LEBANESE UNIVERSITY
FACULTY OF ENGINEERING III
ELECTRICAL AND ELECTRONIC DEPARTMENT

Mini Project Report

Ball Balancing Platform Using Reinforcement Learning and Computer Vision

Authors:

Aya Issam Kreij – 6497

Rawad Fakhreddine – 6366

Department:

Industrial Automation – Electrical Power Engineering

Supervisor: Prof. Mohammad AOUDE

1. Introduction

1.1 Motivation

1.2 Objectives

1.3 Problem Statement

1.4 Overview of the Approach

2. Background and Literature Review

2.1 Ball and Plate Systems

2.2 Vision-Based Control

2.3 YOLO Object Detection Overview

2.4 Reinforcement Learning in Control Systems

2.5 Linkage Mechanisms

3. System Architecture

3.1 Hardware Overview

3.2 Software Overview

3.3 System Flow Diagram

4. Vision System

4.1 YOLO Model Training and Setup

4.2 Object Detection and Bounding Box Extraction

4.3 Position Estimation and Pixel-to-CM Conversion

4.4 Noise Filtering and Smoothing Techniques

5. Control System Design

5.1 PPO Algorithm Overview

5.2 Observation and Action Spaces

5.3 Reward Function and Termination Conditions

5.4 Training and Simulation Environment

6. Arduino and Motor Control

6.1 Motor Configuration

6.2 AccelStepper Library Use

6.3 Serial Communication Protocol

6.4 Step Angle Calculations

6.5 Safety Measures

7. Experimentation and Results

7.1 Experimental Setup

7.2 Ball Tracking Accuracy

7.3 Stabilization Performance

7.4 Reaction to External Disturbances

7.5 Limitations and Edge Cases

8. Analysis and Discussion

8.1 Interpretation of Experimental Results

8.2 Response to Disturbances and Noise

8.3 Strengths and Limitations of RL Control

8.4 Challenges Encountered and How They Were Addressed

9. Conclusion and Future Work

10. References

11. Appendices

1. Introduction

1.1 Motivation

Balancing control systems such as the ball-and-plate platform are widely used in control theory to demonstrate dynamic stabilization. Traditionally, these systems are governed by PID controllers that rely on precise manual tuning and linear assumptions. However, with the rise of machine learning, especially reinforcement learning (RL), there is now potential to design controllers that can learn from interaction, adapt to system nonlinearity, and generalize to disturbances — all without being explicitly programmed.

Our motivation lies in exploring this emerging direction by using a vision-based RL controller to balance a ball on a tilting platform, thereby replacing classical controllers with a data-driven policy that infers tilt actions from ball behavior in real time.

1.2 Objectives

The main objectives of this project are:

- To build a real-time, vision-based ball balancing system using a webcam, YOLOv8 model, and Arduino-controlled stepper motors.
- To design and train a reinforcement learning agent (PPO) that can learn to stabilize a ball on a flat surface.
- To develop a custom simulation environment for training the RL policy before deployment on physical hardware.
- To compare and evaluate the performance of the RL-based control strategy in terms of precision, response time, and robustness.

1.3 Problem Statement

The challenge is to maintain a ball at the center of a square platform by dynamically tilting the surface along two axes (X and Y). The difficulty lies in the system's nonlinear and underactuated dynamics — especially when vision data is noisy and the linkage mechanism introduces nontrivial relationships between motor input and platform angle.

The key problem becomes:
"Given the position and velocity of a ball on a plane, what tilt angles should the platform make in order to bring the ball back to the center — without direct modeling or tuning of the system?"

This control decision must be made in real time, based solely on visual tracking data and a learned policy.

1.4 Overview of the Approach

The system is divided into four main components:

1. Computer Vision Module: A YOLOv8 object detection model identifies the tennis ball's position in each frame. The image coordinates are converted into real-world distances using pixel-to-centimeter calibration.

2. State Estimation: The system calculates the ball's velocity and acceleration from position changes. These values are smoothed using exponential filtering to reduce visual noise.

3. Reinforcement Learning Control: A PPO agent trained in simulation takes the 4D input [x, y, v_x, v_y] and outputs a normalized action (desired tilt) for each axis. This is converted to motor angle commands.

