Automated Labyrinth

Niraj Desai, Christopher Riviere Escobedo, Patrick Murtha, Michael Sevilla

June 1, 2011

1Advised by Dr. Alexander Ihler
Executive Summary

The labyrinth is a game in which a player uses two knobs to tilt the board and guide a ball in the maze from start to finish. The goal of this system was to have a computer solve the labyrinth with speed and precision. The prevalence of inexpensive sensors and a desire for autonomous control of the labyrinth directed the project to have the system perform well with commodity sensors. The hardware of the system played a crucial role in monitoring the electro-mechanical subsystem and greatly affected the structure of the software. The software tools were chosen to save time and increase productivity. The hardware in the labyrinth project consisted of several different components that were chosen to be responsive and low cost. To control the ball from a computer the embedded system must translate electrical control signals from the computer into various board angles. To find the ball, we used template matching—a brute force algorithm that slides a template across the entire image until a match is found. The Kalman filter, a motion tracking algorithm, was used to help the system minimize noise. A proportional-integral-derivative control system was used because the underlying system of the labyrinth combined with the electro-mechanical subsystem introduced several unknown factors such as friction and delay. User interaction was facilitated by a graphical user interface, a Wii Nunchuck, and several software features. Unfortunately, a completely independent labyrinth solver was not achieved. This is because the mechanics underlying the labyrinth board were not linear, as required by the control system. However, we were able to create a quick and accurate ball balancing system.
# Table of Contents

1.1 Introduction .................................................................................................................. 1  
   1.1.1 Goal ....................................................................................................................... 1  
   1.1.2 Use Cases ............................................................................................................ 1  
      Automatic without Interaction ................................................................................ 1  
      Automatic with Interaction ..................................................................................... 1  
      Manual with Interaction ........................................................................................... 1  
   1.1.3 Evaluation Criteria ............................................................................................... 1  
1.2 Overview ....................................................................................................................... 3  
   1.2.1 Hardware ............................................................................................................... 3  
      Setup ......................................................................................................................... 3  
      Electro-mechanical Subsystem ................................................................................ 4  
   1.2.2 Software ............................................................................................................... 4  
      Git ............................................................................................................................... 4  
      OpenCV .................................................................................................................... 5  
      Xcode ....................................................................................................................... 5  
      Arduino .................................................................................................................... 6  
   1.2.3 User Experience ................................................................................................... 6  
1.3 Procedures .................................................................................................................... 7  
   1.3.1 Setup .................................................................................................................... 7  
   1.3.2 Hardware ............................................................................................................... 7  
      Labyrinth ................................................................................................................... 7  
      Electro-mechanical Subsystem ................................................................................ 8  
      Camera Sensor ......................................................................................................... 9  
      Embedded Processor ............................................................................................. 10  
      Power Supply ......................................................................................................... 11  
   1.3.3 Software ............................................................................................................... 12  
      Maze Analysis .......................................................................................................... 12  
      Target Detection ..................................................................................................... 13  
      Path Finding and the Kalman Estimator .................................................................. 14  
      Control System ....................................................................................................... 17  
      Target Advancement ............................................................................................... 19  
   1.3.4 User Experience ................................................................................................... 20  
      Graphical User Interface ......................................................................................... 20
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparency</td>
<td>21</td>
</tr>
<tr>
<td>Physical Input Device</td>
<td>21</td>
</tr>
<tr>
<td>Path Building</td>
<td>23</td>
</tr>
<tr>
<td>1.4 Results</td>
<td>23</td>
</tr>
<tr>
<td>1.5 Constraints</td>
<td>24</td>
</tr>
<tr>
<td>1.5.1 Budget</td>
<td>24</td>
</tr>
<tr>
<td>1.5.2 Manufacturability</td>
<td>24</td>
</tr>
<tr>
<td>1.5.3 Sustainability</td>
<td>24</td>
</tr>
<tr>
<td>1.6 Organization</td>
<td>25</td>
</tr>
<tr>
<td>1.6.1 Team</td>
<td>25</td>
</tr>
<tr>
<td>Niraj Desai</td>
<td>25</td>
</tr>
<tr>
<td>Patrick Murtha</td>
<td>25</td>
</tr>
<tr>
<td>Christopher Riviere Escobedo</td>
<td>25</td>
</tr>
<tr>
<td>Michael Sevilla</td>
<td>25</td>
</tr>
<tr>
<td>1.6.2 Skills Requirements</td>
<td>26</td>
</tr>
<tr>
<td>Bibliography</td>
<td>27</td>
</tr>
</tbody>
</table>
1.1 Introduction

