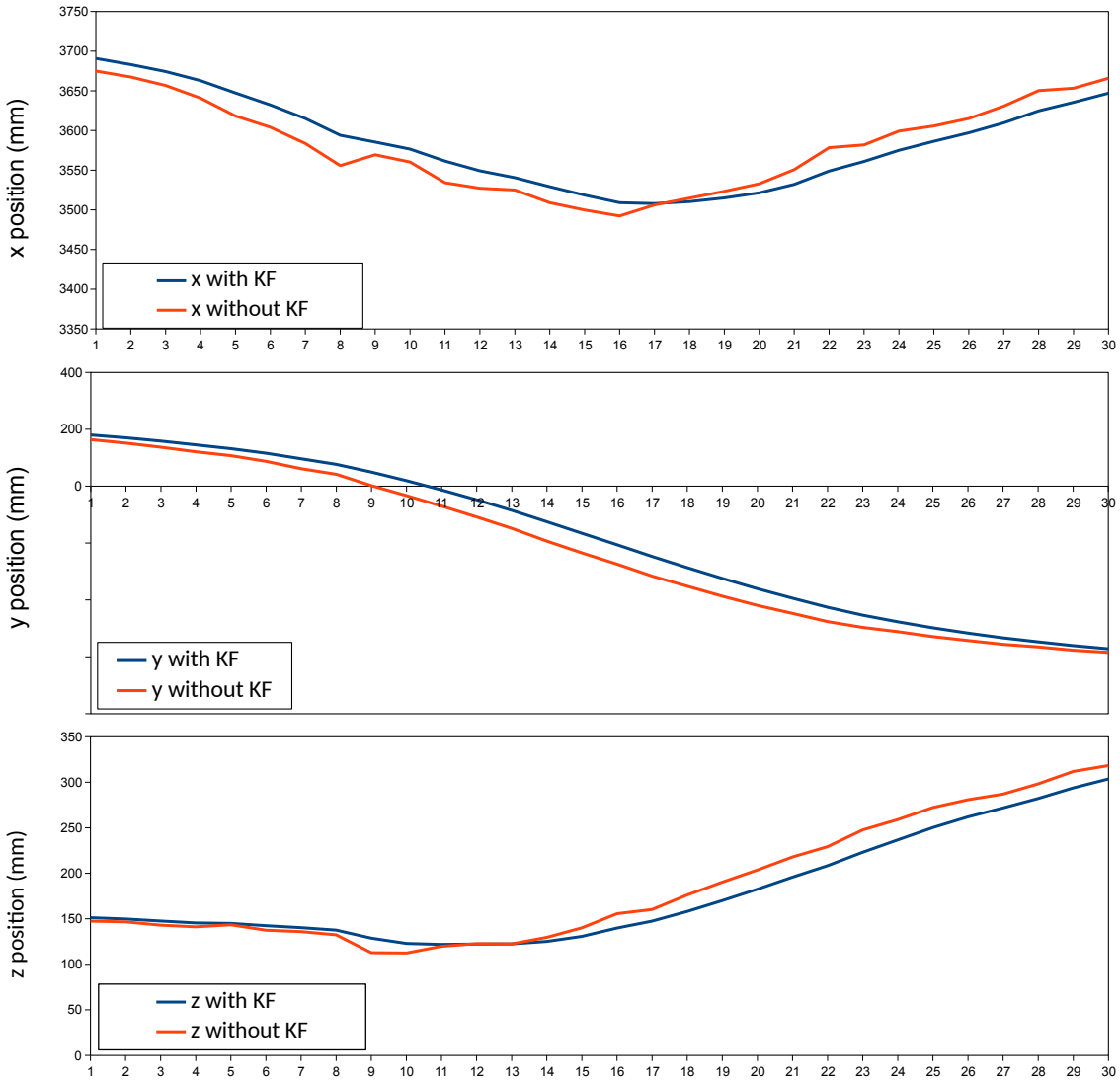


Figure 5.8: Kalman filter tracking in 30 frames for racket position (x, y, z).

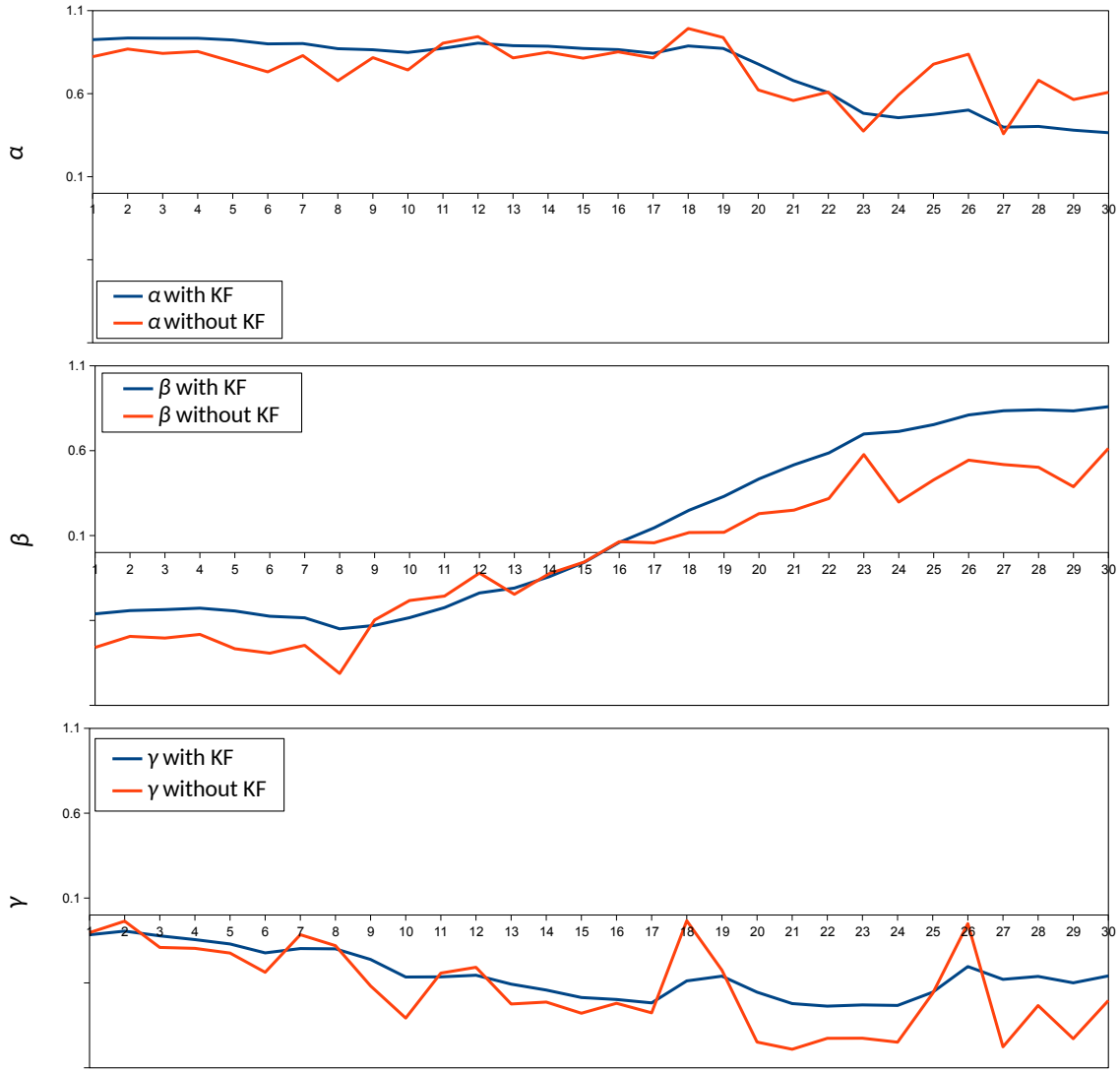


Figure 5.9: Kalman filter tracking in 30 frames for racket orientation (α, β, γ).

5.3.4 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 stroke 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 makes 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 needs 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 the testing set:

Flatten operation

The input size 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 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.

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:

$$output = ReLU(input \cdot kernel + bias) \tag{5.13}$$

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

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%.

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 5.2, we can find the best performing is the 6D pose. The accuracy with 3D orientation is much better than the 3D position

Table 5.2: 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%

5.4 Experiments

In this section, we conduct two experiments to evaluate the performance of our proposed methods. 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. 5.1, 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 (Wei *et al.*, 2016), 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.

5.4.1 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. 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. 5.10. Next, we transform this vector to robot coordinates (namely, world coordinates).

