Ball-Balancing Platform Design Documentation

Jason Christensen, James Humble, Jay Mattinson
Spring 2017
Table of Contents

Table of Contents 1
Introduction 2
Scope 2
Design Overview 3
  Requirements 3
  Hardware Dependencies 4
  Theory of Operation 4
Design Details 5
  Hardware Diagrams 5
  Hardware Details 7
  Software Diagram 9
  Software Details 9
Discussion and Challenges 11
Conclusion 12
1 Introduction

This document explains the design of a ball-balancing platform, its mechanisms as well as its software. The ball balancing platform utilizes a Raspberry Pi, camera, and some servo motors to accurately track and adjust a ball upon the platform. The user is also able to control the ball via a controller. This project utilizes the OpenCV library.

2 Scope

The scope of this document is to explain the design of a ball-balancing platform both in hardware and software, the given requirements, the process of operation, as well as the methods of implementation. This document is meant to be used to understand principles of functionality and hardware generalities. This document includes several charts, figures, and graphics to help better explain and visualize aforementioned details.

This document does not cover basic Raspberry Pi configurations or operations. This document is not meant to be a guide to OpenCV or any other external or third-party library used. This document does not specifically explain or explore any other easily-obtained or public data sheet of any hardware component used in this project.
3 Design Overview

This section outlines the project requirements, the dependencies that exist, and the overall theory of operation.

3.1 Requirements

3.1.1 Interface Definition:

3.1.1.1 Microprocessor: Main processing unit that controls the state of the device. It retrieves data from a camera and uses image processing to calculate and adjust the position of a ball using servo motors.

3.1.1.2 Control Algorithm: The control algorithm captures the data from the camera and performs calculations to track and determine the state of the machine in order to balance a ball on the surface.

3.1.1.3 Servo Motors: Servo motors are attached to the balancing plate adjusting the x-axis and y-axis.

3.1.1.4 Balancing Plate: The balancing plate is a rigid plate that attaches to servo motors and is able to move about two axes in order to balance a ball.

3.1.2 Characteristics

3.1.2.1 Performance:

3.1.2.1.1 Device shall be able to adjust the tilt of a platform along two axes.

3.1.2.1.2 Device shall be able to track the position of a ball on the platform with a camera.

3.1.2.1.3 Device shall be able to balance and steady a ball in one place on the platform.

3.1.2.1.4 Device shall be able to accept input from a joystick to control the position of the ball on the platform.

3.1.2.1.5 Device shall be able to switch between manual mode and autonomous mode.

3.1.2.1.6 Device shall be able to recenter the ball on the platform.

3.1.2.1.7 Device shall require sufficient illumination of the balancing plate in order for the camera to correctly perceive the ball.

3.1.2.2 Physical:

3.1.2.2.1 Device shall not become damaged when carried by the base.

3.1.2.2.2 Device shall be no larger than 24” long, 24” wide, and 36” tall.
3.1.3 **Electrical:**
3.1.2.1.1 Device shall be powered by one 120 V AC outlet.
3.1.2.1.2 Device shall not draw enough current to set a breaker.

3.1.4 **Environmental:**
3.1.2.1.1 Device shall withstand standard indoor conditions, including temperature, humidity, vibrations, acoustics, and gentle human interaction.

3.2 **Hardware Dependencies**
- A microcontroller capable of the following:
  - SPI interfacing (Data in and out)
  - (2) PWM signal generator pins
  - Way to interface with a camera
  - Preferable to have over 1 GHz processor speed
- A power supply capable of at least 5 V and 2 A
- A video camera capable of at least 30 FPS
- (2) High-torque servo motors
- 15” X 15” platform
- 1” Green ball preferably with matte finish
- Black Wii Nunchuck controller

3.3 **Theory of Operation**

The goal of this machine is to maintain a ball balanced upon a platform. The system begins by computing the ball’s location upon the plate through image recognition. The camera feeds its data to the microcontroller which then dissects the image using the OpenCV (Computer Vision) library to determine the ball’s location.

After the ball’s location is determined, the location is fed to PID (Proportional-Integral-Derivative) controllers who determine new servo positions. The PID controllers anticipate where the ball will be, based on where it is and where it has been in the past.