The wooden Labyrinth game was first manufactured by BRIO® in 1946[5]. The labyrinth is a game in which a player uses two knobs to tilt the board and guide a ball in the maze from start to finish. BRIO® describes the Labyrinth as the perfect tool for developing motor skills and patience. Humans have a difficult time playing games of skill, concentration, perception, and patience so an automated solution logically progressed from our love of the original board game. However, computers excel at tasks requiring patience and motor skills. The prevalence of inexpensive sensors and a desire for autonomous control of the labyrinth directed the project to have the system perform well with commodity sensors.

1.1.1 Goal

While creating a computer capable of solving the Labyrinth, it is important to note that the automated Labyrinth solver demonstrates the magnificent capabilities of low-cost sensors and motors in conjunction with excellent software.

1.1.2 Use Cases

This project contains three important use cases, automatic without interaction, automatic with interaction, and manual.

Automatic without Interaction

Automatic mode, referenced as PID on in the graphical user interface (See Figure 1.13), starts the labyrinth solver.

Automatic with Interaction

The user selects points on the screen and the system will position the ball close to the selected spot. Another automatic mode is a build-a-path mode in which the user may create a five point path that the ball tracks.

Manual with Interaction

Manual mode allows the user to tilt the board using a physical input device. (See Section 1.3.4).

1.1.3 Evaluation Criteria

Evaluating the effectiveness of the labyrinth solver consists of both seeing the completion of the labyrinth maze from start to finish, and seeing the completeness of the various modes of manual interaction.
Figure 1.1: Overall Setup
1.2 Overview

An overview of the project’s technical aspects are covered in this section. Functional requirements are also covered in this section.

![Data Flow A/D Feedback Loop](image)

**Figure 1.2:** Data Flow A/D Feedback Loop

1.2.1 Hardware

The only requirement for the hardware to supplement the software in solving the labyrinth. As a subset of the entire system, the hardware played a crucial role in determining the structure of the software.

Setup

Designing a mount for the camera was one of the most difficult tasks during the requirements phase. The camera had to be perfectly centered on the board and had to provide enough focal distance to capture the entire maze when tilted and flat. To calculate the height of the mount above the labyrinth board, the focal aperture would be required; however, focal specifications are rarely bundled with inexpensive camera; therefore, a height-adjustable mount was settled upon. Also, to allow users to clearly view the labyrinth maze, a transparent mount was needed. Further design and construction outcomes are described in section 1.3.
Electro-mechanical Subsystem

A crucial component of the labyrinth solver was the electro-mechanical subsystem, consisting of two servo motors attached to a timing pulley system to accurately tilt the board to a specific degree. To specify this degree, an embedded system acted as the interface between the electro-mechanical subsystem and a personal computer. See Figure 1.2 for an overview of the electro-mechanical subsystem and the process by which it transferred crucial servo data.

1.2.2 Software

An abundance of software tools and systems are available for any project in this day and age. It was crucial to select the proper tools for this project, not only to save time but to increase productivity.

Git

Git is an open source version control system designed for group projects. The repository tool boasts speed and efficiency as a user is able to quickly branch and merge different versions of a project. In addition to using Git as a distributed development environment, the team used Git as a storage system [1].

Git was accessed through terminal using simple script commands to commit and checkout project versions. Publishing commits were monitored with cryptographic authentications of the projects history. In addition to the command prompt, the team used the GitX GUI to view the tree history storage and directory content (Figure 1.3).
OpenCV

Open Source Computer Vision (OpenCV) is an open source collection of C/C++ libraries with functions that are intended for real-time computer vision. OpenCV can run on Linux, Windows, and Mac OS X with interface support for Python, Ruby, MATLAB, and other languages. The OpenCV library has over 500 functions that aid in solving sophisticated computer vision and machine learning problems. Many fields of computer science and engineering can be influenced by the use of OpenCV such as product inspection, medical imaging, security, user interface, and robotics [3].

