2 DOF Ball Balancing
Andrew Pace
E90 Report
8 May 2009

Abstract
For my E90 I developed and implemented a control method for the 2DOF ball balancing problem. Sensing the location of the ball was accomplished using a digital web camera. Modeling was carried out concurrently with physical experimentation and in Simulink to verify the theoretical model. The final control method was a proportional + rate controller able to achieve stable equilibrium of the ball and change the desired location while running.
# Table of Contents

Abstract ........................................................................................................................................... 1  
Introduction ..................................................................................................................................... 3  
Theory ............................................................................................................................................. 3  
Implementation ................................................................................................................................ 7  
  Physical apparatus ............................................................................................................................... 7  
  Calibration ...................................................................................................................................... 9  
Software structure ............................................................................................................................. 10  
  Ball Detection ................................................................................................................................. 13  
  Median Filter ................................................................................................................................. 14  
  Angle Determination ..................................................................................................................... 15  
  GUI creation ................................................................................................................................. 15  
Results ........................................................................................................................................... 17  
  System Calibration ......................................................................................................................... 17  
  System Testing ............................................................................................................................... 18  
Further Work .................................................................................................................................... 22  
Conclusion ........................................................................................................................................ 22  
Bibliography .................................................................................................................................... 22  
Acknowledgements ........................................................................................................................... 22  
Appendix .......................................................................................................................................... 23  
Matlab Code .................................................................................................................................... 23  
  Main.m ........................................................................................................................................ 23  
  Control.m .................................................................................................................................... 26  
  ballFind.m ................................................................................................................................. 27  
  angleServo.m ............................................................................................................................ 29  
Servo Calibration .............................................................................................................................. 30  
System Calibration ............................................................................................................................ 31  
  X-Axis ........................................................................................................................................... 31  
  Y-Axis ........................................................................................................................................... 32
Introduction

The 2 degree of freedom (DOF) ball balancing problem is a classic control theory example. A ball is placed on a flat piece of wood and then any disturbances are attempted to be accounted for so the ball comes to rest in its initial position. In order for this to happen, the plate needed to be controlled in some fashion and the location of the ball needed to be determined. For this project, the plate was controlled using two servo motors, each controlling the tilt of the plate in one axis. The location of the ball was determined through the use of a camera mounted above the plate and then image processing done on the captured image.

The organization of this report is as follows. First, a section on theory will describe the system in general and the methods of control used, as well as the problems encountered with discrete control. After this, there is a section detailing how the project was implemented, describing both the physical apparatus and the structure and implementation of the control program. Next, the results collected and verification of the completed 2-DOF is presented. Finally, possible future work and conclusions are given.

Theory

The transfer function for the system was determined to be $\frac{\alpha}{s^2}$ for both the $x$ and $y$ axes. One assumption was that there is no sliding friction in the system, so once the ball starts to move it will not stop. Any second order system when given an impulse will behave in a similar fashion that is, an impulse will cause continuous linear motion. The $\alpha$ term accounts for the momentum of the ball.

Taking the Laplace transform of the system equation gives

$$L\left\{\frac{\alpha}{s^2}\right\} = at$$

The Laplace transform verifies the system model, as once an impulse is given to the system, the position of the ball will continue to change linearly forever.

In order to experimentally determine the system constant, a step response was given to the system. The equation for this response is

$$L\left\{\frac{\text{angle}}{s} \times \frac{\alpha}{s^2}\right\} = \alpha \times \text{angle} \times \frac{t^2}{2}$$

This yields a parabolic increase in distance for an object slid or rolled down an incline plane, confirming again the transfer function for the open system.

To control the motion of the ball, a negative feedback loop was required. The simplest method to achieve this was to use a proportional controller. As can be seen in Figure 1

![Figure 1-Proportional Controller](image-url)

---

3
the proportional controller only acts on the proportion of the error. The resulting transfer function is:

\[ \frac{Y(s)}{X(s)} = K \times \frac{\alpha}{s^2} \]

where \( K \) is the proportional constant set by the user and \( \alpha \) being the system constant. The closed loop transfer function is then

\[ \frac{X(s)}{Y(s)} = \frac{x_o s}{s^2 + K \alpha} = x_o \frac{s}{s^2 + \left(\sqrt{K \alpha}\right)^2} \]

\[ \mathcal{L}\left\{ x_o \frac{s}{s^2 + \left(\sqrt{K \alpha}\right)^2} \right\} = x_o \cos(t\sqrt{K \alpha}) \]

Producing the following theoretical output in one dimension

![Figure 2-Continuous Proportional Controller Output](image)

A problem with this analysis of the system was it assumed a continuous method of sensing and control. As the actual implementation of the concerned system was controlled with a computer, the overall system becomes discrete in nature. In order to account for this, the system equation and model can be changed

![Figure 3-Discrete Proportional Controller](image)
\[ \text{from } \frac{\alpha}{s^2} \text{ to } \frac{\alpha}{s^2} e^{-T_d s} \]

where \( e^{-T_d s} \) accounts for the delay factor, the zero order hold, and \( T_d \) is the amount of delay. In order to use classic control theory analysis, this delay needed to be changed to a polynomial. An accurate approximation was the first order Padé approximation, where

\[ e^{-T_d s} \approx \frac{-T_d s + 2}{T_d s + 2} \]

The transfer function of the proportional controller with the time delay approximation was then

\[
\frac{X(s)}{Y(s)} = \frac{\frac{K(-T_d s + 2)\alpha}{T_d s + 2}}{1 + K \frac{T_d s + 2 \alpha}{T_d s + 2 \alpha}} = \frac{K(-T_d s + 2)\alpha}{T_d s^3 + 2s^2 - T_d \alpha K s + 2K \alpha}
\]

From the Routh-Hurwitz test for stability, the denominator must have coefficients that are both nonzero and positive. It is clear from the transfer function it is impossible for the proportional value, \( K \), to be adjusted to achieve this. Therefore, the proportional controller was unstable due to the addition of the time delay factor.

