CS 549 & CS 449 Learning for Robotics Final Report

Süleyman Yağız Başaran
Bilkent University Computer Science
Ankara, Türkiye
yagiz.basaran@ug.bilkent.edu.tr

Bartu Özyıldırım
Bilkent University Computer Sciennce
Ankara, Türkiye
bartu.ozyildirim@ug.bilkent.edu.tr

Mert Can Dağdelen
Bilkent University Electrics and Electronics Engineering
Ankara, Türkiye
mert.dagdelen@ug.bilkent.edu.tr

This project develops a robotic system for a precision ring toss game, emphasizing trajectory optimization, ring grasp handling, and controlled throwing motion. The system combines physics-based calculations for predefined environments and CNN-based parameter generation for adaptability in varying conditions. To improve accuracy, the system adjusts its motion strategy to compensate for positional deviations introduced during grasping, using fine-tuned stretching motions and release controls. A machine learning model, trained on a diverse dataset of sample throws, generates optimized parameters to enhance the likelihood of a successful action. Additionally, a perception module captures point clouds of the environment to estimate distances by referencing the ring size.

I. Introduction

This project aims to design and simulate a robot capable of performing a ring toss game. The robot's primary tasks are to detect the environment, securely hold the ring, and execute a controlled throwing motion so the ring lands encircling the target. In a known environment, the robot will use predefined physics parameters for a precise throw. In variable or unknown environments, machine learning-based perception enables the robot to adjust its motion by analyzing environmental data. The inputs include sensor data from perception modules, while the outputs are calibrated arm movements that enhance the chances of a successful throw. This approach demonstrates the robot's potential to adapt its motion dynamically for enhanced precision and flexibility across various conditions.

This project addresses three main tasks: securely holding the ring, optimize the trajectory path for throwing and executing a controlled throw to encircle the target. To achieve these, motion planning with KOMO (K-order Path Markov Optimization) is employed for precise trajectory calculations, optimizing the robot's release angle, speed, and position for an accurate throw. In unknown environments, deep learning algorithms using PyTorch with multiple sample data which enables the robot to detect and locate the target stick, providing spatial data essential for targeting can be used.

Additionally, a physics engine simulates real-world dynamics such as gravity (g) and inertia, ensuring that the throw reflects realistic conditions. Together, these algorithms allow the robot to adapt its perception and movements to meet the demands of the ring toss game.

Tools: RAI Library (KOMO for motion planning) ,Physics Engine, Deep Learning Framework (PyTorch-CNN), Open3D Library.

II. Related work

Robotic throwing has emerged as a significant research area due to its potential to improve the efficiency and adaptability of robotic systems in both structured and unstructured environments. Traditional approaches to throwing tasks were often rooted in analytical models. Early work focused on crafting predefined dynamics models and tuning control parameters for specific object types such as balls or darts. For example, research by Hu et al. [5] used a stereo vision system for ball-throwing tasks, while Gai et al. [6] derived analytical methods for flexible-link manipulators.

Learning-based approaches have shown improved generalization by training neural networks to directly predict optimal control parameters. Works such as TossingBot [4] employed residual physics—a hybrid approach where neural networks learn corrections on top of physics-based predictions—to enable accurate and adaptable object throws. This method significantly reduced the error from unmodeled dynamics and performed well even in unstructured settings. Similarly, Kasaei et al. [2] proposed a deep reinforcement learning (DRL) approach that enabled object throwing into moving baskets while avoiding obstacles.

In addition to end-to-end learning, some studies introduced grasp-aware throwing frameworks, where grasping and throwing were co-optimized. These systems demonstrated that the initial grasp significantly affects the throwing trajectory, requiring precise integration between grasping and throwing modules for improved performance. Werner et al. [3] presented a dynamic throwing controller for hydraulic machines, which leveraged the dynamics of passive joints for highly efficient material handling.

Further developments in perception modules have enhanced robotic awareness of the environment, enabling the identification of dynamic factors such as moving targets and obstacle positions. For instance, Zeng et al.'s [4] perception network used RGB-D images and CNNs for accurate trajectory prediction, while Werner et al. [3] leveraged IMU data to estimate gripper dynamics. These advances allow for a richer understanding of the scene and contribute to better trajectory adjustments.

This project builds upon these works by integrating CNN-based trajectory prediction and a physics-based planner, addressing limitations such as over-reliance on simulation or handcrafted models. Additionally, the use of extensive trial

datasets in our work enhances adaptability to unknown target distances.