In this project, OpenCV was used for its computer vision libraries and functions. Template matching and Kalman filtering used data structures already present in the OpenCV library. The tool proved to be sufficient for real-time image processing; no noticeable latency or delay was observed and the control system efficiently interacted with the code.

Xcode

The Xcode 4 integrated development environment (IDE) is an environment and toolset for building Mac OS X projects and applications. The Apple LLVM compiler integrated into Xcode doubles as a parser and an agent to suggest solutions for mistakes. Live issues alert the user to syntax errors, which are symbolized using a red bubble.

All C/C++/Objective-C code was compiled, debugged, and run in the Xcode IDE. The
Xcode 4 design suite held the "servogui" project and launched the graphical user interface. The Cocoa application programming interface simplified the design of the interface.

Arduino

The Arduino platform is intended for hobby electronics enthusiasts. The Arduino microcontroller was programmed using Wiring in the Arduino IDE. Wiring is a C++ library that is packaged with the Arduino IDE. It contains simple functions for initial setup and main loop. The Arduino IDE conducts simple syntax checking and formatting.

1.2.3 User Experience

While creating a standalone labyrinth solver, it was important to consider the experience generated by the system onto the user. Two important tasks were considered to generate a positive user experience; a graphical user interface with available user input to the system and a graphical user interface with full computational transparency.
1.3 Procedures

Actions and procedures undertaken during the course of this project are covered here.

1.3.1 Setup

Design considerations required a transparent and height-adjustable mount for the camera as discussed in Section 1.2.1. To accomplish these two tasks, the mount was designed on a 20”x18”x0.75” solid wood baseboard. Four 22”x3/8” all-thread rods were used to create a height-adjustable design for the mount. One 14”x11”x0.093” sheet of acrylic was used as the mount for the camera. See Figure 1.1 for detailed design drawings and specifications.

1.3.2 Hardware

The hardware in the labyrinth project consists of several different components that were chosen to be responsive and low cost. Table 1.1 shows the main components of the system as well as the associated costs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Manufacturer</th>
<th>Cost</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labyrinth Board Game</td>
<td>Cardinal</td>
<td>$25</td>
<td>Labyrinth Maze</td>
</tr>
<tr>
<td>Web Camera</td>
<td>Gearhead</td>
<td>$15</td>
<td>Web Camera</td>
</tr>
<tr>
<td>Duemilanove Embedded</td>
<td>Arduino</td>
<td>$20</td>
<td>Embedded Development Board</td>
</tr>
<tr>
<td>Nintendo® Wii NunChuk</td>
<td>Hitec</td>
<td>$20</td>
<td>Physical input device</td>
</tr>
<tr>
<td>Servo</td>
<td>Hitec</td>
<td>$26</td>
<td>Servo motors</td>
</tr>
<tr>
<td>Timing Belts</td>
<td>SDP-SI</td>
<td>$15</td>
<td>Timing belts</td>
</tr>
<tr>
<td>MXL Timing Belt Pulley</td>
<td>Hitec</td>
<td>$27</td>
<td>Timing Pulley</td>
</tr>
<tr>
<td>Hardware</td>
<td>Misc.</td>
<td>$50</td>
<td>Mounting Hardware and Misc Equipment</td>
</tr>
<tr>
<td>Shipping &amp; Tax</td>
<td>Misc.</td>
<td>$15</td>
<td>Misc</td>
</tr>
</tbody>
</table>

Table 1.1: Bill of Materials

Labyrinth

A Cardinal Games wooden Labyrinth board was used as the subject (Figure 1.4). The labyrinth board was designed as a toy for young adults during short sessions of play. The short periods of use and low cost of the game dictate the construction of the game so the top playing field is made out of a light, low cost soft wood with a grainy surface. The original control system is made of a 1/8” metal rod inserted through the front and right side of the main box. On the metal rod, circular control knobs are affixed appropriately sized for small to medium sized hands. The ability to pitch and roll the top playing field is controlled via
a small nylon string attached to the bottom of the board. The left and right movement is generated by wrapping the string around the smooth metal shaft three to five times to generate enough friction while it rotates around its major axis.

![Maze of the Cardinal Games Wooden Labyrinth](image)

**Figure 1.4:** Maze of the Cardinal Games Wooden Labyrinth

**Electro-mechanical Subsystem**

To control the ball from a computer the system must translate electrical control signals from the computer into various board angles. The microcontroller receives commands over a USB connection from the computer and interprets those commands into a pulse width signal format that the servos expect.