There are three modes of operation for the system. The first mode is a auto-balancing mode which balances the ball in the center of the platform. Upon a disturbance, the ball is returned back to the center. The second mode is a user-defined center, auto-balance mode. In this mode, the user uses the joystick on the controller to determine where the ball should be balanced, and the system then takes care of all of the balancing. The third and final mode is the manual-balance mode, where the user controls the servo motors directly using the accelerometers in the controller.
4 Design Details

This section outlines in detail the design of the system in all of its many components and functions. For both the hardware and software of the device, diagrams and flowcharts are provided to help better explain their workings.

3.1 Hardware Diagrams

This section provides hardware diagrams, schematics, and flowcharts that will later be described in the hardware details section.

![Diagram of basic hardware flowchart](image.png)

Figure 1. Basic hardware flowchart
Figure 2. Basic hardware schematic with general pinouts

Figure 3. Connection of the servo arms to the balancing plate
3.2 **Hardware Details**

As seen in Figure 1, the hardware for this system is composed of four main parts: the microcontroller, the servo motors, the camera, and the Wii Nunchuck controller. Each of these components is critical to the design and will be later described in this section.

Figure 2 illustrates how each of these components are physically connected with one another. For the Wii Nunchuck controller, we utilized a Wiichuck adapter to facilitate the connection of the controller to the Pi. This is seen in Figure 4.

**Wii NunChuck Controller**
The Wii controller utilizes I2C protocol in order to communicate, so it is important that its data and clock lines be connected to I2C enabled pins on the Pi. Another important thing to mention is that black and white wii controllers do not have the same initialization protocol. The data sheet for the controller is not included in this document as it is easily obtainable online. The controller further uses the 3.3 V and ground pins on the Pi.

**High Torque Servo Motors**
Still referencing Figure 2, the servo motors are connected to the Pi on pins 23 and 24 (BCM Pins) through their signal lines, and additionally to ground pins. The power lines for the servo motors are connected to an external 5 V power source as the Pi cannot handle the current draw needed to run them. High torque motors were chosen in order to maintain response time, regardless of the applied load, and in order to only use two motors.

**DC Power Supply**
For powering the servo motors, a 5 V power supply with a rating of at least 2A was chosen.

**Microcontroller**
For the microcontroller for this project the Raspberry Pi 3 model B was chosen. This microcontroller met the requirements to have: an I2C interface, (2) PWM
signal generators, and at least a 1.0 GHz processor speed. The Raspberry Pi 3 model B is clocked at 1.2 GHz, with an option of overclocking.

**Raspberry Pi Camera**
For the camera of the system, the Raspberry Pi Camera was chosen as this interfaces nicely with the Raspberry Pi. Another reason this was chosen was because it can be connected through a ribbon cable strip and does not take up any more GPIO pins. The camera was set to run at the standard resolution of 640X480 pixels.

**Platform**
The platform of the system was chosen to be made out of plexiglass and measured 15” x 15”. This was to ensure that the platform was as smooth as possible and with little warp. The platform was mounted on a camera mount, seen in Figure 5, that was slightly modified to give better angles of rotation. Figure 3 shows how the servo arms were connected to the base of the platform. It was necessary to ensure that the arms of the servos would have a small amount of play when extending and retracting so that they would not break. It was also necessary to dampen the platform’s movement with rubber bands as it had a tendency to rotate about its Z-axis.

**Ball**
For this project, it was determined through experimentation that a heavier ball was better for the system; because of this, 1” steel ball bearings were used, and painted green.

**Structure**
The structure of the system was almost entirely made from various kinds of wood. The overhang to hold the camera was constructed to attach from two sides to give better stability.
### 3.3 Software Diagram

![Software Diagram](image)

**Figure 6. Software flowchart**

### 3.4 Software Details

**GPIO Initializations.** During this stage, two important things happen: First, the controller is configured. Using functions from the WiringPi library, the I²C connection to the controller is established. For more information on this, please reference the data sheet for Wii Nunchuck controllers. The second step in this software state is the initialization of the servo motors. This is similar to the controller except that the PIGPIO library was used to establish a connection. For more information on the servo motor initialization, please see a dedicated data sheet.

**Software Initializations.** During the software initializations phase, various software embedded values and ranges are set. Among these values, are bounds for the platform such as: the center point, left and rightmost bounds, top and
bottommost bounds. All of these values were obtained through experimentation and would be specific to the system. Also done in this stage, is the setting of the default mode (Mode 0), and the initialization of the PID controller values. More information on the PID controllers is described below.

