Ball and Beam Software Documentation

Shivam Bhatt

September 2025

Contents

1 Introduction

This document provides basic instructions to get started with the PID controller code for the ball and beam setup. We will first cover the installation and setup instructions for the software environment, then explain how to run your code, and lastly discuss some key details of how the code works.

2 Installation

2.1 Installing Miniconda

Miniconda is a distribution of Python and R that allows us to create isolated virtual environments with different packages and dependencies. Miniconda is useful because it makes it very easy to manage complex dependencies for our projects. Additionally, we can export any environment we create and share it with others. This allows us to share code that will work on any computer regardless of the build configuration and operating system.
For more information, see the Miniconda documentation:
`https://www.anaconda.com/docs/getting-started/miniconda/main`

2.1.1 Installation Steps

To install Miniconda, go to this link and follow the instructions:
`https://www.anaconda.com/docs/getting-started/miniconda/install`

Note

For step 7 of the installation process, choose **yes** when prompted.

2.2 Importing an Environment

Follow these steps to import and activate the Ball and Beam environment:

1. Open a terminal on your Raspberry Pi or laptop.
2. Navigate to the folder where your environment file (`Ball_Beam_env.yml`) is stored:

```
cd path/to/folder/with/yml/file
```

3. Use the following command to create the environment (this may take some time):

```
conda env create -f Ball_Beam_env.yml
```

4. Once the dependencies are installed, activate the environment:

```
conda activate mte_380_env
```

Note

Remember to activate the environment every time you run the code.

2.3 Installing the Arduino IDE

Follow the instructions found here to install the Arduino IDE:
`https://docs.arduino.cc/software/ide-v2/tutorials/getting-started/ide-v2-downloading-and-installing/`

2.3.1 Connecting Your Arduino

Once you have the IDE installed, connect your Arduino to your computer. If you are unsure how to do this, follow these instructions:
`https://www.arduino.cc/en/Guide/ArduinoUno/#install-the-board-drivers`

Note

You likely have an **Arduino Mega**, not an Arduino Uno. Make sure to select the correct board in the Arduino IDE.

After connecting your Arduino, upload the `servo_controller.ino` file to your Arduino.

3 Running the Code

3.1 Steps to Execute

1. Activate the environment:

   ```
   conda activate mte_380_env
   ```

2. Run the desired Python file:

   ```
   python path/to/python_file.py
   ```

4 Software Description

4.1 System Architecture

The ball and beam control system consists of three main components:

- **Python Controller**: Implements the PID control algorithm and communicates with the Arduino
- **Arduino Interface**: Receives position commands and controls the servo motor
- **Computer Vision System**: Provides real-time feedback on ball position using camera

The system uses three Python scripts that work together:

1. `simple_cal.py`: Interactive calibration tool
2. `ball_detection.py`: Computer vision ball detection module
3. `basic_controller.py`: PID controller with GUI

4.2 Calibration Script: `simple_cal.py`

4.2.1 Purpose

This script provides an interactive calibration system that configures the ball and beam setup. It must be run **first** before using the controller, as it generates the `config.json` file required by other scripts.

4.2.2 Usage

Run the calibration script:

```
python simple_cal.py
```

Follow the three-phase calibration process:

1. **Color Calibration**: Click on the ball multiple times to sample its color. Press `c` when you have collected enough samples (recommended: 10-20 clicks).
2. **Geometry Calibration**: Click on the two endpoints of the beam to establish the pixel-to-meter conversion ratio.
3. **Limits Calibration**: Press `l` to automatically find the position range by moving the servo through different angles.
4. **Save Configuration**: Press `s` to save the calibration data to `config.json`.

4.2.3 Key Parameters to Tune

Note

The `BEAM_LENGTH_M` parameter is critical for accurate position measurements. Measure your beam length precisely before calibration.

Parameter	Location	Description
`BEAM_LENGTH_M`	Line 16	Physical beam length in meters. **Must match your hardware.**
`CAM_INDEX`	Line 19	Camera device index (0 for default camera, 1 for external).
`servo_port`	Line 42	Serial port for Arduino (e.g., `COM3` on Windows, `/dev/ttyUSB0` on Linux).
`neutral_angle`	Line 43	Servo angle for level beam (typically 10-20 degrees).

Table 1: Calibration script parameters

4.3 Ball Detection Module: `ball_detection.py`

4.3.1 Purpose

This module handles computer vision tasks for detecting the ball's position in video frames. It uses HSV color space filtering to isolate the ball and calculate its position.

4.3.2 Usage

The ball detection module is typically imported by other scripts:

```
from ball_detection import detect_ball_x
# or
from ball_detection import BallDetector
```

For testing the detection independently:

```
python ball_detection.py
```

4.3.3 Key Parameters

The module reads HSV bounds from `config.json` (generated by calibration). If you need to manually adjust detection sensitivity:

Parameter	Config Section	Adjustment Guidelines
`lower_hsv`	`ball_detection`	Decrease values if ball isn't detected; increase if too much noise.
`upper_hsv`	`ball_detection`	Increase values if ball isn't detected; decrease if too much noise.
`radius` limits	Lines 89-91	Adjust min/max radius thresholds (5-100 pixels) based on ball size.