In order for the ball to eventually come to rest and make the system stable, some damping must be added to the system. The traditional way of doing this is to add a PID controller.

\[
\text{Figure 4-PID controller}
\]

Then the transfer function for the system is

\[
\frac{X(s)}{Y(s)} = \frac{\alpha \left( I + Ps + Ds^2 \right)}{s^3 + \alpha \left( I + Ps + Ds^2 \right)}
\]

As the system is 2\(^{nd}\) order and so should have steady state error of 0, the transfer function can be simplified to

\[
\frac{X(s)}{Y(s)} = \frac{\alpha (P + Ds)}{s^2 + \alpha Ds + \alpha P}
\]

As there is a zero in the root locus, the behavior to be harder to control in practice as now the input is differentiated.
With the judicious use of a rate controller in place of the derivative, the zero could be eliminated from the controlled system. The proportional + rate controller was then

\[
\frac{X(s)}{Y(s)} = \frac{\alpha K_p}{s^2 + \alpha K_p s + \alpha K_p}
\]

This clearly eliminates the zero. Even more interesting then this elimination, the transfer function can be forced into the classic second order system form:

\[
\frac{X(s)}{Y(s)} = \frac{\alpha K_p}{s^2 + \alpha K_p s + \alpha K_p} = \frac{\omega_n^2}{s^2 + 2\zeta \omega_n s + \omega_n^2}
\]

where

\[
\omega_n = \sqrt{\alpha K_p}
\]

\[
\zeta = \frac{K_D \sqrt{\alpha K_p}}{2K_p}
\]

Since both \(K_d\) and \(K_p\) can be chosen, it was possible to arbitrarily decide upon the system’s \(\omega_n\) and \(\zeta\). In other words, the poles can theoretically be placed anywhere. In practice and within the discrete system, where the poles can be placed was restricted by both the amount of amplification possible and the sampling rate.
In an effort to make sure the proportional + rate controller did not have the same flaw as the discrete proportional controller, the Routh-Hurwitz test for stability test for this system was done. The transfer function for Figure 6 was

\[ \frac{X(s)}{Y(s)} = \frac{\alpha K_p e^{-T_d s}}{s^2 + \alpha K_D e^{-T_d s} s + \alpha K_p e^{-T_d s}} \]

And again, using the first order Padé approximation yields a polynomial transfer function of

\[ \frac{X(s)}{Y(s)} = \frac{\alpha K_p (-T_d s + 2)}{T_d s^3 + s^2 (2 - T_d \alpha K_p) + s (2 K_D \alpha - T_d K_p \alpha) + 2 K_p \alpha} \] (2)

It is possible for equation (2) to satisfy the Routh-Hurwitz criteria since \( T_d \) will always have a positive value as will \( 2K_p \alpha \). The only other remaining constraints are for both

\( 2 - T_d \alpha K_D > 0 \)

and

\( 2K_D - T_d K_p > 0 \)

Since this was possible, the proportional + rate controller can yield a stable system.

As the placements of the axes where orthogonal to each other, and both axes shared the same system equation, the same method of control can be used for both axes. As another consequence of the orthogonality of the axes, the control of the ball in each direction acts independently of each other, which allows for independence in terms of control.

**Implementation**

**Physical apparatus**

The basic physical platform consists of two 12” square pieces of particle board connected with a rod and a ball and socket joint. This allowed for unrestricted motion when tilting the board from its level position. In order to tilt the board in any direction, two servo motors needed to be attached. The placement of the servo motors was on two adjacent edge’s centers, as can be seen in Figure 7. The servo motors were then orthogonal to each with respect to a grid running the length and width of the plate.
As can be seen in Figure 7, the axes were laid out according to the placement of the servo motors. The positive direction of both the axes was arbitrarily defined to be towards the top right corner. With the layout of the axes as given, and the placement of the servo motors, the behavior of the ball in each axes could be controlled by only one servo motor. This was very advantageous, as it greatly simplified controlling the motion of the ball.

The servo motors were connected to the plate through a servo horn connected to the servo motor and then a threaded rod attached to the servo horn and plate with ball links. An unforeseen consequence of ball links freedom of motion was the ability of the plate to rotate about the z-axis for about a quarter turn in each direction. The initial design decision to reduce this motion to a manageable amount was to place two angle brackets on opposite corners, as indicated in Figure 8, restricting the amount of available rotation.

In order for the ball to be controlled, its location needed to be determined. One possible idea was using a touch screen device so the weight of the ball would press down hard enough to determine its location. Some advantages of this were that it would work in a variety of lighting conditions and little computational power would be needed, allowing for the possibility of implementing the controller with entirely analog components. This would eliminate the time delay factor, allowing a proportional control device to create sustained constant amplitude sinusoidal oscillations. Another method considered and ultimately decided upon was the use a web camera to take a continuous stream of images and from this determine the location of the ball. This method requires introduces time delays into the system, shifting the domain of the controller from continuous to discrete. The main reason a web camera was used as the measurement device was the desire to use and understand simple vision processing tasks.
A platform for the camera was then constructed, which consisted of a piece of wood attached vertically to the base of the plate and then another piece of wood attached in parallel to the plate's base. The camera was then attached to this stand so it was looking down vertically at the plate. An unintended but desirable consequence of how the stand was attached was that the plate could no longer rotate freely about the z-axis, which allowed for the removal of the angle brackets.