**Mode 0 (Auto-Balance).** During Mode 0, the camera is first polled and a frame obtained. From this frame, the position of the ball is obtained through OpenCV functions. If the ball is not found in the frame, this process is repeated until it is. When the ball appears in the frame and a position has been obtained, the PID values are then updated. Then, using the updated PID values, the servo motors are adjusted such that the ball maintains its position in the center of the platform. Upon pressing the “Z” button on the Wii Nunchuck controller, the program switches to Mode 1.

**Mode 1 (User Defined Auto-Balance).** Mode 1 is very similar to Mode 0. All of the same behavior exists with one main exception: after the ball position is obtained the controller’s joystick’s values are read, and the balance point is adjusted accordingly. This allows the user to control where the ball is balanced with the joystick on the controller. The PID controllers are then updated with the ball’s current position and the desired center point, and the servo motors adjusted accordingly. As before with Mode 0, upon pressing the “Z” button on the controller, the program switches to Mode 2.

**Mode 2 (Manual).** Mode 2 is a manual mode which allows the user to controller the platform directly through the accelerometers in the controller. There is no need to get the ball’s position or update PID controllers. During this stage, the controller’s accelerometer’s values are read and the servo motors are updated directly from said values. Upon pressing the “Z” button, the program switches to Mode 0.

**PID Controller.** The PID controller was largely set up through the native Arduino PID library which is open source. The program was responsible for proper setup of the library and objects. This mainly consisted of the setting of the proportional, derivative and integral weights. For those not familiar with PID control, this essentially means where the ball is, where it has been, and where it is projected to go. This document will not explain further as there are many easily obtainable sources that explore PID controllers in more depth.

**OpenCV.** Using OpenCV (Computer Vision), the ball’s location could be gathered from the captured frames of the camera. It was decided to gather the position in the Hue, Saturation and Value (HSV) spectrum rather than the Red, Blue and Green spectrum (RGB). This was decided because the reflection of the ball on the platform in the RGB spectrum sometimes gave false locations. One
challenge however that was discovered in using the HSV spectrum was the slight sensitivity to lighting. Some locations with different lighting than the lighting where the system was tuned, had less responsive adjustments. Using OpenCV “moment” function in capturing the ball’s location also gave a better approximation as it helped ignore erroneous and stray data.

5 Discussion and Challenges

Several important things were discovered or learned during the execution of the project worth mentioning. Firstly, that it is difficult to tune a PID controller. A couple of methods were tried, namely, the Ziegler–Nichols and Tyreus Luyben methods. At the end of it all, it was found that a combination of both methods, along with some manual tuning/guessing worked best. Further tuning could also yield better results.

It was also found after the physical construction of the system, that the platform was practically unhindered in rotation around the Z-axis. This is not desired as the platform tends to slide out of square as the program is running. This issue was found to stem primarily from the decision to have two pivot joints on the servo arms. For this reason, the system was required to use rubber bands to both dampen and restrict motion in this manner.

This project is an excellent introduction to the OpenCV library, as the commands and setup are relatively simple. There were, however, some pitfalls discovered along the way. The first was lighting, which affected the camera’s ability to recognize the ball, even in the HSV spectrum. In addition to this, the performance of the system was hindered also by the frame rate that was achieved by the Raspberry Pi. This system averaged between 20-25 frames/second, which was fast enough to work as desired, but would be better with higher frame rates.

Despite these mentioned challenges, the project was considered to be a great success. It is (arguably) impressive that such level of computation and adjustment of a real-time system was able to be achieved using a Raspberry Pi.
6 Conclusion

The real-time system, ball-balancing platform is a great introductory project for learning both OpenCV and the function and tuning of PID controllers. The system, while powered by a lowly Raspberry Pi 3 model B, was able to achieve frame rates of approximately 20-25 frames/second, and adjust the servo motors fast enough to balance the ball.

The different modes (auto-balance, user set auto-balance, manual mode), while not completely impervious to fault, performed just as desired and in many ways exceeded expectations.

If this project were to be redone, there would be a few things not done the same way as with most first-time projects; but overall, the project was considered to be a success. It is hoped that future groups can take this project to the next level, and create something even better.