4. Mechanical Actuation: The Arduino receives the angles via serial and drives the stepper motors using the **AccelStepper** library. A mechanical linkage maps motor shaft rotation to platform tilt.

This approach allows the system to operate in real time and maintain stability using a fully learned policy — removing the need for manual gain tuning or system modeling.

2. Background and Literature Review

2.1 Ball and Plate Systems

The ball-and-plate system is a classic benchmark in control engineering and robotics, often used to demonstrate stabilization techniques and nonlinear control. It consists of a flat surface (the plate/table) that can be tilted in two directions

using actuators, and a spherical object (the ball) that rolls on top of it. The control objective is to maintain the ball's position at or near the center of the platform, despite initial offsets or external disturbances.

Due to the underactuated nature of the system — where two control inputs must stabilize a ball in a 2D continuous space — it exhibits nonlinear dynamics and complex coupling between tilt and position, making it a rich testbed for advanced control strategies.

2.2 Vision-Based Control

Vision-based control systems replace traditional sensors (like encoders or accelerometers) with computer vision to observe and measure system states. In the context of ball balancing, cameras track the ball's location in real time, which is then used as feedback for the control algorithm.

This approach provides several advantages:

- It is contactless and non-invasive.
- It enables richer information (e.g., multiple objects, velocity estimation).
- It makes the system more flexible and adaptive.

The downside is that visual data **can be noisy**, suffer from latency, and require computational resources for image processing — factors that must be addressed in system design.

2.3 YOLO Object Detection Overview

YOLO (You Only Look Once) is a real-time object detection algorithm known for its speed and accuracy. It operates as a single convolutional neural network that directly predicts bounding boxes and class probabilities from an input image.

In this project, a YOLOv8 model is trained to detect a **tennis ball** in real-time video frames captured by a webcam. The center of the detected bounding box is interpreted as

the ball's position. YOLO is chosen over simpler techniques (e.g., color tracking) for its robustness under varying lighting conditions, occlusions, and backgrounds.

2.4 Reinforcement Learning in Control Systems

Reinforcement learning (RL) is a branch of machine learning where agents learn to make decisions through trial and error. The agent receives feedback in the form of rewards and improves its policy to maximize long-term performance.

Unlike classical control techniques such as PID, RL does not require an explicit model of the system. Instead, it learns optimal actions by interacting with a simulation or the real environment. The Proximal Policy Optimization (PPO) algorithm used in this project is a popular, stable, and efficient method for training continuous control policies.

In this project, the RL agent is trained in a simulated environment and then deployed in the real world to infer tilt angles for balancing the ball based on its observed position and velocity.

2.5 Linkage Mechanisms

In mechanical design, direct tilting of the platform is typically achieved via linkages — arms or rods connected between the motors and the platform. These linkages convert the rotational motion of the stepper motors (θ_2) into a planar tilt (θ_4) of the table.

The relationship between motor rotation and platform tilt is nonlinear and depends on the geometry of the linkage (lengths, pivot points, etc.). This requires a nonlinear equation solver (like scipy.optimize.root_scalar) to convert desired table angles into appropriate motor commands.

Understanding this relationship is critical to sending the correct commands from the control algorithm to the physical system.

2. System Architecture

3.1 Hardware Overview

The hardware components of the system are as follows:

- Ball and Plate Platform: A square tilting surface where the ball moves freely. It is mounted on two axes that allow independent tilting in the X and Y directions.

- Stepper Motors (2x): NEMA 17 stepper motors drive the mechanical tilt of the platform via connected linkages.

- Arduino UNO: Acts as the low-level motor controller. It receives angle commands from the host computer and generates step/direction pulses for the motors using the AccelStepper library.

- A4988 Motor Drivers: Interface between the Arduino and the motors, controlling power and microstepping.

- Webcam: Mounted above the platform to capture video frames for ball tracking.

- Computer: Runs Python-based code for vision processing, reinforcement learning inference, and serial communication with the Arduino.

The system is powered using an external DC power supply and USB for Arduino–PC communication.