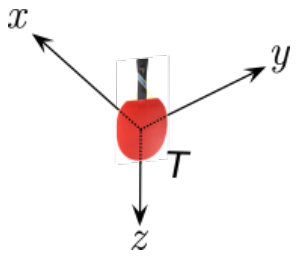


Figure 5.10: The tool coordinate system.

The values $[X,Y,Z,\alpha,\beta,\gamma]$ can be read from the KUKA controller, where X,Y,Z are the racket's 3D position and α,β,γ 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.

$$R_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \tag{5.14}$$

$$R_Y = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{5.15}$$

$$R_Z = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{5.16}$$

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

$$u_W = R_Z R_Y R_X * u_T. \tag{5.17}$$

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 angles between two norm vectors from the robot and cameras are defined as the orientation error. As shown from Fig. 5.11 , the position error is below 13 mm with an average of 7.8 mm and the orientation error is under 15.0° with 7.2° average value.

5.4.2 Comparison with Human Pose Estimation

We directly apply Convolutional Pose Machines (CPMs) (Wei *et al.*, 2016) to extract the human body keypoints including ear, eye, nose, neck, shoulder, elbow, wrist and hip (14

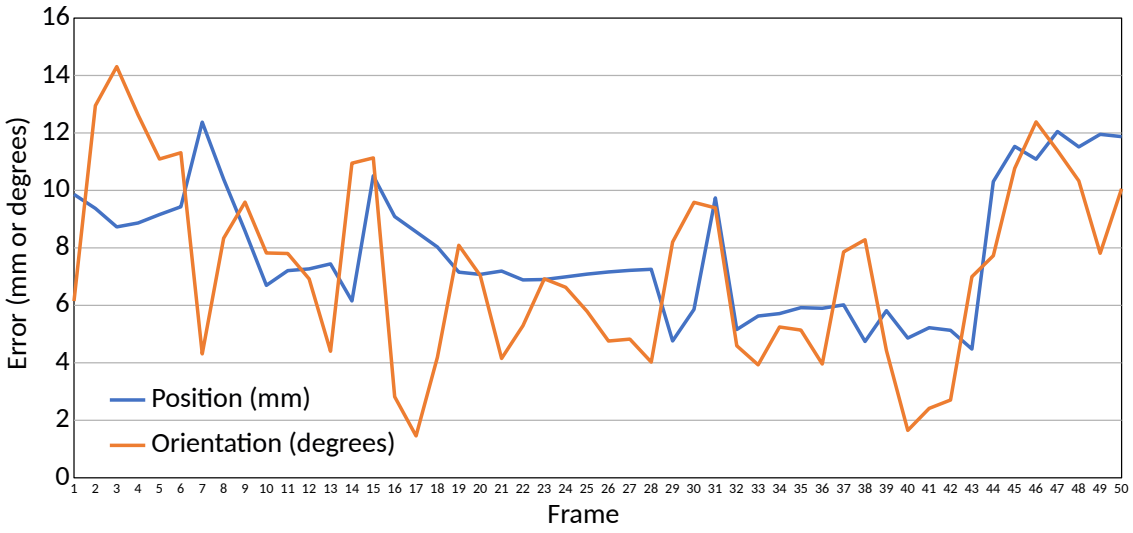


Figure 5.11: Racket pose evaluation in the robot coordinates.

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. 5.12.

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

However, CPMs have a crucial issue of requiring high-level GPU hardwares. They 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 7 ms (150 FPS) and give the opportunities to train the robot having a human-like movement.

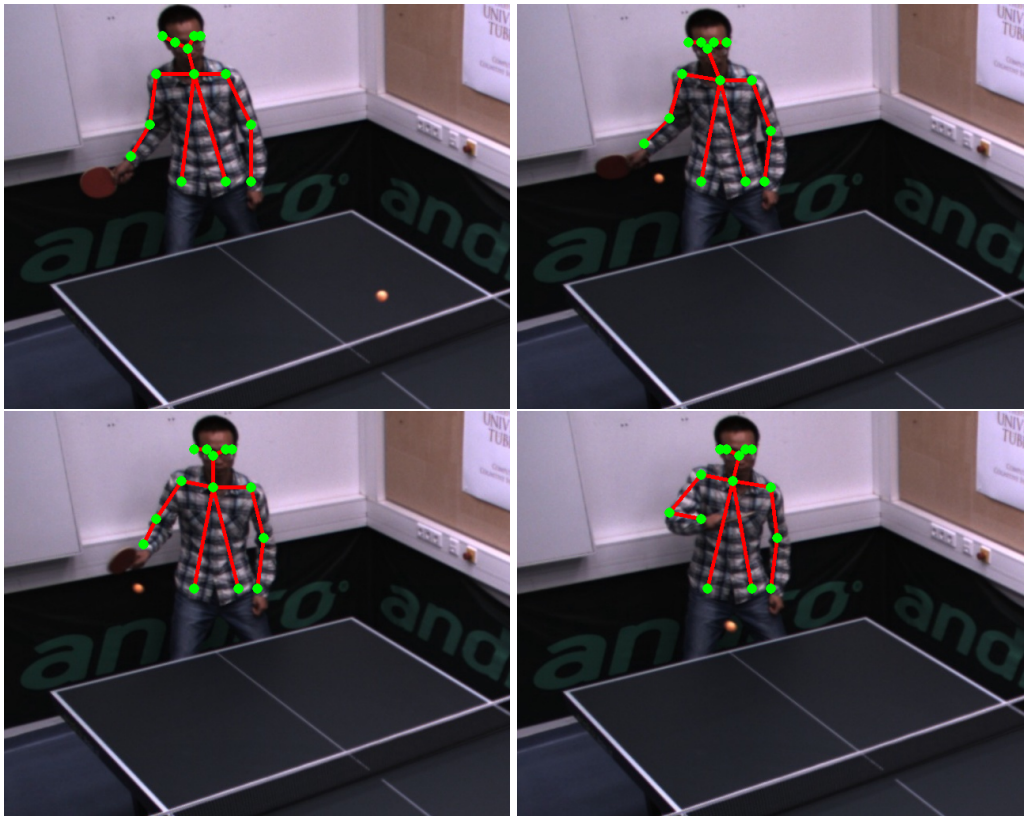


Figure 5.12: Human pose estimation in image sequences, using Convolutional Pose Machines. Human body keypoints, including two ears, two eyes, one nose, one neck, two shoulders, two elbows, two wrists and two hips (14 keypoints in total), are shown in green circles.

5.5 Conclusions

In this chapter, 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 were performed to evaluate the accuracy of pose detection and classification.

Chapter 6

Robust Stroke Recognition via Vision and IMU

Stroke recognition in table tennis is a challenging task, due to the variety of movements. Many different sensors have been adopted in robotic table tennis, with the goal of detecting the players' movements. In this chapter, 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.

Large parts of this work have been pre-published in Gao *et al.* (2021b).

6.1 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 (Cao *et al.*, 2018), hand keypoints (Simon *et al.*, 2017) or 3D human pose (Pavllo *et al.*, 2019) 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 (Zimmermann *et al.*, 2018). Motion sensor based methods adopt low-cost accelerometers, gyroscopes, and sometimes magnetometers to detect the human's linear acceleration and angular velocity (Zhao *et al.*, 2018) 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 (von Marcard *et al.*, 2018).

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 sports. An AI Coach system for athletic training (Wang *et al.*, 2019b) 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.* (2019) 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 chapter is the table tennis racket pose estimation. This gives our table tennis robot (shown in Fig. 6.1) 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 chapter 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 high 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.

6.2 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 (Xiang *et al.*, 2017) directly estimates the 6D object pose with an end-to-end network from a single image. Sundermeyer *et al.* (2018) present an implicit method for 3D orientation estimation based on Augmented Autoencoders (AAEs), which is trained on synthetic images. The 3D translation is then

Figure 6.1: 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 at the ceiling is used to capture the human player movements from above (Fig. 6.2 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.

computed according to the pinhole camera model. A pixel-wise voting network (PVNet) (Peng *et al.*, 2019) 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) (Li *et al.*, 2019) 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.* (2017) 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 any dataset available. In Chapter 5, we 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 Kawakami *et al.* (2021) 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.