Many different features and specifications are offered in the different servo lines. There is a trade off of cost vs. features in all the servos available for purchase. Common features are 180° sweep angle, torque, type of bearing, and type of controller. The servos used in this system is a commodity servo that is easily obtained and inexpensive; the sweep time is 0.14 seconds and the gears and has plastic bearings.

A typical servo has a full range of 180° with 0° being the neutral state. Pulse width modulation is a signal format that holds a positive voltage on the line finite amount of time and then returns to zero potential for the remainder of the 20ms frame. For example, to get a positive 90° angle the line voltage is held high for 2400µs and then held low for 17,600µs. Figure 1.5 displays a typical servo translation of degrees to pulse widths.
The first design of the system consisted of a rubber band rotating the original control knobs. This setup worked but had serious flaws due to the elasticity of the rubber bands; rubber bands stretch too much to have adequate control of a time sensitive process like playing the labyrinth. Due to the faults of the rubber band system, an alternative was required. A system with timing gears allowed for precise movement of the board. Timing gears utilize toothed belts that transfer rotational forces quickly and accurately.

The actual control from a typical packet coming out of the computer consists of a setup byte that tells the microcontroller which servo to move, followed by three bytes that are interpreted as degrees from 0 to 180. The microcontroller combines the three bytes into a degree value and then linearly scales the pulse width output to a corresponding value from 600µs to 2400µs.

Camera Sensor

The GearHead Quick 1.3 MP Night Vision camera (WC1300BLK) was chosen because it was cost efficient and it easily mounted on the acrylic stand. The camera has a 1.3-megapixel resolution and 640x480 video resolution, which provides enough resolution for the system. The camera also supported a built-in snapshot feature that was used for testing purposes and initial image processing simulations. The camera was placed on top of the acrylic stand and stayed stationary because the lens of the camera protruded through a hole in the acrylic. Once the camera was positioned with the entire board in view, it was taped to the acrylic sheet with electrical tape. The bolts on the all-thread rods were adjusted often because the reassembly of the system usually produced different setup variables. The usb cable of the camera was held down with plastic ties that were fastened through holes in the acrylic and was input to the MacBook.

Three different webcams were tested during the project. In early simulations, a Logitech® webcam was used but it had several issues: size (attached mount was intrusive), lack of
auto-focus (raw images were not sharp), and lighting. The GearHead proved to be a more sustainable camera compared to the Logitech webcam because the video stream was more focused and the camera was easier to mount. The LED lights, which were supposed to evenly light the board, lit the board inconsistently of specular highlights reflecting off the wood. The last camera, a Logitech HD Camera was seen as an upgrade over the GearHead but also had mounting issues and the high resolution made the software lag an unacceptable amount.

Embedded Processor

The Arduino DueMilanoove was the interface between the hardware and software in the system. The Arduino is an open source microcontroller that is low cost and uses an ATmega328 processor. The official software toolchain for Arduino is known as the Arduino Development Environment. It includes a cross-platform text editor and compiler for Arduino programs called Sketches. The sketch is uploaded from the computer to the USB-powered Arduino.

Before starting the Labyrinth system, the Arduino asks the user to select an operating mode, which can be either manual mode, with the Wii Nunchuk, or automatic mode. In automatic mode the application runs and solves a maze in real-time via image processing and control systems. Otherwise, the system is controllable via Nunchuck through its joystick and accelerometers.

The Arduino had two distinct tasks depending on the mode of the system. When the system is in automatic mode, the Arduino is responsible for receiving servo positions in the form of bytes from the MacBook. The Arduino then issued electrical pulses to the two hobby servos to physically tilt the labyrinth board. The Arduino was able to communicate with
the MacBook using an Arduino serial library. Serial communication is the process of sending data bit by bit. A 9600 baud (symbols/sec) rate was used since only a handful of bytes were needed per video frame. A faster baud rate would not help performance and instead would be wasteful in terms of energy and processing power.

When the system is in manual mode, the Arduino is responsible for translating the Nunchuk motion data into servo positions. The Arduino communication library established communication with the Nintendo® Wii Nunchuk. Similar to automatic mode, the Arduino continuously receives bytes from the Nunchuk and sends pulses to both of the servos.