III. METHODOLOGY

1. Grasping Logic:

Grasping the ring involves equipping the robot's end-effector with a robust and adaptable gripping mechanism to securely hold the object while allowing fine adjustments for stabilization. By utilizing KOMO (K-order Path Markov Optimization), the robot can determine the optimal path to approach and grasp the ring, ensuring the motion is smooth, collision-free, and respects kinematic constraints.

2. Throwing Logic:

The basic throwing motion involves coordinating the robot's arm and wrist to generate the necessary velocity and angle for the throw. This motion typically includes a smooth acceleration phase, where the arm builds up momentum, followed by a precise release at the optimal point. The robot must account for the ring's weight, center of mass, and the desired trajectory while ensuring the motion is controlled and repeatable. Proper tuning of joint angles, velocities, and torques is critical to achieving an effective throwing motion.

3. Projectile Motion:

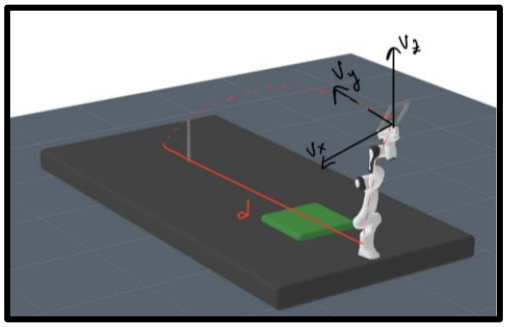

Figure 1: Projectile Motion calculations

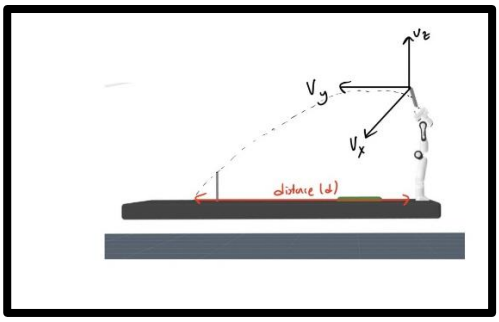

Figure 2: Projectile Motion second calculations

Equations calculate the velocity components v_x, v , and v_z for a projectile motion.

$$v_x = x_{tgt} - x_{rel} / t_{flight}$$

$$v_y = y_{tgt} - y_{rel} / t_{flight}$$

$$v_z = z_{tgt} - z_{rel} + 0.5 * g * t_{flight}^2 / t_{flight}$$

4. Feasible Release Position Calculation:

This involves selecting the interval of possible release points $[y_m , y_{max}]$ based on the robot's path and reachability. The release position Prelease = [xrelease, yrelease, zrelease] is chosen to minimize the effort velocity

$$effort = \|v_{release}\| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

5. Deciding on variables:

Determines the duration of flight for the projectile motion based on the release position and target.
q[2]: Stretching position of the robot in y-plane in the world frame.
t_1 and t_2: How fast the robot executes the throw path.
$h_{release}$: The height that robot releases the item.

6. Robot Arm Strech and Path Planning:

Our robot's base is located at: (0 1.8 .05)
Our robot stretch towards:

```
q = bot.get_qHome()
  q[2] = q[2] - 0.31
  q[3] = q[3] + 4
```

q[2] - 0.31 line comes from the grip angle of the ring.

7. Target Position and Velocity Adjustment:

Table 1: Projectile Motion Velocities

Target Position	v_x	v_y	v_z
[3.0,2.5,1.5]	3.0	0.90	5.405
[4.0,2.0,2.0]	4.0	0.40	5.905
[2.5,3.0,1.8]	2.5	1.40	5.705
[3.5,1.5,1.2]	3.5	-0.10	5.105
[4.5,2.7,2.5]	4.5	1.10	6.405

For flight time, 1 second and release point (0,1.8, 1) calculated with projectile motion formulas. We found a feasible release position and its velocity from our stretch point and we calculated the path towards release point using KOMO and calculated the speed parameter. Then by setting up the path and stime interval of the botOp accordingly to hit our target. Afterward by trial and error method we make minimal changes on our calculations to have a working code.