![Figure 8- Picture of complete apparatus](image)

*Angle bracket is marked with a black square*

### Calibration

Calibration was done for two different devices, the camera and the servo motors. To calibrate the camera, the center of the plate was marked and then the corresponding pixel value was found. To find the inch/pixel ratio, a black and white checkerboard pattern was placed on the plate and a snapshot was taken. The pixel values of several square’s corners were taken and then the difference in x, y pixels was calculated. The physical distance between each pair of points was measured with a ruler, and then the ratio of inch/pixel was calculated for each length segment followed by taking the overall mean. From these two values, the center pixel and the conversion ratio, the location of the ball on the plate can be determined in inches given a pixel value. Not only is this beneficial to physically check the location of the center of the ball, but it ensures the system constant has units of inch/sec² rather than pixel/sec². The resulting equations were

\[
\begin{align*}
x_{\text{Loc}}(\text{in}) &= - (x - 171) \ast 0.05625 \text{ in / pixel} \\
y_{\text{Loc}}(\text{in}) &= (y - 121) \ast 0.05625 \text{ in / pixel}
\end{align*}
\]  

(3)

Servo calibration preceded similarly for both of the servo motors. Using a program from the ENGR 5 [1], a variety of pulse widths were sent to the servo motors, causing them to rotate to different angles. The rotation affected the tilt of the board in either x or y direction. A positive
tilt angle was decided to cause a positive change in the displacement of the ball in the given axis so the signs would match with the Simulink model. Another decision, again in order to match the Simulink model, was to measure the angle in degrees and then convert to radians. Before the tilt angle was measure for the x-axis, the y-axis was zeroed using a level. Once this was done, different pulse widths were sent to the servo corresponding to the x-axis and the plate’s tilt was then measured. Using MatLab, a linear curve fit was found for this data. The process was then repeated on the y-axis.

From these equations, a MatLab program was written that would take in the servo number, the serial port information, and the desired angle and then output the corresponding pulse width to the desired servo motor. Because the measurements were imperfect, when a servo angle of zero radians was sent, the plate was minutely tilted enough to cause the ball to roll. With a level placed in the center of the plate, the needed adjustments of both equations’ x-intercepts were determined so an input angle value of zero would ensure correspond to a 0 radian tilt in the corresponding axis. The value of adjustment was small enough to not markedly affect the achieved tilt angle of the plate for a given input angle. The resulting equations relating the desired angle to the output pulse width (pw) were:

\[
pw = 3301*\text{angle} + 1395 \quad \text{X-axis servo}
\]

\[
pw = -3739.7*\text{angle} + 1430 \quad \text{Y-axis servo}
\]

**Software structure**

The programming language the controller was implemented in was MatLab. C++ was considered as another possibility because it would allow for greater speed and with OpenCV, computer vision processing tools. MatLab was ultimately decided to be used because the programming would be easier and graphical output and image processing would also be simple. In order to use a digital camera as the input sensor to the control program, two additional toolboxes for MatLab needed to be used, the Image Acquisition Toolbox and the Image Processing Toolbox.

When acquiring the image through the Image Acquisition Toolbox, there were two possible methods of grabbing image data from the camera. One way was to use the function `getsnapshot(vid)`, where `vid` was a video camera object and the function returned an image.
While this method was easy to use and implement, the problem was the time required. Whenever `getnsnapshot` was called, the camera would first startup, then take a picture, and finally turn off, resulting in a long time delay. Another method was to use a trigger, which is method of telling MatLab what to do with acquired images. The benefit of using a trigger was now a function could be called after a certain number of frames had been acquired by the camera without having the delay between starting and stopping the camera.

In order to take advantage of this, certain properties needed to be set. By setting the trigger properties to have an unlimited number of frames per trigger, the camera would take a continuous stream of images until it was stopped. The number of frames acquired before calling a function was set to one, although this was later adjusted when a GUI was made. The final parameter the needed to be set for the trigger was to call the function ‘control’ whenever the desired number of frames had been collected. ‘Control’ was the function created to implement the actual controller.

```matlab
set(vid,'FramesPerTrigger', Inf);
set(vid,'FramesAcquiredFcnCount', 1);
set(vid,'FramesAcquiredFcn', {'control'});
```

The overall flow of control of the program is depicted in Figure 11. First, main.m is called and initializes all the global variables and readies the camera for data collection and the serial port. Then it starts the camera. Whenever the camera takes a new picture, the function control.m is called. From here, ballFind.m is called with the image data and it and returns the location of the ball. Control.m then takes the location and determines the rate of the ball. Next, a median filter is applied to the past three data points of both location and rate. At this point, the derived control equations are applied to calculate the desired angles of the servos. The servo angles are then set. Control.m returns nothing and the program waits until the next picture triggers control.m. After a predetermined amount of time, main.m stops the video acquisition and then displays plots of relevant data. Before finally ending, it closes and deletes both the serial port handle and the video acquisition object.
Program initializes
  Setup servo motors
  Setup web camera
  Initialize variables
Start camera

Time elapsed <= desired runtime

Yes
control.m

Grab most recent image

control.m

Calculate rate
Apply median filter
Rate
Location
Determine and set angle of servos

ballFind.m

Find the location of the ball
  Edge Detection
  Dilate lines
  Fill image
  Remove edges
  Erode image (twice)
  Calculate centroids
  Determine pixel location of the ball
Convert pixel to inch value
Return coordinate of ball in inches