The \( I^2C \) protocol is used to transmit bytes between the Nunchuk and Arduino. \( I^2C \), created by Philips, is a simple two-wire multi-master single slave standardized bus communication protocol. The SDA wire is for data and the SCL wire is for the clock. Each device has a unique address and can function as a transmitter or receiver. A device that is a master is one who initiates data transfers and generates the clock \[9\]. In the Labyrinth system, the master, or Arduino requested bytes from the slave, or Nunchuk device.

In both the automatic and manual modes, two bytes, that represent the ascii characters ~~, are used to reset the Arduino. Resetting the Arduino involves resetting the system to an initialization state where a user can decide whether the system will run in automatic or manual mode.

**Power Supply**

An adjustable power supply was required to accurately tune the servo system. A 9-Volt wall DC power supply supplied 1.0A to a custom-built adjustable voltage circuit which outputs 7.3-Volts with minimal amperage degradation.

A National Semiconductor LM317 3-terminal adjustable voltage regulator provided voltage regulation to the power supply circuit. To determine the resistance ratio, the following formula was used:

\[
V_{out} = 1.25V(1 + \frac{R_2}{R_1}) + I_{ADJ}(R_2)
\]
1.3.3 Software

Several commercial software components were used to effectively program the system software.

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIT</td>
<td>Open Source</td>
<td>git-scm.com</td>
<td>Source code management</td>
</tr>
<tr>
<td>OpenCV</td>
<td>Open Source</td>
<td>opencv.willowgarage.com</td>
<td>Image Processing Library</td>
</tr>
<tr>
<td>XCode</td>
<td>Apple Developer</td>
<td>developer.apple.com/tools/xcode</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>Arduino</td>
<td>Open Source</td>
<td>arduino.cc</td>
<td>Integrated Development Environment</td>
</tr>
</tbody>
</table>

**Maze Analysis**

Detecting the maze on the labyrinth board requires several finely tuned thresholding sequences. The first step is to convert the input into a grayscale image using RGB averaging, as so:

\[
Grayscale(R, G, B) = \frac{R + G + B}{3}
\]

Next, the image is segmented using a binary thresholding algorithm. This algorithm, built into OpenCV, allows the foreground to be distinguished from the background. In this particular situation, in the foreground lies the protruding walls of the maze, and in the background lies the objective: the holes and the maze path. The binary threshold is chosen for this particular case due to its accurate nature while selecting the $i$th pixel closest to the predefined threshold.

In order to detect the holes from a collection of lines, the contour detection feature of
OpenCV is used. This contour detection method uses mean-shift segmentation of Hough circles to detect hole density on the Labyrinth board.

Hough circles uses the parametric form of a circle to detect points along a specified radius, as so:

\[ x = x_0 + r \cos \theta \]
\[ y = y_0 + r \cos \theta \]

Target Detection

A fast and accurate ball detector was necessary to keep track of the target as it moves across the game board. The ball detection method was an implementation of the techniques found in the technical paper, [4]. Template matching is a brute force algorithm that slides the template, which in our case was the 20x20 pixel template of the ball, across the entire image until a match is found. If the image is too large then this algorithm can take a long time to complete. To minimize the number of computations required for ball detection, the OpenCV libraries include a feature called Region of Interest (ROI). This useful feature allows the algorithm to designate a sub-region within an image to operate on. This reduced the search space from a resolution of 640x480 down to a resolution 80x80. ROI provides a great increase in speed while performing the algorithm and allows the ball tracker to work in real-time.

A confidence measure was used to evaluate how well a section of the image matched the ball template. The confidence measure was the average of all the pixels in each channel of the RGB ball template. Sum of squared error (SSE) was used to compare the current ball template with the new matching portion of the video image. The bounds for the confidence
measure were determined experimentally by running the template matching program on a video of the ball moving through the maze. The maximum SSE was recorded and tweaked so that the values were large enough to track the ball during fast motion, but small enough to prevent the ball tracker from matching a hole or other portions of the Labyrinth board.