Table 2: Ball detection parameters

4.3.4 Detection Algorithm

The detection process follows these steps:

1. Convert frame from BGR to HSV color space
2. Create binary mask using HSV bounds
3. Apply morphological operations (erosion and dilation) to clean noise
4. Find contours and select the largest one
5. Calculate ball center and radius using minimum enclosing circle
6. Convert pixel position to meters from center

4.4 PID Controller: `basic_controller.py`

4.4.1 Purpose

This is the main control script that implements a PID controller with real-time GUI for parameter tuning. It integrates camera tracking, PID computation, and servo control into a complete system.

4.4.2 Usage

After running calibration, start the controller:

```
python basic_controller.py
```

The script will:

- Open a camera window showing ball tracking
- Display a GUI with sliders for PID gain tuning
- Connect to the Arduino servo controller
- Begin controlling the ball position

4.4.3 GUI Controls

4.4.4 PID Tuning Guidelines

The Proportional-Integral-Derivative (PID) controller adjusts the beam angle to maintain the ball at the desired position. Use this systematic tuning approach:

1. **Start with P-only control**: Set Ki=0 and Kd=0. Increase Kp until the system responds quickly but oscillates.
2. **Add derivative control**: Gradually increase Kd to dampen oscillations. Typical range: 0-10.

Control	Function
Kp Slider	Adjust proportional gain (0-100). Start with 10-20.
Ki Slider	Adjust integral gain (0-10). Start with 0, increase slowly if steady-state error persists.
Kd Slider	Adjust derivative gain (0-20). Start with 0-5 to reduce oscillations.
Setpoint Slider	Set desired ball position in meters from center.
Reset Integral	Clear accumulated integral error (useful after large disturbances).
Plot Results	Display position and control output graphs.
Stop	Shut down controller and close all windows.

Table 3: Controller GUI elements

3. **Add integral control**: If steady-state error remains, slowly increase Ki. Too much Ki causes instability. Typical range: 0-2.

4. **Fine-tune**: Make small adjustments to optimize performance.

Note

The controller multiplies the error by 100 (line 49) for easier tuning. If you modify this scaling factor, you'll need to retune all PID gains proportionally.

4.4.5 Key Parameters to Tune

Parameter	Location	Description
K_p	GUI slider	Proportional gain. Higher values increase responsiveness but may cause oscillation.
K_i	GUI slider	Integral gain. Eliminates steady-state error but can cause overshoot.
K_d	GUI slider	Derivative gain. Reduces oscillations and improves stability.
`dt`	Line 45	Control loop time step (default 0.033s = 30Hz). Match to your frame rate.
Output limits	Line 56	Servo angle limits ($\pm$15 degrees). Adjust based on your physical constraints.
Error scaling	Line 47	Error multiplication factor (default 100). Lower for more sensitive control.

Table 4: Controller tuning parameters

4.5 System Workflow

The complete workflow for using the ball and beam system:

1. **Hardware Setup**: Assemble the ball and beam apparatus, connect the camera and Arduino.
2. **Run Calibration**: Execute `simple_cal.py` to generate `config.json`.
3. **Verify Detection**: Optionally test `ball_detection.py` to ensure proper ball tracking.
4. **Start Controller**: Run `basic_controller.py` and tune PID gains using the GUI.
5. **Analyze Performance**: Use the "Plot Results" button to visualize system response.
6. **Iterate**: Adjust PID gains and re-test until desired performance is achieved.

4.6 Configuration File Structure

The `config.json` file contains all calibration data:

```
{
  timestamp: 2025-09-29T10:30:00,
  beam_length_m: 0.2,
  camera: {
    index: 0,
    frame_width: 640,
    frame_height: 480
  },
  ball_detection: {
    lower_hsv: [5, 150, 150],
    upper_hsv: [20, 255, 255]
  },
  calibration: {
    pixel_to_meter_ratio: 0.000625,
    position_min_m: -0.095,
    position_max_m: 0.092
  },
  servo: {
    port: COM3,
    neutral_angle: 15
  }
}
```

You can manually edit this file to fine-tune parameters without re-running calibration.

5 Troubleshooting

5.1 Common Issues

- **Environment activation fails**: Ensure Miniconda is properly installed and the PATH is configured correctly
- **Arduino not detected**: Check USB connection and verify the correct COM port is selected
- **Import errors**: Verify the environment is activated before running Python scripts

6 Additional Resources

- Miniconda Documentation: `https://docs.conda.io/`
- Arduino Documentation: `https://www.arduino.cc/reference/en/`
- PID Control Tutorial: Search for "PID control theory" for background information