Main.m

Stop camera
  Stops collecting images
Close serial port
Close camera
Display plots as required

Figure 11-Program Flow diagram
Ball Detection

The algorithm used to detect the ball involves multiple assumptions. First, there is only one ball on the plate at any given time. This is a reasonable assumption as the system is intended to only control the motion of one ball at any given time. Second, the ball appears not to touch the edge when looking at it from above. While this assumption has some flaws, controlled paths of the ball may involve travelling close to the edge, the median filter is able to account for this. Third, the light conditions are such that most of the light is being cast from above, yielding little or no shadows cast on the plate. Such an assumption is valid in laboratory conditions, where the lights are on the ceiling and shades can be drawn on the windows.

Figure 12: Images from top left to bottom right: Original grayscale image. Sobel edge detection. Dilated lines. Filled holes. Edges removed. Eroded image. Final marked image.

The image that came into the ball detection algorithm, ballFind.m, was a grayscale image, Figure 12a. This is because the initialization parameters of the camera specified the image captured to be in grayscale rather than rgb. From this image, edges were found using the sobel operator. The sobel operator calculates the gradient of the given image magnitude for each point along both dimensions. The resulting vector, made by combing the previous gradients, is in the direction of the largest magnitude increase with length corresponding to how fast the magnitude is changing [2]. The resulting image is black and white with edges being marked, Figure 12b.
Next, the image is dilated, making the white edge lines thicker, Figure 12c. This is done to ensure the circumference of the circle has a solid line around it as sometimes after the edge detection is run there is not a solid perimeter around the radius of the ball. Next, flood fill algorithm was applied, Figure 12d. Since the ball had a solid line around it, there the circle is filled in.

In order to accurately detect which of the blobs in the image was the circle, more image processing needed to be done. Next, any object touching the edge is then removed, Figure 12e. This was done because the cables originating from the servo motor controller come out from underneath the board and enter the image and then leave. Finally, to get rid of any remaining small noise patterns on the board, the image was eroded twice, Figure 12f. Eroding decreased the size of the remaining objects and by doing this eliminated any small noise.

At this point, the ball was generally the largest object left in the image, but there was the possibility some other object could still be detected. To reduce this possibility, the centroid for each of the remaining objects in the image was calculated. Of these values, the area of each object was compared with the approximate area of the ball. If it fell within a given error boundary, the coordinates for the centroid with the largest error given these parameters was stored. Next, the function converted the pixel value of the centroid into inches and checked the x and y coordinates. If the calculated coordinate for either value was greater than 6”, then that value was set to 0. Otherwise, the coordinate values remained unchanged. Also, if no object was found matching these criteria, the coordinates 0,0 was used. Finally, the coordinates for the location of the ball were returned, Figure 12g.

As can be seen in Figure 12g, reprinted above, the exact center of the ball was not always returned even under mostly ideal conditions, in this case it was shifted to the left of the actual center. The reason for this was the shadow cast by the ball from the lighting in the room. Since this shadow was not even on all sides, the final centroid of the ball was shifted from its true value. As this shift was less than a quarter inch, no effort was made to correct for this.

**Median Filter**

As a first order derivative was used, noise was a problem. In addition to this, the location data sometimes would return a value far off from the actual location. In order to correct for this, a simple median filter was implemented on the position data and the derivative. The three most
recent data points were stored and the median of these items was returned. While this is not a linear filter and so would be more difficult to incorporate into a model, experimentally it sufficed and was better than a mean filter as sometimes noise appeared far enough away that the calculated mean would drastically differ from the actual mean.

As can be seen in Figure 13, the median filter on location was not always beneficial. The rate data though, which is computed through a first order derivative, was much more noisy, as can be seen in the original data, blue line Figure 13 right. After the median filter was applied, the data becomes much smoother as the large fluctuations are eliminated. The resulting curve was more sinusoidal, which was to be expected as the linear data is also sinusoidal.

**Angle Determination**

In order to determine the angle at which to turn the servo motors, the control loop, as seen in Figure 1 and Figure 5 had to be converted to a mathematical equation. For the proportional controller, this simply involved determining the error between the current position, found with the median filter and the desired position and multiplying this difference with $K_p$, the proportional constant.

$$\text{newAngle} = (\text{desiredLocation}(.;1)-\text{medLoc}(.;1)) \times K_p; \quad (4)$$

For the proportional + rate controller, a derivative was needed. Because the system is discrete, a first order approximation was used to calculate the derivative.

When implementing the control program, the derivative of the location, or the velocity of the ball, was determined using a first order approximation. The controller implementation involved taking the proportional part from equation (4) and adding to it the rate passed through the median filter multiplied with $K_d$, the derivative constant.

$$\text{newAngle} = (\text{desiredLocation}(.;1)-\text{medLoc}(.;1)) \times K_p - \text{medDeriv}(.;1) \times K_d; \quad (5)$$

**GUI creation**

After an adequate control program was implemented and tested, a graphical user interface was created to make interacting with the program easier.
As can be seen in Figure 14, the implemented GUI contains the necessary features for a simple demonstration. The Start button begins the operation of the program, which causes the ball to either move in towards the center or in a square pattern, depending on the selection made on the left hand side of the screen. The plot has an indication for both the desire position of the ball, the blue plus, and the measured position of the ball, red star. At each instance when the servo motors were moved, the plot was updated. When the user was finished with the program, the Stop button is clicked and the location of the ball is no longer controlled.