The OpenCV template matching method used was the Normalized Correlation Coefficient since it is better adapted at distinguishing between lighting effects. To account for changes in lighting, dynamic template matching was used in the form of a simple IIR filter. The filter the paper uses is,

\[ \text{NewTemplate} = \text{PreviousTemplate} \times \alpha + \text{bestNewTemplateMatch} \times (1 - \alpha). \]

In this system, alpha was set to 0.85 and the OpenCV function `addWeighted()` was used to make the updates. Alpha was set to a high value because the image of a ball should not change drastically while the system is running. Dynamic template updating was necessary as the ball moved through the Labyrinth board. Differences in lighting caused the template matching function to incorrectly match with holes instead of the ball. The template matching function in OpenCV was able to correctly match the ball in all function invocations.

**Path Finding and the Kalman Estimator**

The Kalman filter is a motion-tracking algorithm that averages actual measurements and estimations to predict motion. The filter was used in this project to help the system minimize noise. Weighting the estimation and the measurement of the ball’s location helped stabilize the system by eliminating noisy readings. In the system, the Kalman filter’s predictive capabilities assisted the computer vision processing and the control system.

The Kalman filter is calculated using the ball’s previous position, velocity, and acceleration. A system model is calculated and updated at each frame so that the system has an idea of how the ball is behaving within the specified physical constraints. Since the tracking is constantly changing (dynamic) the Kalman was ideal for modeling the system and reducing noise based on what we already know about the Labyrinth and the way it works. This technique is known as ”data fusion” [3].

The Kalman filter uses measurements over time to track a moving object in the real world. The running state matrix averages the current measurement matrix and all previous state matrices by weight so that an idea of the motion of the object can be tracked. Weight is assessed by certainty, with high certainty events given more weight. Noise constants can be implemented and different errors factors can also be factored into the system to help emulate the real world system (i.e. coefficient of friction of the board, string tensions, image processing errors, etc.). The Kalman filter is fast and efficient because its multiplications are linear and can be based on simple techniques, such as Markov chains. Its speed and efficiency made the Kalman filter ideal for the system because latency was reduced and image processing
and response calculation from the control system could be done in real time.

The Kalman Filter tracks a model based on the physical properties of the system. Three assumptions must be satisfied in order to apply the Kalman Filter to the system:

1. System must be linear: can be modeled by a multiplication of matrices
2. Noise must be white: not reliant in time
3. Noise must be Gaussian: model is statistically based on average and covariance

The equations for the Kalman filter are based off simple physical motion and calculus. The variables for the system are listed for convenience:

\( F_k \): state-transition or transfer matrix at \( k \)
\( x_k \): state of the system at \( k \)
\( H_k \): observation or measurement matrix at \( k \)
\( Q_k \): covariance of process noise at \( k \)
\( R_k \): covariance of observation noise at \( k \)
\( B_k \): control-input model at \( k \)
\( u_k \): control inputs
\( w_k \): process noise assumed to have Gaussian distribution \( \mathcal{N}(0, Q_k) \)
\( Q_k \): covariance matrix

The computation used to update the matrices at each frame (or loop iteration) is:

\[ x_k = F_k x_{k-1} + B_k u_k + w_k \]

In the filter design for the automated labyrinth, the control-input model and the control inputs were ignored \[3\].

The important thing to note is that each state, \( x(t|t) \) is based of the measurement and the uncertainty, \( S(t|t) \). If the ball is not found, the location of the ball must be predicted at time \( t + 1 \) given no measurement and using only \( x(t|t) \) and \( S(t|t) \). This predicted, or ”pre”, state is the predicted location of the ball given the position of all previous measurements of the ball up to time \( t \).

The Kalman filter was implemented by defining the initial parameters, specifying the noise parameters, updating the filter, providing the data as input to the control system, and reporting the results. Each step was embedded in the code and varied depending on whether the ball could be located using template matching.

Matrices for the Kalman filter were created based on the parameters that would be tracked
in the system. In the system, the dynamics of the ball and system were tracked by the
dynamic parameters: horizontal position (x), vertical position (y), horizontal velocity (dx),
and vertical velocity (dy). Since four variables were tracked, the dynamicParameters was
set to "4". The dimension of the planar space also needed to be specified as the measure-
ment parameters for the instantiation of the Kalman filter. Since the Labyrinth board lies
in 2-D space with a constant width and height, the measurementParameters was set to
"2". Calls to cvCreateKalman() and cvCreateMat() instantiated the Kalman filter and
took the dynamicParameters and measurementParameters values as parameters.

