Reinforcement Learning Powered Robotic Arm Using Embedded AI Techniques

Arman Hosseinpour
Supervisor: Assoc. Prof. Dr. Hatice Doğan

Dokuz Eylül University
Engineering Faculty
Electrical and Electronics Engineering Department

Abstract

This project develops an embedded AI framework for real-time pick-and-place robotic tasks, focusing on transferring learned models from simulation to physical systems. A custom reinforcement learning (RL) environment was built using physics simulation tools, and an image processing module extracts object coordinates to guide the RL model. The trained model is optimized and deployed on a Raspberry Pi 3 for real-time performance, demonstrating how integrating RL, image processing, and embedded AI enables adaptable and efficient robotic solutions for applications like automation and manufacturing.

Introduction

This project aims to develop a flexible, low-cost embedded AI system for real-time pick-and-place tasks using a robotic arm. It addresses the growing need for adaptable automation solutions in environments with limited resources by combining reinforcement learning, image processing, and embedded computing on devices like the Raspberry Pi. The robotic arm is trained in a custom simulation environment to learn how to reach target object positions based on coordinates detected by a camera system.

The system integrates a vision module that provides real-time object location data to the reinforcement learning model, which controls the arm's movements. A magnetic coil enables the robot to pick and place objects smoothly and efficiently. The trained model is optimized to run on low-power embedded hardware, ensuring energy efficiency and real-time performance. The physical robot is constructed using 3D printing and stepper motors, and an easy-to-use interface is provided to allow users to operate and fine-tune the system without technical expertise. Future enhancements may include applications like robotic chess playing or 3D environment mapping for improved safety and adaptability.

Methodology and Application

Simulation

The simulation setup provides a virtual environment for designing, testing, and optimizing the robotic arm before deploying it in the real world [1]. PyBullet was chosen as the physics engine due to its ease of use, real-time performance, and compatibility with URDF files [2] . The robot was modeled using a URDF file, defining its links, joints, physical properties, and appearance, and imported into PyBullet [3]. Figure 2 shows the simulation environment used in this project.