One issue that was encountered in the creation of the GUI was the response time. When a button was clicked, for instance Stop, the program would take several seconds before responding to the command. The reason for this was how often the trigger function for the camera was called. At first, after every new image the camera acquired, the frames acquired function was called. Because the control function took longer to compute than the frame rate of the camera, several instances of the control function would be waiting on MatLab’s operating queue at any given time. Once a button on the GUI was pressed, a new command would be added to MatLab’s queue and because there were several other commands preceding it, a delay was encountered before the desired command was issued. To account for this, the frames acquired function count value, the number of frames the camera needed to acquire before calling the desired function, was set to four. While this increases the time delay of the control loop, for the purpose of the GUI, it offers more immediate feedback since the command queue is drastically shortened.
Results

System Calibration

In order to accurately determine the system constant, \( \alpha \), several angles were given to ensure the system constant found was not dependent upon the input angle. For each input angle, the ball was placed in the extreme location so it would have the largest distance to move. For example, if the angle was +.01 rad along the x-axes, then the ball was placed at \( x = -5.5^\circ \). When determining the system constant, because only one dimension was being tested at a time, the ball was constrained to a single dimension through the use of two pieces of cardboard. Some problems which arose during the process from the ball detecting algorithm were the detection of an incorrect object and the not locating the ball since it was too close to an edge. After obtaining a good data set, as can be seen in Figure 15, a curve fit was done on the data to obtain the system constant.

\[ y = 2.46x^2 + 3.06x + 4.35 \]

\[ \text{Figure 15-Step response data along y-axis} \]

Red line: Quadratic curve fit
Blue line: Actual data

From equation (1), the system coefficient was related to the \( x^2 \) coefficient by

\[ K = \frac{2 \times C}{\text{angle}} \]

where \( C \) is the \( x^2 \) coefficient from the parabola curve fit

One problem that arose was the calculated system constant was vastly different from +angle to the –angle. The first time this occurred, a recalibration was done of the servo motors in order to ensure the angle input was proper. The second time it occurred, it was noticed the backlash of the servo motors connection was rather large and was caused by the loosening of the screw attaching the rod to the horn. This was fixed by simply using a nut and washer to securely fasten the rod to the horn, thus reducing, but still not entirely eliminating, the backlash.
After the system constant was calculated for 3 different trials each for 4 different angles, +.02, +.01, -.01, -.02, the mean was taken. With the process completed for the x-axis, it was a matter of simply changing the direction of the cardboard pieces to the y-axis and then changing which servo motor was being moved.

<table>
<thead>
<tr>
<th>Axis</th>
<th>System constant in/rad sec^2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-axis</td>
<td>229.5</td>
</tr>
<tr>
<td>y-axis</td>
<td>232.1</td>
</tr>
</tbody>
</table>

**System Testing**

When the one-dimensional system had a system constant, a proportion controller was designed to test the system and verify both the calculated system constant and the equation of the system were correct. The obtained results did not match with the theoretical prediction of sustained oscillations, instead the oscillations would start out small and then grow exponentially.

![Proportional Controller: Experimental Results](image.png)

*Figure 16-Proportional Controller: Experimental Results*

A key aspect had been left out of the theoretical model, as this was being controlled by a computer, the controller was not continuous but discrete. In order to model this in Simulink, a zero order hold was added to the model, resulting in Figure 3. Using Simulink to simulate the response of this system for a zero-order hold delay of .067, since the camera operates at about 15Hz, results in the graph depicted in Figure 17.
Figure 17- Discrete Proportional Controller: Simulink Results

Comparing Figure 16 to Figure 17 shows both result in a sinusoidal output with exponential growing amplitude. This supports the theoretical conclusion that the time delay caused the instability in the proportionally controlled system. While the amplitude grew at different rates for the Simulink and the experimental results, the difference was due to the zero-order hold. In the physical system, the total amount of time delay is a combination of the camera frame rate, the median filter, and the time the control program takes to operate.

As a test to see if the proportional + rate controller worked as expected in one dimension, a square wave input was set as the desired position and the motion of the ball was constrained to the x-axis. The square wave had amplitude of 3 with a period of 20 seconds.
As can be seen from the graph, the final resting position was not the desired position. Some possibilities for this were because of the neglect of friction and a minute change in the board tilt level did not always cause the ball to move. The calculated percent overshoot for Figure 18, using

\[ \%OS = \frac{c_{\text{max}} - c_{\text{final}}}{c_{\text{final}}} \times 100\% \]

was found to be on average 16.88\%. The theoretical percent overshoot was 9.98\%, found using

\[ \%OS = e^{-\left(\frac{\pi \sqrt{\frac{1}{\zeta^2}}}{\sqrt{1-\zeta^2}}\right)} \times 100\% = e^{-\left(\frac{5.914}{\sqrt{1-0.5914^2}}\right)} \times 100\% = 9.98\% \]

This results in a percent error of 69\%. Most of the error was due to the fact that the percent overshoot was calculated for a continuous second order system, while the actual system had the time delay.

Once the ball could be controlled to rest at the center and a square wave could approximately be followed in one dimension, the restriction to one dimension was removed and the ball was controlled in two dimensions. As the two dimensions control did not differ from the one dimension control, once the system constant was found for the additional dimension, the ball could be controlled to rest at the center. After this, the desired position of the ball was changed with time to trace out a simple figure. The tested shape for this process was a square. First, the ball was told to move to the center, (0,0), then to (-3,3), next to (3,3), next to (3,-3), then to (-3,-3), and then back to (-3,3) and final come to rest back at the center. Each of the desired locations
was held for 20 seconds before it was changed to the next one. This was simply tracing a square wave in 2 dimensions.