6.3 Methodology

6.3.1 Overview

IMUs are widely used in wearable devices to measure human activity in real-time and with high accuracy. In this thesis, we mount a MetaMotionR (MMR) IMU (MbientLab, 2015) at the bottom of the racket handle, as shown in Fig. 6.2. With Bosch sensor fusion technology, the MMR sensor can provide robust linear accelerations and rotation angles via Bluetooth 4.0 at 100 Hz. Beange (2019) 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,

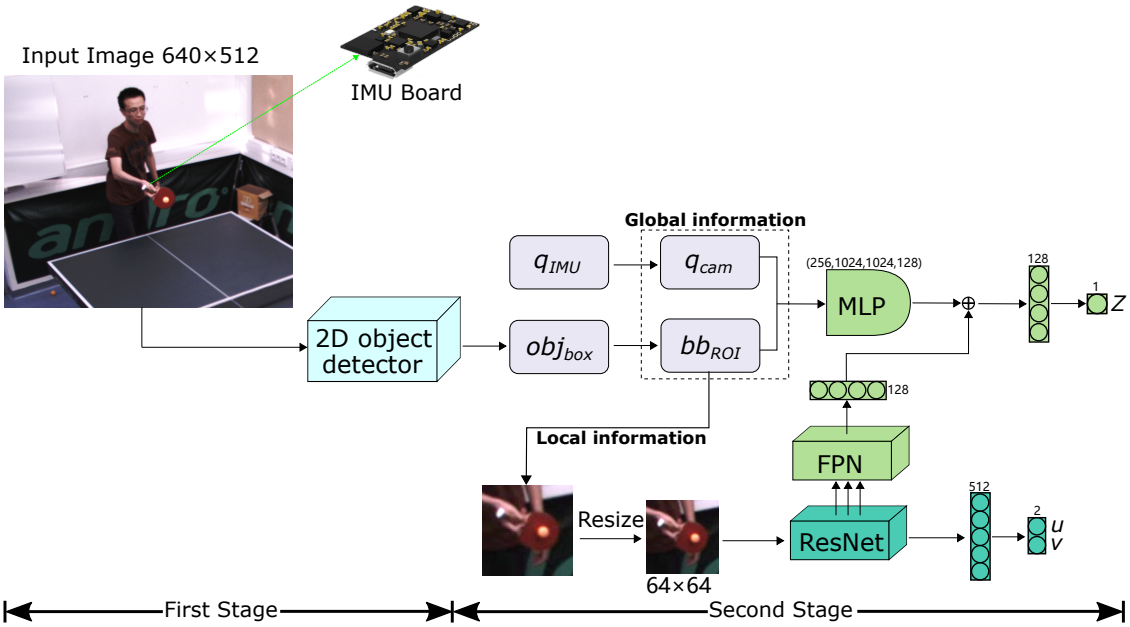


Figure 6.2: CNN architecture for the racket position estimation during testing in our scenario. The rotation q_{IMU} is read from a wireless IMU as a 4D quaternion in the IMU frame. It is transformed to the camera frame as q_{cam}. The images with 640×512 pixels are first fed into a pre-trained 2D object 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. 6.2. 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. 6.1.

as shown in Fig. 6.2. 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 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:

$$X = \frac{(x'_{min} + u - c_x)\,Z}{f_x}, \quad Y = \frac{(y'_{min} + v - c_x)\,Z}{f_y} \tag{6.1}$$

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 latter 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.

6.3.2 Racket Centroid Extraction

In order to detect the racket in images, we employ a self-pretrained YOLOv4 (Bochkovskiy *et al.*, 2020) 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:

$$\begin{cases} s &= max(h, w) \\ N &= randint(-\alpha s, \alpha s) \\ (h', w') &= (\beta s + s, h') \\ (x'_c, y'_c) &= (x_c, y_c) + N \\ (x'_{min}, y'_{min}) &= (x'_c, y'_c) - 0.5(h', w') \end{cases} \tag{6.2}$$

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. 6.2). This *DynamicResize* technique is based on the Dynamic Zoom In (DZI) in Li *et al.* (2019). In contrast to the DZI that enlarges the crops, here we simply shrink them, since the texture-

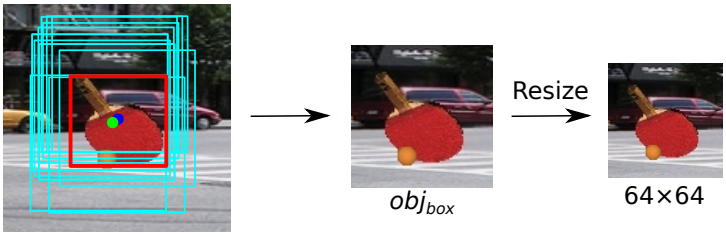


Figure 6.3: An example for *DynamicResize* during training. **Left**: The detected bounding box (red) from YOLOv4 and the dynamically computed *ROI* candidates (cyan). The bounding box center $[x_c, y_c]$ and the racket centroid $[u, v]$ are marked as blue and green circle, respectively **Middle**: the randomly selected bb_{ROI} for training. **Right**: the final resized crop.

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. 6.3. Then a ResNet18 (He *et al.*, 2016) is deployed to extract the deep features, followed by two dense layers with 512 and 2 units $[u, v]$, respectively as shown at the bottom of Fig. 6.2.

6.3.3 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, Wu *et al.* (2019) run a RetinaNet (Lin *et al.*, 2017) on the input images and concatenate 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.

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. 6.2). A Resnet-FPN network (in Fig. 6.4) 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:

$$\mathcal{L}_{\text{pos}} = \gamma_1 \cdot |Z - \hat{Z}| + \gamma_2 \cdot \left\| C_{pos} - \hat{C}_{pos} \right\|_1 \tag{6.3}$$

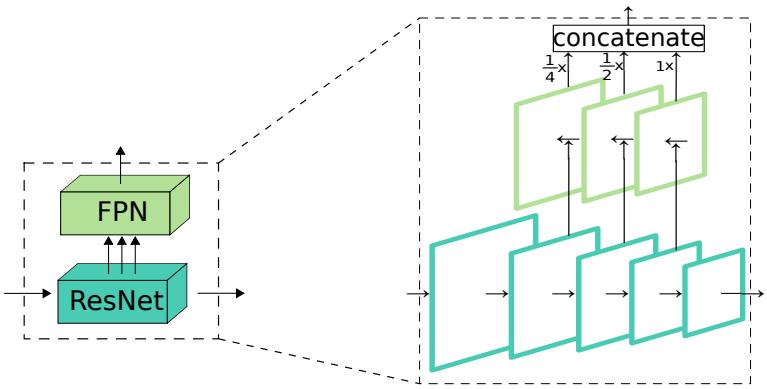


Figure 6.4: ResNet-FPN (Feature Pyramid Network).

where Z is representing the estimated and $\hat{Z}$ the 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.

6.4 Experiments

6.4.1 Dataset

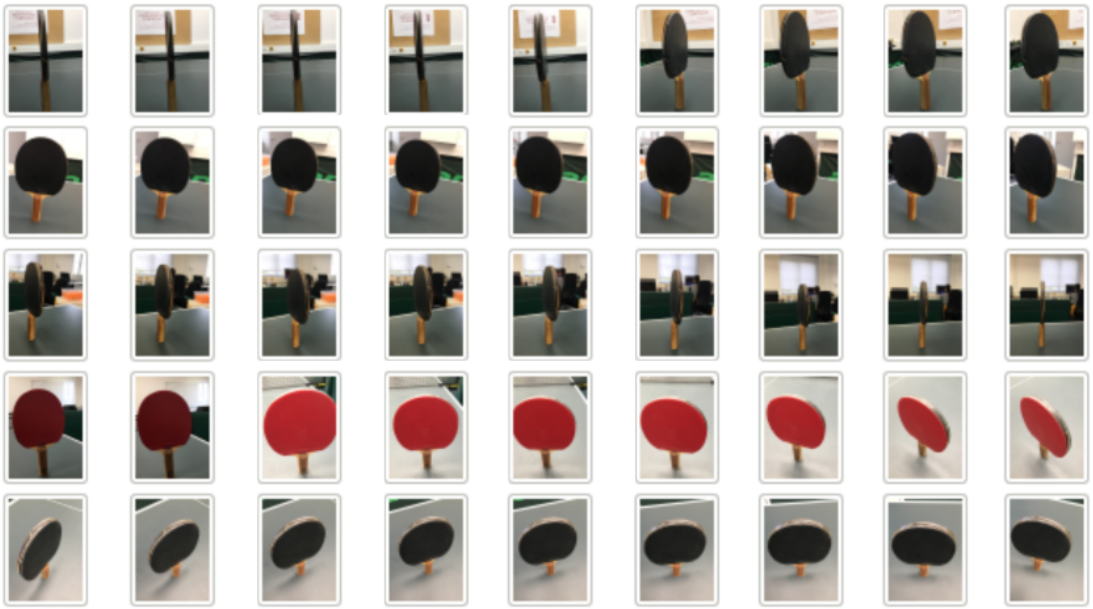


Figure 6.5: A set of image examples for reconstructing the racket CAD model by the Meshroom software (Wang and Li, 2016).