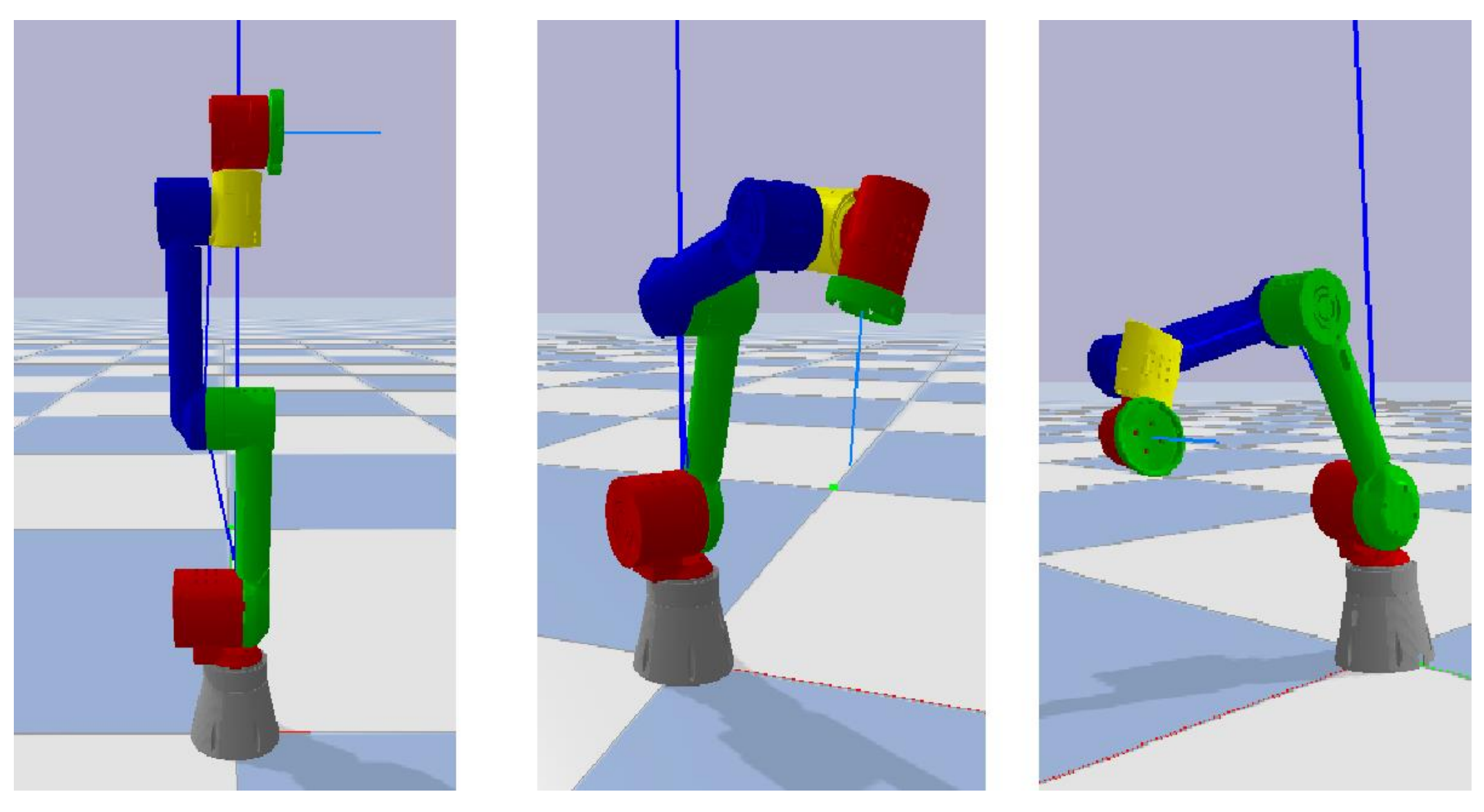


Figure 1: Simulation environment for robotic arm tasks.

To train reinforcement learning agents, a custom Gymnasium environment was created, tailored specifically for the robotic arm's pick-and-place task [4]. This environment integrates PyBullet for realistic physics and defines task-specific rules, limits, and goals. The action space is defined using a MultiDiscrete format, allowing fine-grained control of each motor, while the observation space provides joint states and target coordinates. A carefully designed reward function encourages efficient, safe, and accurate movements. Equations 1 to 5 describe the reward function.