As can be seen in Figure 19 and Figure 20, 2 DOF ball balancing was achieved. While the exact resting condition was not achieved for each corner, the overall square structure was
achieved as well as controlling the motion of the ball. Figure 19 shows the overall path of the square wave trace and Figure 20 shows the motion of the ball plotted against time.

**Further Work**

Future work might be to increase the complexity of the model to more accurately match the system. Along with changing the model, techniques from digital signal processing could be used to more accurately model the system as well as get better response from the system using discrete techniques. Another possibility to increase the performance of the system, is to implement trajectory following to allow the ball to trace out paths such as circles accurately. In order to improve user interaction with the system, the GUI could be changed to allow for dynamic changing of $K_p$ and $K_d$, or alternatively, the $\omega_n$ and $\zeta$ constants. To improve the sensing technique, a Horwitz circle filter could be used as an alternative to the image processing techniques depicted in Figure 12. As an alternative method for sensing, the camera could be removed and haptic sensing could be used by measuring the torque on the servo motors and calculating the position of the ball from this measurement.

**Conclusion**

The goal of this project was to design and implement a solution to the 2-DOF ball and plate balancing. Over the course of a semester, a working apparatus was constructed and a controller implemented. The method for controlling the plate’s tilt was two servo motors attached to make an orthogonal coordinate plane. The sensing device was a web camera combined with image processing techniques to determine the location of the ball. Continuous and discrete controllers were explored after it was realized the system was discrete and was therefore unstable if only a proportional controller was used. The final implemented controller was a proportional + rate controller, which resulted in the system being controlled in the 2 dimensions.

**Bibliography**


**Acknowledgements**

I would like to thank Professor Cheever for the help and guidance he gave me throughout this project. Also, I would like to thank Grant Smith for the help he gave with the construction of the physical apparatus.
Appendix

Matlab Code

Main.m

clear all
%global variables
global s;
%global derivative;
global t0; %initial starting time of clock
global medianLocation;
global location;
global derivative;
global medianDerivative;
global angleStore;
global counter;
global desiredLocation;
global radius;

global Kp; %calculated from desired zetta and wn at start of program
global Kd;

%user defined variables
zeta=0.6; %damping factor %.15 low
wn=1; %natural frequency 1
alpha(1,1)=229.5305; %system constant x direction
alpha(2,1)=232.1167; %system constant y direction
totalTime=50;
%for some reason, program doesn't operate for given amount of time
%operates a little longer

%desired final position of the ball (in inches)
desiredLocation(1,1)=0; %x desired location
desiredLocation(2,1)=0; %y desired location
%
%initialize all required hardware (camera and serial port)
%serial port
s=instrfind; %Find any serial links (we can have only 1)
delete(s); %... and delete.
%Create a new serial communications link
s=serial('COM1','Baudrate',115200,'Terminator','CR');
fopen(s); %... and open it

%camera
vid=videoinput('winvideo');
vid.ReturnedColorSpace='grayscale'; %one less thing for function to do
src=getselectedsource(vid);

%set camera parameters
src.ZoomMode='manual';
src.PanMode='manual';
src.TiltMode='manual';
% depending on where the camera is placed:
src.Zoom=56;
src.Tilt=-10;

set(vid,'FramesPerTrigger', Inf);
set(vid,'FramesAcquiredFcnCount', 1);
set(vid,'FramesAcquiredFcn', {'control'});

% flatten both servos
angleServo(2,0,s);
angleServo(1,0,s);

disp('initialized');

%% initialize variables
% variables to store data
derivative=zeros(2,30*totalTime);
angleStore=zeros(2,30*totalTime);
medianLocation=zeros(2,30*totalTime);
medianDerivative=zeros(2,30*totalTime);
location=zeros(3,30*totalTime);
radius=zeros(1,30*totalTime);

counter=0; % used to count the number of times control is called

Kp=wn^2./alpha;
Kd=2*zeta*wn./alpha;

%% start program
start(vid);
t0=clock;
pause(totalTime);
stop(vid);
disp('done');

% eliminate excess 0s at end of arrays
leftZero=size(location(3,:),2);

% finds the left right most 0, which couldn't be a time
for i=size(location(3,:),2):-1:1
    if location(3,i)==0
        leftZero=i;
    end
end

location(:,leftZero:end)=[];
derivative(:,leftZero:end)=[];
medianLocation(:,leftZero:end)=[];
medianDerivative(:,leftZero:end)=[];
angleStore(:,leftZero:end)=[];
radius(:,leftZero:end)=[];
subplot(1,3,1)
%plot(time, location, 'r--');
%hold on
plot(location(3,:), medianLocation(2,:));
%hold off
xlabel('time (s)')
ylabel('displacement (in)')
title(['Proportion plus rate controller', 10, ' zeta=', num2str(zeta),'
 Wn=', num2str(wn)])
%legend('Without median filter','With median filter')

subplot(1,3,2)
% if max(derivativeStore)/max(medianDerivative) >=5
% plotyy(time, derivativeStore, time, medianDerivative); % else
% plot(time, derivativeStore, 'r--');
% hold on
% plot(location(3,:), medianDerivative(1,:));
% hold off
%end
xlabel('time (s)')
ylabel('derivative (in/sec)')
title(['Proportion plus rate controller', 10, ' zeta=', num2str(zeta),'
 Wn=', num2str(wn)])
%legend('derivative','median derivative')