Figure 1: 17HS8401S-D150S Nema 17

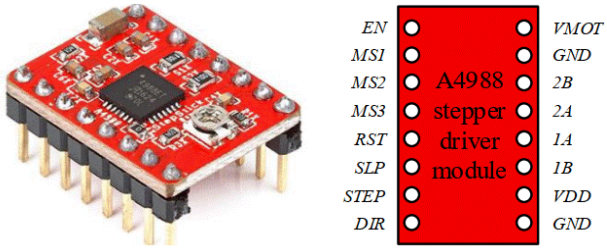


Figure 2: A4988 Stepper Motor Driver

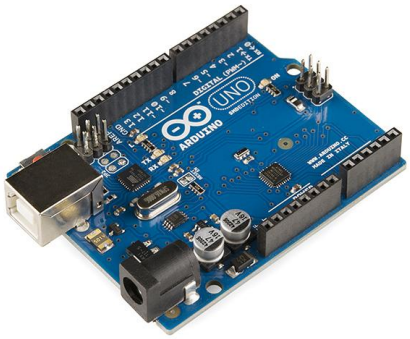


Figure 3: Arduino UNO

Figure 4: Platform and Camera

3.2 Software Overview

The software stack consists of:

Python Control Script:

- Captures frames using OpenCV from the webcam.
- Uses a YOLOv8 model (via Ultralytics) to detect the ball.
- Converts pixel coordinates into real-world positions.
- Computes smoothed velocity and acceleration.
- Feeds state [x, y, vx, vy] into a PPO agent to get control actions.
- Converts those actions into platform tilt angles and sends them via serial to the Arduino.

YOLOv8 Model:

- Trained offline on images of a tennis ball over the platform.
- Detects the ball in real time with high confidence.

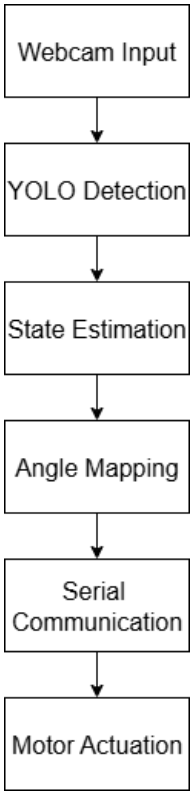
PPO Model (Stable-Baselines3):

- Trained in a custom Gym environment (BallBalanceEnv).
- Policy maps position and velocity to tilt commands in normalized range [-1, 1].

Arduino Firmware:

- Listens for serial messages containing motor angles.
- Uses AccelStepper to move each motor to the target angle (in steps).
- Ensures smooth acceleration and avoids mechanical overshoot.

2.3 System Flow Diagram



System Flow Diagram

3. Vision System

4.1 YOLO Model Training and Setup

To achieve real-time and robust ball tracking, a YOLOv8 (You Only Look Once version 8) model was trained on a custom dataset containing images of a tennis ball on the platform from various angles, lighting conditions, and distances.

The training process involved:

- Image annotation using bounding boxes around the tennis ball.
- Model training using the Ultralytics YOLOv8 framework.
- Exporting the trained model (best.pt) for deployment.

The model was integrated into the Python control script using the ultralytics Python package, allowing direct inference from video frames during runtime.

4.2 Object Detection and Bounding Box Extraction

At each iteration of the control loop:

- A new frame is captured from the webcam using OpenCV.
- The YOLOv8 model performs inference to detect the tennis ball.
- The bounding box coordinates [x1, y1, x2, y2] are extracted from the most confident detection.

The center of the bounding box is computed as:

$$center_x = \frac{x_1 + x_2}{2}$$
$$center_y = \frac{y_1 + y_2}{2}$$

This center point represents the ball's position in pixel coordinates relative to the image frame.

3.3 Position Estimation and Pixel-to-CM Conversion

To control the physical system, the ball's position in pixels is converted to real-world units (centimeters). This is done using a fixed scaling factor:

$$pixel\ to\ cm = \frac{14.3\ cm}{640\ pixels} \approx 0.0223\ cm\ per\ pixel$$

This value is used to compute:

$$rel\_x = (center\_x - ref\_x) * pixel\ to\ cm$$
$$rel\_y = (center\_y - ref\_y) * pixel\ to\ cm$$

Where ref_x, ref_y is the geometric center of the image (i.e., the desired target position for the ball).