$$R = w_d R_d + w_c R_c + w_t R_t + w_a R_a \tag{1}$$

w_d, w_c, w_t, and w_a are scalar weights for distance, collision, target, and angle rewards, respectively, and $\Delta d = d_{t-1} - d_t$ is the change in distance.

$$R_d = \begin{cases} -2 \cdot \exp\left(100(d_{t-1} - d_t)\right), & d_{t-1} < d_t \\ \exp\left(100(d_{t-1} - d_t)\right), & d_{t-1} > d_t \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$R_t = \begin{cases} 250, & d_t < 0.01 \end{cases} \tag{3}$$

$$R_c = \begin{cases} -250, & \text{if collision} \end{cases} \tag{4}$$

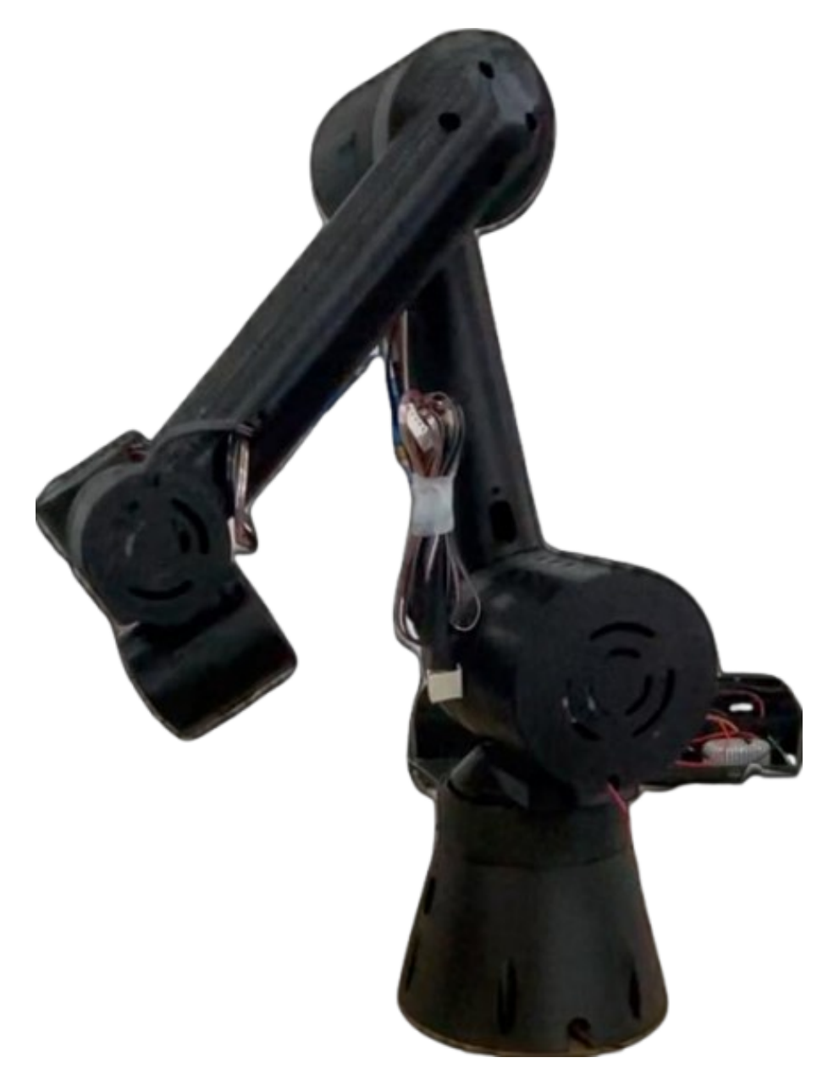
$$R_a = \begin{cases} \exp\left(\frac{30-\theta}{30}\right), & \theta < 30^\circ \\ -1, & \text{otherwise} \end{cases} \tag{5}$$