% subplot(1,3,3)
% plot(location(3,:),radius);
% xlabel('time (s)')
% ylabel('radius')
% plot(location(3,:), angleStore(2,:))
% title('Servo #1 angle')
% xlabel('time(s)')
% ylabel('rad')

%% clean up opened hardware
angleServo(2,0,s);
angleServo(1,0,s);
delete (vid)
clear vid
close(s)
delete (s)
clear s
function control(obj, event)
%% Operates whenever a new frame is acquired from the webcam
%% It grabs an image and gets the event time
%% With this information it decides how much to tilt the plate

global s;

global medianLocation;
global location;
global angleStore;

global derivative;
global medianDerivative;
global counter;

global t0;
global Kp;
global Kd;
global desiredLocation;
global radius;

counter=counter+1; %increment number of times control has been called
im=peekdata(obj,1);
[location(1, counter) location(2,counter)]=ballFind(im);
location(3,counter)=etime(clock,t0);

if counter==1
    medLoc=location(1:2,counter);
    derivative(:,counter)=[0,0];
    medDeriv=zeros(2,1);
elseif counter==2
    medLoc=median(location(1:2,counter-1:counter),2);
    timDif=location(3,counter)-location(3,counter-1);
    derivative(:,counter)=...
        (location(1:2,counter)-location(1:2,counter-1))/timDif;
    medDeriv=median(derivative(:,counter-1:counter),2);
else %counter>=3
    medLoc=median(location(1:2,counter-2:counter),2);
    timDif=location(3,counter)-location(3,counter-1);
    derivative(:,counter)=...
        (location(1:2,counter)-location(1:2,counter-1))/timDif;
    medDeriv=median(derivative(:,counter-2:counter),2);
end

% trace out a square

t=location(3,counter);
if t<5
desiredLocation=[0,0];
elseif t<10
    desiredLocation=[-3;3];
elseif t<15
    desiredLocation=[3;3];
elseif t<20
    desiredLocation=[3;-3];
elseif t<25
    desiredLocation=[-3;-3];
elseif t<30
    desiredLocation=[-3;3];
else
    desiredLocation=[0,0];
end

newAngle=(desiredLocation(:,1)-medLoc(:,1)).*Kp-medDeriv(:,1).*Kd;

angleServo(1,newAngle(1),s); %adjust x servo
angleServo(2,newAngle(2),s); %adjust y servo
%radius(counter)=sqrt(medLoc(1,1)^2+medLoc(2,1)^2);
%angleStore(2,counter)=newAngle(2);
medianLocation(:,counter)=medLoc(:,1);
medianDerivative(:,counter)=medDeriv(:,1);

ballFind.m

function [xloc, yloc]=ballFind(Im)
    %takes the image file and outputs a 2x1 array with the of the x and y
    %location of the center of the ball
    %the image coming in is assume to be already in gray scale

    %edge detection using the sobel method

    [junk threshold] = edge(Im, 'sobel');
    fudgeFactor = 1.2;
    BWs = edge(Im,'sobel', threshold * fudgeFactor);

    %thicken the lines
    se90 = strel('line', 3, 90);
    se0 = strel('line', 3, 0);
    BWsdil = imdilate(BWs, [se90 se0]); %%make the lines thicker

    %fill in the holes
    BWdfill = imfill(BWsdil, 'holes');

    %
    %remove objects that are connected to the border as well as lighter
    %then the surroundings.
    %In this case, any white coloring touching the border.
    %A possible problem with this is the ball might move and in the image it
might touch the boarder.
BWNoborder=imclearborder(BWdfill, 4);

% erode the image
seD = strel('diamond',1);
BWseg = imerode(BWNoborder,seD);
for j=1:2
    BWseg = imerode(BWseg,seD);
end

% Find the center of the ball
% Find the centroid of the regions
% Centroid returns the weighed center of the areas, and the largest area
% should be the ball.
% A problem is that the ball's exact center may not be returned, since it
% is possible part of its shadow is connected to the outline of the ball.
L = bwlabel(BWseg);
region = regionprops(L, 'Centroid');
area=regionprops(L,'Area');
max=0;
reference=0;
for i=1:numel(area)
    if area(i).Area>max
        if area(i).Area>175 && area(i).Area<300 && area(i).Area>max
            max=area(i).Area;
            reference=i;
        end
    end
end
if reference==0 % couldn't find the ball
    xloc=0;
yloc=0;
else % found what is optimistically the ball
    [xloc1,yloc1]= ...
        pixelToInch(region(reference).Centroid(1),region(reference).Centroid(2));
    if xloc1>6
        xloc=0;
    else
        xloc=xloc1;
    end
    if yloc1>6
        yloc=0;
    else
        yloc=yloc1;
    end
end
function [xloc, yloc] = pixelToInch(x,y)

% Input: the x and y coordinates in pixels
% Output: x and y coordinates in inches (center of plate is 0,0)

xMiddle=171;
yMiddle=121;
scale=0.05625; %inch/pixel
% pixel difference
x=-(x-xMiddle); %camera is looking at it opposite
y=y-yMiddle;
% inch difference
xloc=x*scale;
yloc=y*scale;

angleServo.m

function angleServo(servoNum, angle, serial)
% Inputs: servoNum, angle (radians), serial port
% Output: The edge corresponding to the servoNum tilts to the angle
% In order to turn correctly, the function uses the linear data fit.
% Allowable inputs:
% servoNum: 1 , 2
% negative angle, rotate down
% positive angle, rotate up

time='T0';

if (servoNum==1)
    pw=3301*angle+1395;
    cmd=['#1P' num2str(pw) time];
    fprintf(serial,cmd);
elseif (servoNum==2)
    pw=-3739.7*angle+1430;
    cmd=['#2P' num2str(pw) time];
    fprintf(serial,cmd);
else
    error('Wrong Servo Num');
end
**Servo Calibration**