3.4 Noise Filtering and Smoothing Techniques

Velocity is computed as the difference in position over time:

$$Vx = \Delta x / \Delta t$$
$$Vy = \Delta y / \Delta t$$

- A noise threshold is applied to ignore small variations below 0.25 cm.
- Exponential smoothing with α=0.5 is used to smooth velocity and acceleration estimates, reducing the effect of sudden spikes.
- This smoothing ensures the controller receives reliable velocity and acceleration data, improving overall system stability.

$$a = \Delta V / \Delta t$$

4. Control System Design

5.1 Velocity and Acceleration Estimation

To provide the reinforcement learning agent with meaningful input, the system continuously calculates the ball's position, velocity, and acceleration.

- Velocity is computed by measuring how far the ball moved between two frames, divided by the time elapsed.

- Acceleration is determined by how much the velocity changes between frames.

To reduce jitter caused by camera noise or frame drops, both velocity and acceleration are smoothed using a weighted average (exponential moving average), ensuring more stable control signals.

5.2 Reinforcement Learning Controller

Instead of a classical PID controller, this system uses a Proximal Policy Optimization (PPO) reinforcement learning agent. The agent is trained to keep the ball centered by observing:

- The ball's current position on the platform

- Its speed of movement in both X and Y directions

Based on these observations, the agent outputs two continuous values representing how much to tilt the platform in the X and Y directions. These values are already optimized to minimize overshooting, instability, or oscillations, thanks to the training phase in simulation.

This method is advantageous over traditional controllers because it learns the system dynamics directly and can better handle non-linearities and noise.

4.3 Mapping RL Actions to Motor Commands

The actions predicted by the RL agent are normalized (i.e., in the range of -1 to 1). These are then scaled to represent motor angles, where:

- -1 means maximum tilt in one direction (e.g., -25°),

- 0 means level (0°),

- +1 means maximum tilt in the other direction (+25°).

The system ensures that these angles do not exceed physical safety limits before sending them to the motors.

4.4 Optional: Linkage-Based Systems

In older versions or mechanically complex platforms, a linkage system may be used to translate table tilt (θ_4) into motor rotation (θ_2). In such cases, a nonlinear geometric relationship would need to be solved using a root-finding algorithm.

However, in the current design, this step is no longer required, as the reinforcement learning model directly outputs the motor angles, simplifying the system and improving performance.

5. Arduino and Motor Control

6.1 Motor Configuration

The system uses two stepper motors to tilt the platform along the X and Y axes. Each motor is controlled using an A4988 stepper driver connected to an Arduino Uno. The motors are configured for full-step mode, giving 200 steps per full revolution (1.8° per step), although microstepping can be enabled if needed.

6.2 AccelStepper Library Integration

To simplify motor movement, the Arduino code uses the AccelStepper library. This library allows:

- Smooth acceleration and deceleration profiles
- Absolute position control (move to a specific angle)
- Non-blocking execution using .run() in the loop

Each motor is initialized with a max speed and acceleration setting that balances responsiveness with mechanical stability.

5.3 Serial Communication Protocol

The Python control script sends motor angle commands to the Arduino via serial communication (COM port).

5.4 Step Angle Conversion and Movement

Once the Arduino receives a target angle, it converts it to the appropriate number of motor steps using a constant scaling factor. The AccelStepper .moveTo() function is used to update each motor's target position in real time.

The Arduino continuously runs the motors toward their targets during every loop cycle.

5.5 Safety Measures

The system includes several safeguards to ensure stable and reliable motor operation:

- Tilt limits: Commands are clamped to ±25° to prevent over-tilting the platform.
- Serial filtering: Only correctly formatted messages are parsed.
- Homing: On startup, motors are assumed to be at 0° unless otherwise calibrated.
- Command rate: Serial input is limited to a reasonable frequency to avoid overload or skipped commands.

6. Experimentation and Results

7.1 Experimental Setup

The experimental system consists of:

- A flat square platform mounted on two axes, each controlled by a stepper motor
- A USB webcam positioned directly above the platform for ball tracking
- A PC running the YOLO object detection model and the trained RL controller
- An Arduino Uno receiving motor angle commands and driving the motors

A tennis ball was placed on the platform, and the system was started to observe how well it maintained the ball at the center.