To train the proposed model, we create a synthetic dataset which can be labeled automatically. A racket CAD model is first reconstructed from a set of real racket images (see Fig. 6.5), with the free, open-source reconstruction software Meshroom (Wang and Li, 2016), based on the structure from motion (SfM) technique. This results in a reconstructed 3D mesh in Fig. 6.6 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 (Cignoni *et al.*, 2008). The final high-quality 3D model is shown in Fig. 6.6 right.

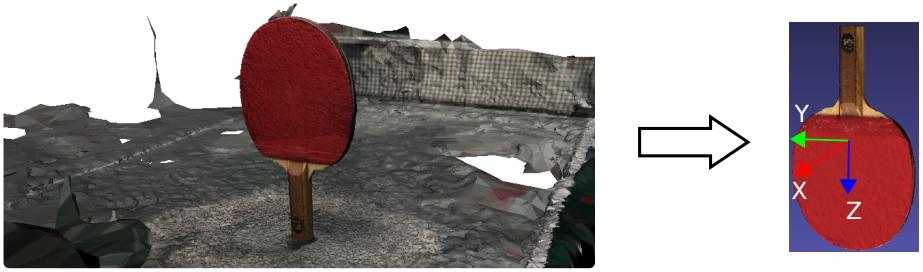


Figure 6.6: **Left**: Reconstructed mesh with background from multiple views using Meshroom software. **Right**: the final CAD model with its coordinates.

By using domain randomization (DR) (Tobin *et al.*, 2017), 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. 6.8(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 refection markers on the racket and then capturing the human player motions with an OptiTrack system. However, the markers must be placed at the surface in a critical configuration (see Fig. 6.7 left), which would result in many occlusions in images. Therefore, we decided 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 orientations, we collect an evaluation dataset of 208 images (Fig. 6.8(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. 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. 6.8(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.

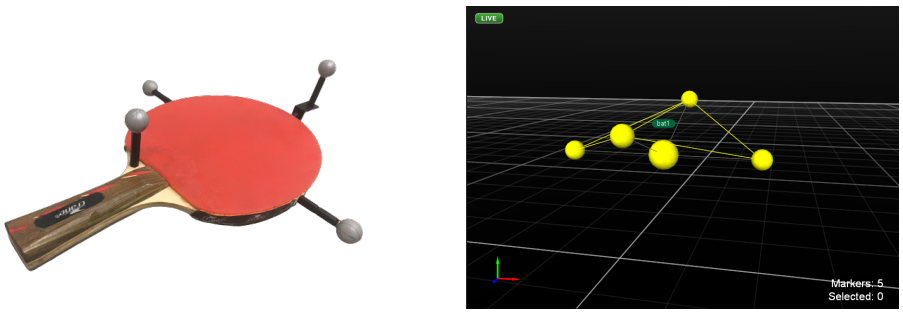
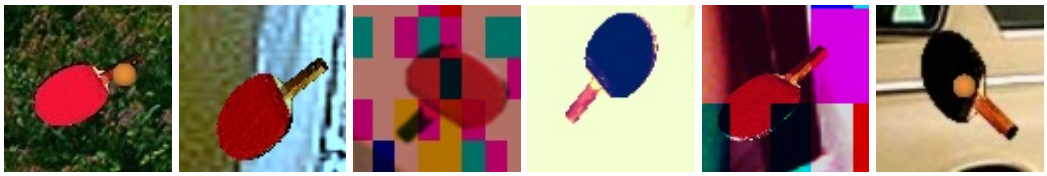
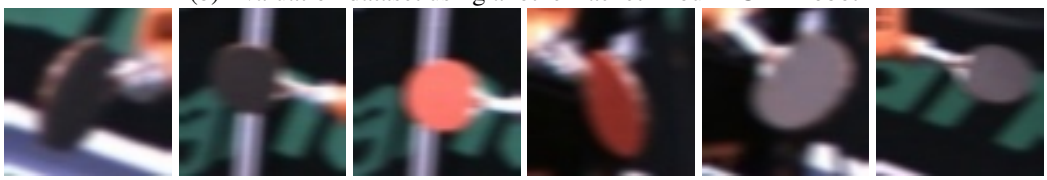


Figure 6.7: **Left**: a racket mounted with four reflection balls. **Right**: the 3D position of each ball and their pivot point in the OptiTrack system.



(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

Figure 6.8: Cropped examples for training and evaluation of the racket position estimation.

6.4.2 Training and Inference

As shown in Fig. 6.2, 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 512×512

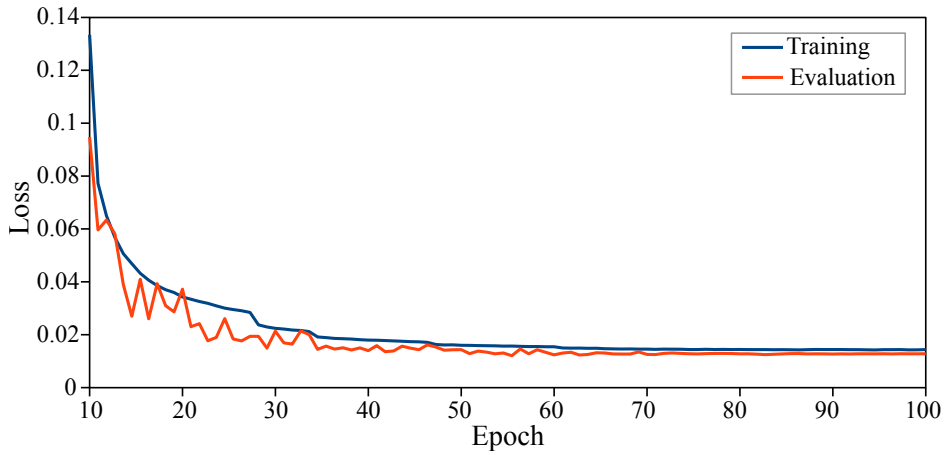


Figure 6.9: Losses for training and evaluation.

and change the trained model from darknet (Bochkovskiy *et al.*, 2020) to the tkDNN (Verucchi *et al.*, 2020) framework. The activation function used in the second stage is ReLU (Agarap, 2018). 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 6.1: 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, which is plotted in Fig. 6.9. 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.

6.4.3 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.

In Table 6.2, we compare our method with current research in which different sensors are used. Zhang *et al.* (2017) 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 Chapter 5, 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 (Sundermeyer *et al.*, 2018; Li *et al.*, 2019; Staszak and Belter, 2019). Among them, Sundermeyer *et al.* (2018) and Li *et al.* (2019) 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 Staszak and Belter (2019) have 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.

Table 6.2: Evaluation for racket position estimation.

	Sensors	E_{trans}	$\leq$5cm	$\leq$10cm
Zhang *et al.* (2017)	camera,IMU,marker	-	-	-
Sundermeyer *et al.* (2018)	single camera	39.1 cm	6.7%	17.3%
CDPNLi *et al.* (2019)	single camera	36.6 cm	7.5%	21.8%
Staszak and Belter (2019)	single camera	23.5 cm	10.6%	25.0%
Ours* (Chapter 5)	stereo cameras	2.8 cm	91.8%	100.0%
Ours (no FPN)	single camera, IMU	6.8 cm	48.6%	85.1%
Ours (with motion blur)		5.2 cm	60.1%	93.2%
Ours		**4.7 cm**	**65.0%**	**95.5%**

Furthermore, two additional experiments, with motion blur (in Fig. 6.8(c)) and without FPN layers, are performed to simulate a moving racket and do an ablation study, respectively. Fig. 6.10-6.13 shows four examples with different movements. The CAD model of the racket is reprojected into images with green contour to visualize the 6D pose. 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`