<table>
<thead>
<tr>
<th>X-axis</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pulse width (ms)</td>
<td>Angle (rad)</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>-0.086</td>
<td></td>
</tr>
<tr>
<td>1150</td>
<td>-0.073</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>-0.061</td>
<td></td>
</tr>
<tr>
<td>1250</td>
<td>-0.044</td>
<td></td>
</tr>
<tr>
<td>1300</td>
<td>-0.030</td>
<td></td>
</tr>
<tr>
<td>1350</td>
<td>-0.014</td>
<td></td>
</tr>
<tr>
<td>1400</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>1450</td>
<td>0.017</td>
<td></td>
</tr>
<tr>
<td>1500</td>
<td>0.031</td>
<td></td>
</tr>
<tr>
<td>1550</td>
<td>0.044</td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td>0.066</td>
<td></td>
</tr>
<tr>
<td>1650</td>
<td>0.079</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y-axis</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pulse width (ms)</td>
<td>Angle (rad)</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>0.079</td>
<td></td>
</tr>
<tr>
<td>1150</td>
<td>0.073</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>0.061</td>
<td></td>
</tr>
<tr>
<td>1250</td>
<td>0.044</td>
<td></td>
</tr>
<tr>
<td>1300</td>
<td>0.026</td>
<td></td>
</tr>
<tr>
<td>1350</td>
<td>0.017</td>
<td></td>
</tr>
<tr>
<td>1400</td>
<td>0.005</td>
<td></td>
</tr>
<tr>
<td>1450</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>1500</td>
<td>-0.017</td>
<td></td>
</tr>
<tr>
<td>1550</td>
<td>-0.035</td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td>-0.052</td>
<td></td>
</tr>
<tr>
<td>1700</td>
<td>-0.079</td>
<td></td>
</tr>
</tbody>
</table>
**System Calibration**

**X-Axis**

<table>
<thead>
<tr>
<th>angle=</th>
<th>-0.02</th>
<th>system Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>derivative</td>
<td>Slope</td>
</tr>
<tr>
<td>trial 1</td>
<td>-4.543</td>
<td>-2.5085</td>
</tr>
<tr>
<td>trial 2</td>
<td>-5.3132</td>
<td>-2.6382</td>
</tr>
<tr>
<td>trial 3</td>
<td>-4.9342</td>
<td>-2.792</td>
</tr>
<tr>
<td>average</td>
<td>-4.93013</td>
<td>-2.64623</td>
</tr>
<tr>
<td>angle</td>
<td>-0.01</td>
<td></td>
</tr>
<tr>
<td>trial 1</td>
<td>-1.3234</td>
<td></td>
</tr>
<tr>
<td>trial 2</td>
<td>-2.3319</td>
<td>-1.2661</td>
</tr>
<tr>
<td>trial 3</td>
<td>-2.4831</td>
<td>-1.2957</td>
</tr>
<tr>
<td>average</td>
<td>-2.4075</td>
<td>-1.29507</td>
</tr>
<tr>
<td>angle</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>trial 1</td>
<td>1.5211</td>
<td>0.76026</td>
</tr>
<tr>
<td>trial 2</td>
<td>1.8263</td>
<td>0.94654</td>
</tr>
<tr>
<td>trial 3</td>
<td>1.6416</td>
<td>0.79183</td>
</tr>
<tr>
<td>average</td>
<td>1.663</td>
<td>0.832877</td>
</tr>
<tr>
<td>angle</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>trial 1</td>
<td>4.4619</td>
<td>2.2953</td>
</tr>
<tr>
<td>trial 2</td>
<td>4.1107</td>
<td>2.2129</td>
</tr>
<tr>
<td>trial 3</td>
<td>4.2575</td>
<td>2.3291</td>
</tr>
<tr>
<td>average</td>
<td>4.2767</td>
<td>2.2791</td>
</tr>
</tbody>
</table>

= 229.5305  
unweighted average of the four trials
## Y-Axis

<table>
<thead>
<tr>
<th>angle= -0.02</th>
<th>system Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>derivative</td>
</tr>
<tr>
<td>trial 1</td>
<td>-4.7226</td>
</tr>
<tr>
<td>trial 2</td>
<td>-4.4742</td>
</tr>
<tr>
<td>trial 3</td>
<td>-4.9631</td>
</tr>
<tr>
<td>trial 4</td>
<td>-5.2051</td>
</tr>
<tr>
<td>average</td>
<td>-4.84125</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>angle= -0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>trial 1</td>
</tr>
<tr>
<td>trial 2</td>
</tr>
<tr>
<td>trial 3</td>
</tr>
<tr>
<td>average</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>angle= 0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>trial 1</td>
</tr>
<tr>
<td>trial 2</td>
</tr>
<tr>
<td>trial 3</td>
</tr>
<tr>
<td>average</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>angle= 0.02</th>
</tr>
</thead>
<tbody>
<tr>
<td>trial 1</td>
</tr>
<tr>
<td>trial 2</td>
</tr>
<tr>
<td>trial 3</td>
</tr>
<tr>
<td>average</td>
</tr>
</tbody>
</table>

=> 232.1167
unweighted average of the four trials