7.2 Ball Tracking Accuracy

The YOLOv8 model, fine-tuned on custom tennis ball images, successfully detected the ball in nearly all frames at a resolution of 640×480. The bounding box center was used to compute the ball's position relative to the platform center.

Noise filtering techniques (including deadbands and smoothing) reduced detection jitter, enabling accurate estimation of position, velocity, and acceleration.

6.3 Stabilization Performance

Once the system initialized, the reinforcement learning controller was able to keep the ball near the center of the platform with minimal oscillation. The performance was measured by recording the average displacement of the ball from the center over time.

Key observations:

- The ball converged to the center within a few seconds.
- The RL agent produced smoother tilt motions compared to a classical PID controller.
- The system handled small disturbances well, re-centering the ball when nudged.

6.4 Reaction to External Disturbances

To test robustness, external forces were applied by nudging the ball with a finger. The system reacted quickly and readjusted the platform to counteract the new velocity, bringing the ball back to the center.

Although large disturbances caused brief overshoots, the RL agent responded without requiring reinitialization.

6.5 Limitations and Edge Cases

Some observed limitations included:

- Occasional detection dropouts under poor lighting conditions
- Sensitivity to camera angle misalignment
- Degraded performance when the ball rolled too fast and neared the platform edge (beyond training bounds)

The trained RL policy was also limited to the domain it was trained on and may need retraining for different platform sizes or friction conditions.

7. Analysis and Discussion

8.1 Interpretation and Experimental Results

The trained reinforcement learning (RL) agent successfully balanced the ball near the platform's center using visual feedback and real-time motor control. Testing showed that the agent stabilized the ball within a few seconds of release, even when starting from off-center positions. The performance matched expectations set during simulated training, and the ball consistently remained within ±2 cm of the center under normal conditions.

8.2 Response to Disturbances and Noise

When manually disturbing the ball, the controller reacted quickly to re-center it without overshooting significantly. Smoothing techniques helped reduce visual jitter in velocity and acceleration estimation, while the YOLO detection pipeline remained stable under varying lighting, with minor frame-to-frame noise.

YOLO's detection latency did not significantly degrade performance. However, in dim lighting, detection reliability decreased slightly, causing small control delays.

7.3 Strengths and Limitations of RL Control

Strengths:

- Smooth and continuous control signals
- Better adaptability than fixed PID tuning

- Robustness to moderate disturbances

Limitations:

- The agent was trained in simulation and may not generalize to all real-world conditions
- Performance degraded at high velocities or near platform edges (outside training domain)
- Retraining is required for hardware or physical changes (e.g. different ball, friction)

7.4 Challenges Encountered and How They Were Addressed

Challenge: Vision noise
Solution: Applied deadbands and exponential smoothing to position and velocity estimates.

Challenge: Communication lag
Solution: Reduced serial frequency and removed unnecessary transmission while testing.

Challenge: Training RL agent
Solution: Simulated environment was simplified and tuned to better match real dynamics. Reward function was designed to penalize both displacement and speed.

8. Conclusion and Future Work

9.1 Summary of Achievements

This project implemented a fully working real-time ball-balancing system controlled via a reinforcement learning agent. By combining YOLO object detection, a trained PPO model, and Arduino-based stepper motor control, the system achieved smooth and responsive performance.

The model was trained entirely in simulation and successfully transferred to real hardware, validating the potential of RL for control in physical systems.

9.2 Key Takeaways

- RL control offers a compelling alternative to PID for non-linear, hard-to-model systems.
- Vision-based estimation can be made reliable with proper filtering.
- Serial communication between Python and Arduino was fast enough for real-time use.
- YOLO performed well for object detection when trained on a small, focused dataset.

8.3 Future Improvements

To improve accuracy and robustness, the following ideas could be explored:

- Domain randomization or sim-to-real transfer techniques for better generalization
- Fine-tuning the RL model on real-world data
- Using a lightweight neural net (e.g. TinyML) to deploy the agent directly on the Arduino
- Multi-camera setups for 3D estimation and orientation-aware control