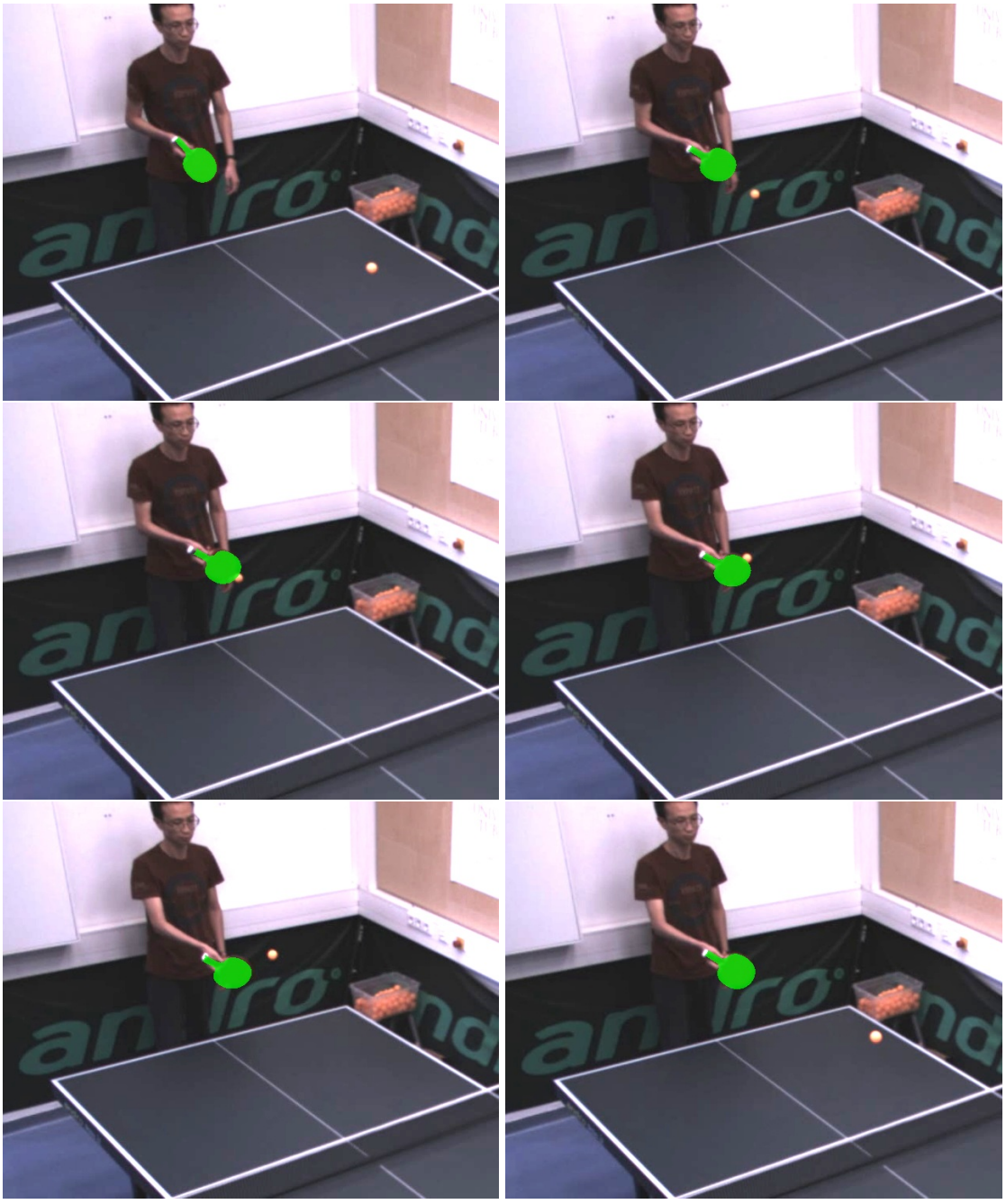


Figure 6.10: Counter Hit.

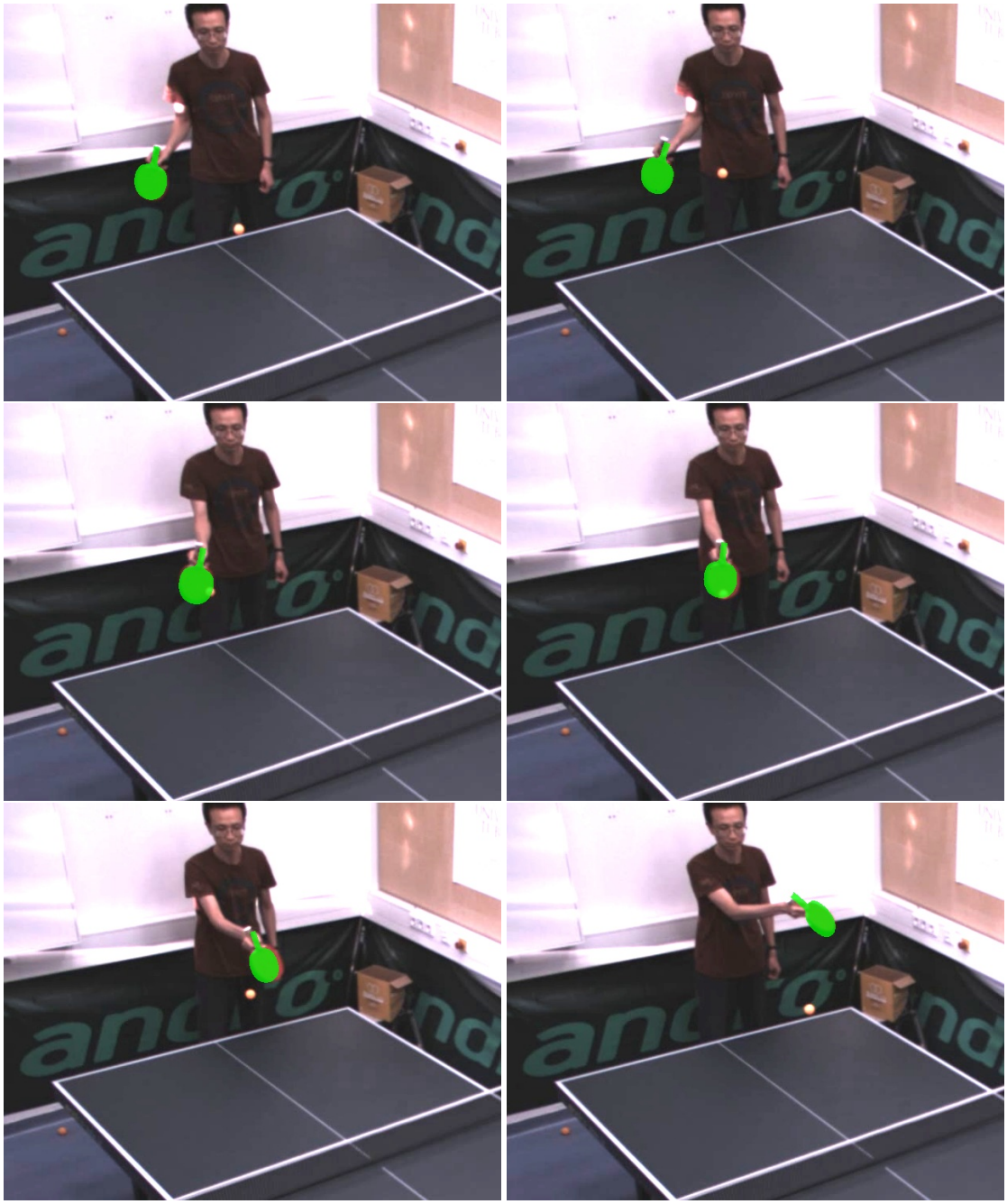


Figure 6.11: Side Spin.