8. Deep Learning:

First, it is realized that the variables are almost randomly affecting the success rate. After trial and error, some variables ranges has been decided to randomly generated. The ranges for random data generation is decided according to the relatively successful numbers from before with the help of trajectory optimizing. 1000 data frames are collected by running the simulation and saving the data to .csv files. Success is measured and saved in these .csv files as this;

minimum distance of each parts of the frame should be less than our equal to some number at a specific height. Specific height is chosen as stick height, and the threshold is chosen as 0.1 which is half of the rings diameter. Then, these data frames are fed into a 2 different models to predict best possible combination of variables. The first model is a simple 1 layer CNN with a linear layer at the end. This model tries to catch the patter between our variables and the success. For the second model, success rate functions possibility of not being calculated properly or not mirroring the true success rate, the aim is changed to minimizing final distance of the ring to stick, as well as the distance at the specific height. This model is multi output MLP. This a MLP for multi output regression. This is consisting of 3 linear layers with a single activation function at the end. First model's accuracy was %80, the second model was %70. However, second model was giving unrelated results to the real world, which failed. The first model gives 10 different data points with highest probability of success as in the Table 2.

Conv1D Layer (1x3, kernel size = 3)
ReLu activation function
Linear Layer

Figure 3: Structure of CNN Model

9. Perception Module

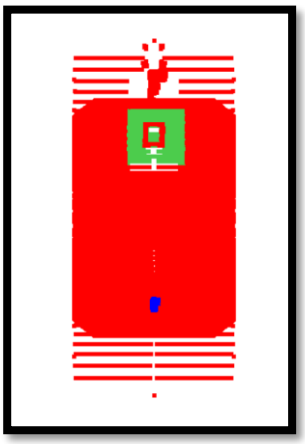

Figure 4: Environment point-cloud with cropping

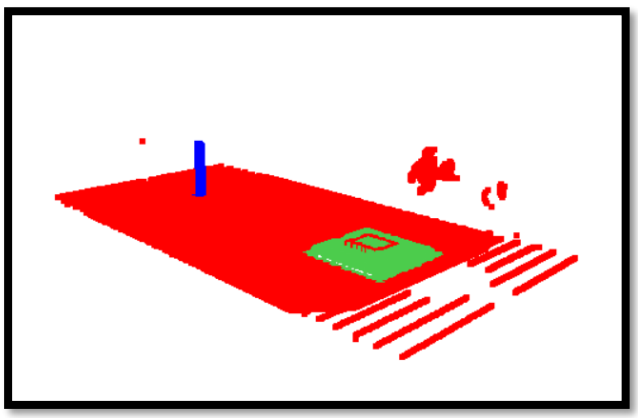

Figure 5: Environment point-cloud with cropping

We capture visual input as 3D point clouds using RGB cameras to detect important elements in the scene, such as the pole and the ring. Instead of constructing heightmaps, the point cloud data represents the environment in three dimensions, capturing both spatial information and color directly. This data is then cropped to isolate relevant objects such as the ring and the pole for further analysis and trajectory planning. However, this wasn't useful to us as we didn't use this data captured from the cameras to do any object manipulation. Therefore, this open for further development.

IV. Experiments and Results

A. *Environmental Setup*

In order to run the project without any problems, the systems needs are Python 3.8.10 version and RAI Robotics 0.1.10 version. Open3D version 0.13.0. Latest PyTorch in Linux and Google Colab.

B. *Enhanced Simulation in a Known Environment*

1. Holding the Ring: Implement a gripping mechanism for the robot's end-effector to hold the ring securely but allow precise release at the right moment for the throw. - Refine the grip to allow slight rotation or adjustments, simulating how a human might stabilize an object before throwing.
2. Defining the Throwing Movement: Create a dynamic throwing motion, considering force, angle, and rotation of the ring. Use the physics engine to simulate gravity, air resistance, and rotational inertia. - Incorporate inverse kinematics to adjust the robot's joints for a fluid, consistent throw arc toward the target. - Implement trajectory planning to determine precise release point and velocity to ensure the ring can encircle the target post.
3. Throwing to Encircle the Target: Add a feedback loop with error correction. If the ring's path deviates, the system adjusts force or angle for subsequent throws. - Integrate a physics-based model of "encircling success" so the robot can recognize a successful throw around the target.

C. *Variable Prediction by Machine Learning*

In the first part in section B, we calculated variables based on projectile motion. However, the results were not precise thus we modified the variables with trial and error approach. Then, instead of generalizing those variables by using a machine learning model, we tried to find optimal variables for different target locations.