Training

Proximal Policy Optimization (PPO) was used due to its stability and efficiency for continuous control tasks [5]. Training was performed using `stable-baselines3` on a Ryzen 9 5950X CPU, with PyBullet simulation running at approximately 350 FPS. The goal was to move the robotic arm's end-effector to random targets given in the observation space. The algorithm used a learning rate of 0.0003, a discount factor of 0.99, and updated its policy every 2048 steps over a total of 5 million timesteps.

Real World

The robotic arm was fabricated using PLA for structural parts and ABS for motor-adjacent components, printed over seven days with an FDM printer to ensure strength and stability [6]. Actuators consisting of stepper motors with 1:38 gear reductions were assembled and mounted onto the PLA structure, achieving 5 degrees of freedom aligned with the URDF model from simulation. An electromagnetic gripper capable of lifting up to 2.5 kg, with a grab-detection switch, was used instead of a mechanical gripper. The control system centers on a Raspberry Pi 3 managing five A4988 stepper drivers via GPIO pins, with additional relays controlling the gripper and a cooling fan. Calibration and detection switches are connected to input GPIOs with pull-down resistors for reliable signals, while current limits on drivers were set to protect the motors.



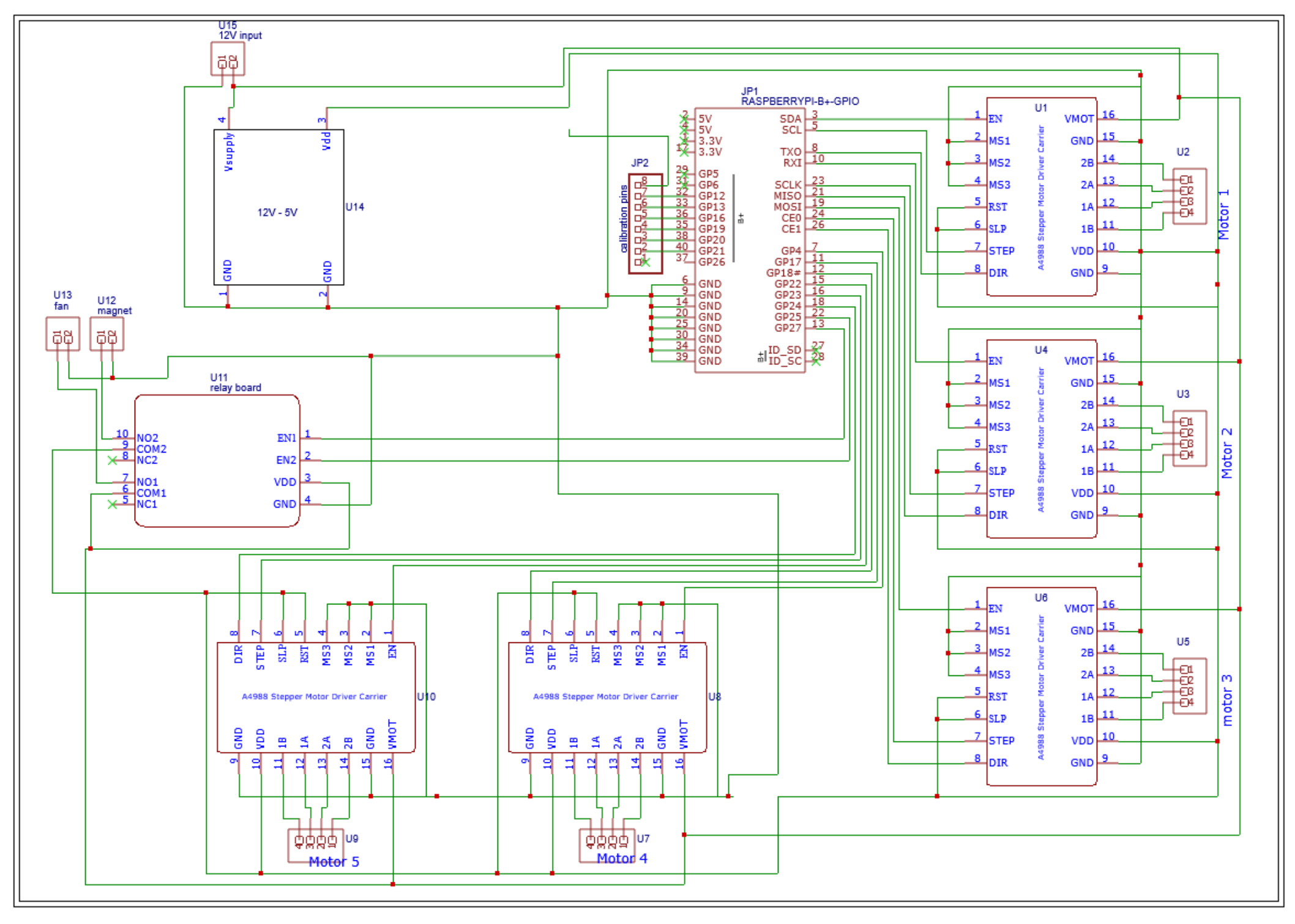


Figure 2: Wiring diagram of the circuit designed.

After training on a powerful computer, the model was optimized for the Raspberry Pi 3 through pruning to remove unnecessary weights, exporting the policy network to ONNX format, and running it efficiently with ONNX Runtime. Embedded programming utilized multi-threading to independently control each motor via dedicated threads, manage model inference, and handle UI communication, ensuring smooth real-time operation. The system architecture includes a `MotorControl` class for individual motor management and calibration, a `Robot` class overseeing all motors and signals, and a `Server` class that bridges communication between the robot and user interface. This modular design, illustrated in the class and state diagrams, enables efficient control and coordination of the robotic arm on embedded hardware.

The image processing module, triggered from the UI by clicking "Detect Object," captures a photo of the chessboard and uses OpenCV to detect blue boundary lines for camera calibration and perspective correction, then identifies the green object's position and sends its coordinates to the Raspberry Pi. The UI, built with React and JavaScript, communicates with the robot's Python server via an API, offering motor control sliders for real-time joint movement, a 3D model that visualizes joint rotations, detailed joint status and activity graphs, plus buttons to run the trained model, perform object detection, and control the magnet and fan—providing a seamless, user-friendly interface for managing the robotic system.

Conclusion

This project developed an embedded AI system for a robotic arm to perform real-time pick-and-place tasks by training a reinforcement learning model in a custom physics-based simulation environment, which enabled faster, safer training without hardware damage. The system integrates image processing to detect objects and provide coordinates to the RL model, enhancing adaptability to changing conditions. After training, the model was optimized to run efficiently on a low-power Raspberry Pi 3, demonstrating that smart robotic control is achievable on low-cost hardware. Overall, this work highlights the potential for affordable, practical robotics using reinforcement learning, image processing, and embedded AI, with opportunities for future improvements in sensing, task complexity, and multi-robot collaboration.

Acknowledgment

I would like to thank my supervisor Assoc. Prof. Dr. Hatice Doğan, my teammate Ayhan Tavangar for his contributions, and Eren Erdem for his help with 3D printing.

References

[1] Stephen James and Edward Johns. 3d simulation for robot arm control with deep q-learning. *ArXiv*, abs/1609.03759, 2016.

[2] Erwin Coumans. *PyBullet Physics Engine Documentation*, 2024. Accessed: 2024-12-30.

[3] Willow Garage. *Unified Robot Description Format (URDF)*, 2010. Accessed: 2024-12-30.

[4] Farama Foundation. *Gymnasium Documentation*, 2024. Accessed: 2024-12-30.

[5] Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep reinforcement learning for robotics: A survey of real-world successes. *Annual Review of Control, Robotics, and Autonomous Systems*, 8, 2024.

[6] Jeff Kerr. We-r2.4 six axis robot arm, 2023. Accessed: 2025-01-04.