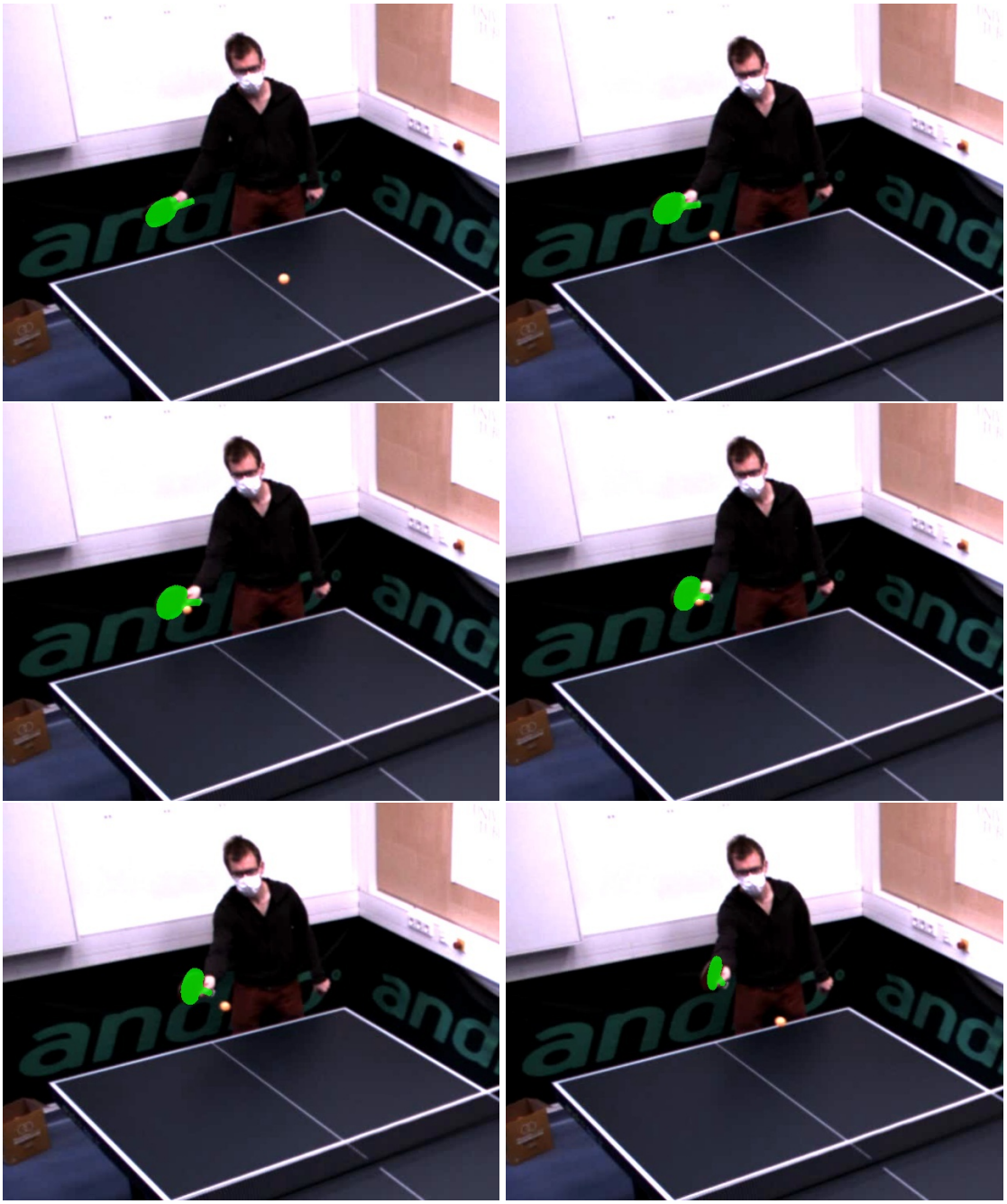


Figure 6.12: Back Spin.

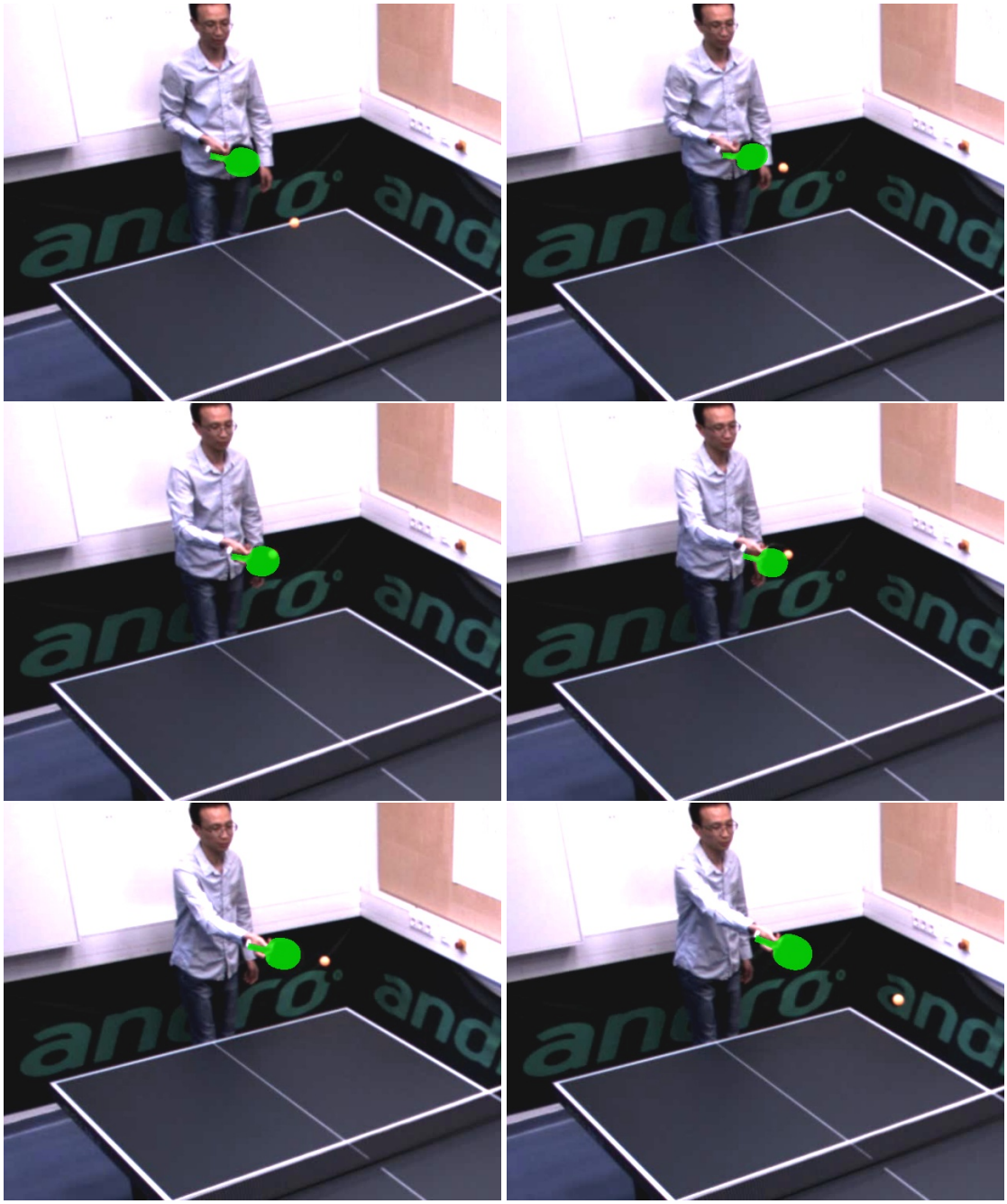


Figure 6.13: Counter Hit with the black side.

6.5 Conclusions

In this chapter, 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 that 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.

Chapter 7

Conclusions

7.1 Summary

This thesis focuses on sensor fusion and stroke learning for robotic table tennis. The study is composed of ball 3D position estimation, robot stroke learning, and racket 6D pose detection, as well as corresponding datasets for training and evaluation. Each part could be run in real-time, which is necessary for robotic table tennis. Different sensors, including an IMU and four cameras, were utilized for estimating the object pose.

In Chapter 3, we developed a calibration approach for multiple stationary cameras. The initial intrinsic and extrinsic matrices for cameras were first calculated by classic calibration methods with a circle grid pattern. The calibration error tested against balls mounted on the KUKA robot were 11.0 mm for stereo cameras and 15.0 mm for four cameras. To improve it, we developed a new loss function and post-optimized the extrinsic transformations simultaneously by capturing a new set of pattern images from each camera. The final accuracy was 3.2 mm for stereo cameras and 2.5 mm for four cameras. Two approaches, traditional image processing and deep learning, were proposed for ball 2D pixel position extraction. The former one achieved higher mean IoU scores and lower ball center error in pixels than the latter one, but a slower inference speed (1.51 ms vs 0.67 ms) when testing in our manually labeled dataset.

In Chapter 4, we proposed an optimal stroke learning approach for teaching the robot to play table tennis. A realistic Gazebo simulation environment was built for the ball's dynamics and the robot's kinematics, which included three entities: ball, table, and robot. We decomposed the learning strategy into two stages: ball hitting state prediction and optimal stroke learning, on which we mainly focused in this thesis. Based on the controllable and applicable actions of our robot, a multi-dimensional reward function and Q-value model were proposed. The comparison with other RL methods was performed using an evaluation dataset of 1000 balls in simulation. An efficient retraining approach was proposed to close the sim-to-real gap. The testing experiments in reality showed that the robot could successfully return the ball to the desired target with an error of around 24.9 cm and a success rate of 98% in three different scenarios: one using a ball throwing machine and two with human players.

In Chapter 5, a stereo-vision based approach was proposed for the racket 6D pose

detection, which could help the robot to imitate the movements of a human player and recognize the ball trajectory better. By using two monocular cameras, we could extract feature points from the racket's contour in image coordinates. With the 3D geometrical information of a racket, a wide baseline stereo matching method was 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 was adopted to track the racket pose, and a neural network with two hidden layers was used to classify the pose movements. We conducted two experiments to evaluate the accuracy of racket pose detection and stroke classification. The average error in position and orientation was around 7.8 mm and 7.2° compared with the ground truth from a KUKA robot. Using our racket pose as features for an MLP, the classification accuracy for five stroke types is 98%, the same as the human pose features from Convolutional Pose Machines (CPMs).