We collected sample data by experimenting with random variables given in Table 2 except success_prob, for 1000 trials. We limited the parameters in a range around a known successful attempt to generate useful data in a limited amount of time. Range is found by trajectory optimization calculations. Known succesfull attempt is like these variables: *q2 = -0.31, time1 = 0.1, time2 = 1.2, release_threshold = 0.52*.

Table 2: Data Variables with Best Success Probability

q2	q3	time1	time2	release_threshold	success_prob
-0.28	2.0	0.15	1.3	0.6000	0.835536
-0.29	2.0	0.15	1.3	0.6000	0.834987
-0.28	2.0	0.15	1.3	0.5625	0.834457
-0.30	2.0	0.15	1.3	0.6000	0.834436
-0.29	2.0	0.15	1.3	0.5625	0.833905
-0.31	2.0	0.15	1.3	0.6000	0.833884
-0.28	2.0	0.15	1.3	0.5250	0.833372
-0.30	2.0	0.15	1.3	0.5625	0.833351
-0.32	2.0	0.15	1.3	0.6000	0.833330
-0.29	2.0	0.15	1.3	0.5250	0.832817

This shows the best possible configurations for successful throws.

Table 3: Hyper-parameters

Parameter	Value
#hidden layers	4
#hidden units per layers	3
Optimizer	Adam
Batch size	16
Learning Rate	0.001
Activation	ReLU
#Epoch	2000

Our model architecture for training the neural network consisted of four hidden layers, each containing three hidden units. We employed the Adam optimizer for efficient parameter updates with a batch size of 16 to balance computational efficiency and convergence stability. The learning rate was set to 0.001 to ensure gradual improvements during the training process. The activation function used was ReLU (Rectified Linear Unit), which enabled faster training by mitigating vanishing gradient issues. The training process spanned 2000 epochs, allowing the model sufficient iterations to converge towards optimal parameters. To generate training data, we conducted 1000 trials with varying release parameters and collected performance metrics, which informed the model's predictions and enhanced its accuracy in generating successful throws across different conditions.

V. Evaluation and Discussion

The performance of our system was assessed by evaluating the precision ring toss task using a combination of physics-based trajectory calculations and machine learning optimizations. The use of KOMO ensured accurate path planning, while adjustments to gripper stretching and release parameters addressed deviations in the x-axis due to the initial grasp pose. Conducting 1000 trials enabled us to collect extensive training data, and the CNN-based model achieved a reasonable success rate in predicting effective release parameters. However, the perception module's limitations in leveraging depth data to estimate precise distances indicate room for improvement.

Despite achieving a notable success rate, our experiments showed that false positives affected performance due to the nature of the ring's hollow center. Introducing a feedback mechanism to dynamically adjust parameters based on real-time data could improve the system's robustness. Additionally, in the future, incorporating a deep perception module with depth-aware features would enhance the robot's spatial understanding, enabling more precise detection and object manipulation. Overall, the results demonstrate the potential of combining physics-based approaches with neural networks to achieve dynamic, high-precision tasks, although further development of the perception module remains critical for improving accuracy and adaptability.

VI. Video

References

[1] A. Siswoyo, “Developing A Robot to Improve The Accuracy of Ring Retrieval and Throwing at The ABU Robocon Indonesia Robot Competition,” International Journal of Applied Sciences and Smart Technologies, vol. 5, issue 2, pp 315–334, p-ISSN 2655-8564, e-ISSN 2685-9432

[2] Kasaei, H. & Kasaei, M. (2022). Throwing Objects into a Moving Basket While Avoiding Obstacles. arXiv:2210.00609.

[3] Werner, L., Nan, F., Eyschen, P., Spinelli, F. A., Yang, H., & Hutter, M. (2024). Dynamic Throwing with Robotic Material Handling Machines. arXiv:2405.19001

[4] Zeng, A., Song, S., Lee, J., Rodriguez, A., & Funkhouser, T. (2020). TossingBot: Learning to Throw Arbitrary Objects with Residual Physics. IEEE Transactions on Robotics, arXiv:1903.11239

[5] Hu, J.-S., & collaborators. Stereo Vision-Based Ball Throwing System.

[6] Gai, Y. Analytical Approaches for Flexible-Link Manipulator Control.

[7] https://m.youtube.com/watch?v=5DXqmLxgHWM

[8] https://m.youtube.com/shorts/33O1HMONYk0

[9] https://marctoussaint.github.io/robotics-course/rai.html

[10] https://github.com/MarcToussaint/rai-tutorials

[11] https://github.com/muddasser-mm/Thrower_Goalie_RobotArms