In Chapter 6, we proposed a two-stage approach to directly recognize the table tennis racket's movement by sensor fusion of an IMU and a camera. A bounding box around the racket could be extracted from an RGB image in the first stage. An efficient and lightweight CNN architecture was 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 was available at a frame rate of 100 Hz. In the experiments, two datasets were 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. Finally, after transforming the racket trajectories from the human to the robot side, we performed one behavior cloning experiment on the robot. This gave the potential to directly mimic the human trajectory and avoid path planning with additional libraries.

7.2 Future Work

Although we have achieved some successes in robotic table tennis, there are still many challenges and limitations, which are open and worth exploring.

One limitation is the fixed hitting position along the X axis ($X = 676$ cm in the robot coordinates), which is not the usual way by which a human player hits the ball. A dynamic hitting position is necessary to improve the robot's performance, which can be predicted or learnt using RL in simulation. In addition, the predicted velocity and spin of the ball at the hitting position are not visible directly by the cameras, which means the ball's state (position, velocity, and spin) is only partially observable. This could be modeled as a partially observable Markov decision process (POMDP). It is a generalization of a Markov decision process (MDP). Therefore, instead of using MDPs like in this work, we can also treat this RL problem as POMDPs which can make use of the whole trajectory of the ball as states to improve the learning strategy.

Another limitation is that the proposed two-stage approach for racket detection is failing 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 back-

ground, as shown in the demo video. In this case, we could utilize tracking methods to identify coherent relations between frames. After knowing the racket pose (action) and the corresponding ball position (state), we could directly initialize the RL policy from reality rather than simulation. Also, the racket pose can help to predict the ball's future state more precisely.

Futhermore, our robot has difficulties with back spin balls, since a wrist singularity occurs when the axes of joints 4 and 6 become coincident. This can cause these joints to try to rotate 180 degrees instantaneously. To solve this problem, we could manually add some joint constraints in order to avoid the robot moving to the singularity position. This means we should optimize the robot path in the joint space in an efficient way, rather than directly use the Reflexxes library in the Cartesian space.

Symbols

m	mass of the table tennis ball
g	gravitational acceleration
F_d	air drag
F_g	gravitation
F_m	Magnus force
C_D	drag coefficient
C_M	lift coefficient
ρ_a	air density
s	state
r	reward
p	position
v	linear velocity
ω	angular velocity
h	height
w	width
θ	model weight
ϕ	model weight
$\mathcal{D}$	replay buffer
$\mathcal{B}$	minibatch
$\mathcal{N}$	Gaussian noise
H	homography matrix
X	axis or coordinate in 3D
Y	axis or coordinate in 3D
Z	axis or coordinate in 3D

Abbreviations

2D	Two dimensional
3D	Three dimensional
6D	Six dimensional
ADAM	adaptive moment estimation
CAD	Computer-aided Design
CNN	Convolutional Neural Network
COCO	Common Objects in Context
DDPG	Deep Deterministic Policy Gradient
DNN	Deep Neural Network
DoF	Degree of Freedom
EKF	Extended Kalman Filter
FCN	Fully Convolutional Network
FPS	Frame per Second
IMU	Inertial Measurement Unit
LM	Levenberg Marquardt
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
NMS	Non-maximum Suppression
ODE	Open Dynamics Engine
PnP	Perspective-n-Point
PPO	Proximal Policy Optimization Depth
RANSAC	RANdom Sample Consensus
R-CNN	Region Based Convolutional Neural Networks
ReLU	Rectified Linear Unit
RGB	Red Green Blue
RGBD	Red Green Blue Depth
RL	Reinforcement Learning
ROI	Region of Interest
ROS	Robot Operating System
SAC	Soft Actor Critic
SFM	Structure from Motion
SPG	Stochastic Policy Gradient
SSD	Single Shot MultiBox Detector
SVM	Support Vector Machine

TD3	Twin Delayed Deep Deterministic Policy Gradient
VGG	Visual Geometry Group
VOC	Visual Object Classes
YOLO	You Only Look Once (a special CNN network architecture)

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. `https://www.tensorflow.org`.

Agarap, A. F. (2018). Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*.

Agarwal, S., Mierle, K., and Others (2012). Ceres solver. `http://ceres-solver.org`.

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., *et al.* (2020). Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, **39**(1), 3–20.

Arun, K. S., Huang, T. S., and Blostein, S. D. (1987). Least-squares fitting of two 3-d point sets. *IEEE Transactions on pattern analysis and machine intelligence*, (5), 698–700.

Asai, K., Nakayama, M., and Yase, S. (2019). The ping pong robot to return a ball precisely. `https://www.omron.com/global/en/technology/omrontechnics/vol51/016.html`.

Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, **39**(12), 2481–2495.

Beange, K. (2019). *Validation of Wearable Sensor Performance and Placement for the Evaluation of Spine Movement Quality*. Ph.D. thesis, University of Ottawa.

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

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., *et al.* (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Blank, P., Hoßbach, J., Schuldhaus, D., and Eskofier, B. M. (2015). Sensor-based stroke detection and stroke type classification in table tennis. In *Proceedings of the 2015 ACM International Symposium on Wearable Computers*, pages 93–100.

Blank, P., Groh, B. H., and Eskofier, B. M. (2017). 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*, pages 2–9.

Bochkovskiy, A., Wang, C.-Y., and Liao, H.-Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.

BostonDynamics (2013). Atlas. `https://www.bostondynamics.com/atlas`.

Brachmann, E., Krull, A., Michel, F., Gumhold, S., Shotton, J., and Rother, C. (2014). Learning 6d object pose estimation using 3d object coordinates. In *European conference on computer vision*, pages 536–551. Springer.

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

Bridgeman, L., Volino, M., Guillemaut, J.-Y., and Hilton, A. (2019). Multi-person 3d pose estimation and tracking in sports. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Büchler, D., Guist, S., Calandra, R., Berenz, V., Schölkopf, B., and Peters, J. (2020). Learning to play table tennis from scratch using muscular robots. *arXiv preprint arXiv:2006.05935*.

Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E., and Sheikh, Y. (2018). OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. In *arXiv preprint arXiv:1812.08008*.

Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E., and Sheikh, Y. (2019). Openpose: realtime multi-person 2d pose estimation using part affinity fields. *IEEE transactions on pattern analysis and machine intelligence*, **43**(1), 172–186.

Chen, G., Xu, D., Fang, Z., Jiang, Z., and Tan, M. (2013). Visual measurement of the racket trajectory in spinning ball striking for table tennis player. *IEEE Transactions on Instrumentation and Measurement*, **62**(11), 2901–2911.

Chu, W.-T. and Situmeang, S. (2017). Badminton video analysis based on spatiotemporal and stroke features. In *Proceedings of the 2017 ACM on international conference on multimedia retrieval*, pages 448–451.

Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G. (2008). Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference*, volume 2008, pages 129–136.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223.

Duan, K., Bai, S., Xie, L., Qi, H., Huang, Q., and Tian, Q. (2019). Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6569–6578.

Elliott, N., Choppin, S., Goodwill, S., Senior, T., Hart, J., and Allen, T. (2018). Single view silhouette fitting techniques for estimating tennis racket position. *Sports Engineering*, **21**(2), 137–147.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2012). The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, **24**(6), 381–395.

Fryer, J. G. and Brown, D. C. (1986). Lens distortion for close-range photogrammetry. *Photogrammetric engineering and remote sensing*, **52**(1), 51–58.

Fu, J., Liu, J., Tian, H., Li, Y., Bao, Y., Fang, Z., and Lu, H. (2019). Dual attention network for scene segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3146–3154.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR.

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. IEEE.

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**, 23–39.

Gao, Y., Tebbe, J., and Zell, A. (2021a). Optimal stroke learning with policy gradient approach for robotic table tennis. *arXiv preprint arXiv:2109.03100*.

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

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969.

He, Y., Sun, W., Huang, H., Liu, J., Fan, H., and Sun, J. (2020). Pvn3d: A deep point-wise 3d keypoints voting network for 6dof pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11632–11641.

Heikkila, J. and Silvén, O. (1997). A four-step camera calibration procedure with implicit image correction. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition*, pages 1106–1112. IEEE.

Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., and Navab, N. (2012). Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian conference on computer vision*, pages 548–562. Springer.

Hirschmuller, H. (2005). 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)*, volume 2, pages 807–814. IEEE.

Hodan, T., Michel, F., Brachmann, E., Kehl, W., GlentBuch, A., Kraft, D., Drost, B., Vidal, J., Ihrke, S., Zabulis, X., *et al.* (2018). Bop: Benchmark for 6d object pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34.

Huang, Y., Schölkopf, B., and Peters, J. (2015). Learning optimal striking points for a ping-pong playing robot. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4587–4592. IEEE.

Kaehler, A. and Bradski, G. (2016). *Learning OpenCV 3: computer vision in C++ with the OpenCV library*. " O'Reilly Media, Inc.".

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., *et al.* (2018). Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR.

Kalra, A., Taamazyan, V., Rao, S. K., Venkataraman, K., Raskar, R., and Kadambi, A. (2020). Deep polarization cues for transparent object segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8602–8611.

Kawakami, S., Ikumo, M., and Oyaas, T. (2021). Omron table tennis robot forpheus. `https://www.omron.com/innovation/forpheus.htmls`.

Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.-M., Lam, V.-D., Bewley, A., and Shah, A. (2019). Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., and Matsubara, H. (1997). Robocup: A challenge problem for ai. *AI magazine*, **18**(1), 73–73.

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. (2010). *On-Line Trajectory Generation in Robotic Systems: Basic Concepts for Instantaneous Reactions to Unforeseen (Sensor) Events*, volume 58. Springer.

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. IEEE.

Kyrarini, M., Haseeb, M. A., Ristić-Durrant, D., and Gräser, A. (2019). Robot learning of industrial assembly task via human demonstrations. *Autonomous Robots*, **43**(1), 239–257.

Lampert, C. H. and Peters, J. (2012). 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.

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

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. IEEE.

Li, Z., Wang, G., and Ji, X. (2019). 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*, pages 7678–7687.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988.

Liu, C., Hayakawa, Y., and Nakashima, A. (2013). Racket control for a table tennis robot to return a ball. *SICE Journal of Control, Measurement, and System Integration*, **6**(4), 259–266.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.

Mahjourian, R., Miikkulainen, R., Lazic, N., Levine, S., and Jaitly, N. (2018). Hierarchical policy design for sample-efficient learning of robot table tennis through self-play. *arXiv preprint arXiv:1811.12927*.

Manovich, N. (2018). Computer vision annotation tool (cvat). `https://github.com/openvinotoolkit/cvats`.

MbientLab (2015). mbientlab. `https://mbientlab.com`.

Micusik, B. and Wildenauer, H. (2015). Descriptor free visual indoor localization with line segments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3165–3173.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

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

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

Noh, H., Hong, S., and Han, B. (2015). Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528.

Ohya, H. and Saito, H. (2004). Tracking racket face in tennis serve motion using high-speed multiple cameras. *CiteSeer, unpublished*.

OpenAI (2018). Intro to policy optimization. `https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html`.

Pavllo, D., Feichtenhofer, C., Grangier, D., and Auli, M. (2019). 3d human pose estimation in video with temporal convolutions and semi-supervised training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7753–7762.

Peng, S., Liu, Y., Huang, Q., Zhou, X., and Bao, H. (2019). Pvnet: Pixel-wise voting network for 6dof pose estimation. In *CVPR*.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE.

Potmesil, M. and Chakravarty, I. (1982). Synthetic image generation with a lens and aperture camera model. *ACM Transactions on Graphics (TOG)*, **1**(2), 85–108.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660.

Rad, M. and Lepetit, V. (2017). Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3828–3836.

Redmon, J. and Farhadi, A. (2017). Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271.

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

SICK (2020). S300 safety laser scanner. `https://cdn.sick.com/media/docs/3/13/613/operating_instructions_s300_safety_laser_scanner_en_im0017613.pdf`.

Silva, R., Melo, F. S., and Veloso, M. (2015). Towards table tennis with a quadrotor autonomous learning robot and onboard vision. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 649–655. IEEE.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., *et al.* (2016). Mastering the game of go with deep neural networks and tree search. *nature*, **529**(7587), 484–489.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., *et al.* (2017). Mastering the game of go without human knowledge. *nature*, **550**(7676), 354–359.

Simon, T., Joo, H., Matthews, I., and Sheikh, Y. (2017). Hand keypoint detection in single images using multiview bootstrapping. In *CVPR*.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Song, C., Song, J., and Huang, Q. (2020). Hybridpose: 6d object pose estimation under hybrid representations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 431–440.

Srivastava, R., Patwari, A., Kumar, S., Mishra, G., Kaligounder, L., and Sinha, P. (2015). Efficient characterization of tennis shots and game analysis using wearable sensors data. In *2015 IEEE sensors*, pages 1–4. IEEE.

Staszak, R. and Belter, D. (2019). Hybrid 6d object pose estimation from the rgb image. In *ICINCO (1)*, pages 541–549.

Sundermeyer, M., Marton, Z.-C., Durner, M., Brucker, M., and Triebel, R. (2018). Implicit 3d orientation learning for 6d object detection from rgb images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 699–715.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tareen, S. A. K. and Saleem, Z. (2018). A comparative analysis of sift, surf, kaze, akaze, orb, and brisk. In *2018 International conference on computing, mathematics and engineering technologies (iCoMET)*, pages 1–10. IEEE.

Tebbe, J., Gao, Y., Sastre-Rienietz, M., and Zell, A. (2018). A table tennis robot system using an industrial kuka robot arm. In *German Conference on Pattern Recognition*, pages 33–45. Springer.

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. IEEE.

Tekin, B., Sinha, S. N., and Fua, P. (2018). Real-time seamless single shot 6d object pose prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 292–301.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). 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)*, pages 23–30. IEEE.

Toyota (2019). The development diary of cue, the ai basketball robot. `https://global.toyota/en/newsroom/corporate/28587721.html`.

Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., and Birchfield, S. (2018). Deep object pose estimation for semantic robotic grasping of household objects. *arXiv preprint arXiv:1809.10790*.

Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. (1999). Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer.

Tulyakov, S., Ivanov, A., and Fleuret, F. (2018). Practical deep stereo (pds): Toward applications-friendly deep stereo matching. *arXiv preprint arXiv:1806.01677*.

UESTC (2016). Robot badminton battle at south china sports festival. `https://www.youtube.com/watch?v=qZk_j4D1SjU`.

Uijlings, J. R., Van De Sande, K. E., Gevers, T., and Smeulders, A. W. (2013). Selective search for object recognition. *International journal of computer vision*, **104**(2), 154–171.

Verucchi, M., Bartoli, L., Bagni, F., Gatti, F., Burgio, P., and Bertogna, M. (2020). Real-time clustering and lidar-camera fusion on embedded platforms for self-driving cars. In *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, pages 398–405. IEEE.

von Marcard, T., Henschel, R., Black, M. J., Rosenhahn, B., and Pons-Moll, G. (2018). Recovering accurate 3d human pose in the wild using imus and a moving camera. In

Proceedings of the European Conference on Computer Vision (ECCV), pages 601–617.

Wang, C., Xu, D., Zhu, Y., Martín-Martín, R., Lu, C., Fei-Fei, L., and Savarese, S. (2019a). Densefusion: 6d object pose estimation by iterative dense fusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3343–3352.

Wang, J., Qiu, K., Peng, H., Fu, J., and Zhu, J. (2019b). Ai coach: Deep human pose estimation and analysis for personalized athletic training assistance. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 374–382.

Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., *et al.* (2020). Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*.

Wang, Z. and Li, J.-Z. (2016). Text-enhanced representation learning for knowledge graph. In *Ijcai*, pages 1293–1299.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, **8**(3-4), 279–292.

Wei, S.-E., Ramakrishna, V., Kanade, T., and Sheikh, Y. (2016). Convolutional pose machines. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 4724–4732.

Wu, D., Zhuang, Z., Xiang, C., Zou, W., and Li, X. (2019). 6d-vnet: End-to-end 6-dof vehicle pose estimation from monocular rgb images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0.

Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. (2017). Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*.

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.

Xu, T., Liu, Q., Zhao, L., and Peng, J. (2018). Learning to explore via meta-policy gradient. In *International Conference on Machine Learning*, pages 5463–5472. PMLR.

Zhang, H., Wu, Y., and Yang, F. (2009). Ball detection based on color information and hough transform. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, volume 2, pages 393–397. IEEE.

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

Zhang, R., Tsai, P.-S., Cryer, J. E., and Shah, M. (1999). Shape-from-shading: a survey. *IEEE transactions on pattern analysis and machine intelligence*, **21**(8), 690–706.

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. IEEE.

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, 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., Yang, R., Chevalier, G., Xu, X., and Zhang, Z. (2018). Deep residual bidir-lstm for human activity recognition using wearable sensors. *Mathematical Problems in Engineering*, **2018**.

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. IEEE.

Zimmermann, C., Welschehold, T., Dornhege, C., Burgard, W., and Brox, T. (2018). 3d human pose estimation in rgbd images for robotic task learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1986–1992. IEEE.