The function calls instantiated a measurement matrix and a state matrix. The measure-
ment matrix was based of the data from the web cam. The state matrix was created based
on the current measurement matrix and all previous states.

The noise parameters were specified through the process noise covariance (process_noise_cov),
measurement noise covariance (measurement_noise_cov), and post calculation error noise
covariance (post_noise_cov). The process noise covariance is the amount of uncertainty for
the entire system at each time step. The measurement noise covariance and the post cal-
culation error noise specify the noise ranges for incoming and outgoing data to the system.
All these covariances were modeled off Gaussian curves. The values were adjusted according
to the control system’s reactions. The following values represented the best results for the
Labyrinth system.

<table>
<thead>
<tr>
<th>Covariance</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Noise</td>
<td>[1,1,15,15]</td>
</tr>
<tr>
<td>Measurement Noise</td>
<td>1</td>
</tr>
<tr>
<td>XCode</td>
<td>1000</td>
</tr>
</tbody>
</table>

![Figure 1.9: Kalman Noise](image)

On each pass through the image processing loop, the Kalman filter state was updated to
account for the location of the ball. Template matching reports the location of the center of the ball and stores the value in the measurement field. `cvKalmanCorrect()`, which took the Kalman filter and the measurement matrix as parameters, adjusted the state matrix and the current state and stored the value in `kalman` to `state_post`. A prediction matrix is also updated so that the system knows whether the ball is moving or not.

If the ball is not found, the system relies entirely on the Kalman filter for the location of the ball. In this scenario, the pre-state matrix is copied to the post-state matrix because the system assumes the ball continues in the same orientation and with the same speed as before. The region of convergence is also relocated based on the predicted ball location.

The proportional-integral-derivative system (Section 1.3.3) was implemented using the posterior Kalman filter states because the Kalman filter more accurately accounted for ball position using noise and measurements. The noise covariances that were specified at declaration help reduce the noise of the measurements because they add a degree of certainty to measurements. For example, if the reading of the ball is far from where the Kalman filter predicts the location of the ball, it is safe to assume that the measurement is noisy.

Observable output of the Kalman filter can be seen on the graphical user interface, or GUI (Section 1.13). Different colored vectors and circles were drawn on the real time image to represent different measurements.

These vectors are clearly demonstrated in Figure 1.11.

- **Red prediction dot**: the position of the ball estimated by the predicted matrix in the Kalman filter. This was only used for debugging so it was left out of the deliverable version of the system.

- **Yellow velocity vector**: the velocity of the ball at the current frame. The magnitude and direction of the vector shows speed and direction of the ball in the system.

**Control System**

A proportional-integral-derivative (PID) controller was crucial to supporting accurate loop feedback control. Because the underlying system of the labyrinth combined with the electro-mechanical subsystem (embedded system, servos, and the timing pulley) introduced several unknown factors such as friction and delay, PID was the best option.

The PID controller algorithm is as follows:

\[ u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t) \]

The PID controller required the use of three, heavily integrated constants represented in
the function above as $K_p$ or the proportional constant, $K_i$ or the integral constant, and $K_d$ or the derivative constant. The equation above can be described as the total sum of the weighted current error, the weighted sum of previous errors, and the weighted difference of the current error with the previous error.

The following table describes how the PID can be used effectively as a part of the labyrinth solver.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Definition at current time $i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_p$, proportional</td>
<td>Distance between the path target and the ball at time $t_i$</td>
</tr>
<tr>
<td>$K_i$, integral</td>
<td>Sum of the distances between the path target and the ball from time $t_0$ to $t_i$</td>
</tr>
<tr>
<td>$K_d$, derivative</td>
<td>Difference between $t_i$ and $t_{i-1}$</td>
</tr>
</tbody>
</table>

The following table demonstrates the effects of altering each constant independently. All information was gathered during the manual tuning phase on the labyrinth and is not representative of all PID-controlled systems.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_p$, proportional</td>
<td>Promotes precision</td>
<td>May cause instability</td>
</tr>
<tr>
<td>$K_i$, integral</td>
<td>Decreases noise &amp; response time</td>
<td>Increases overshoot</td>
</tr>
<tr>
<td>$K_d$, derivative</td>
<td>Reduces Overshoot</td>
<td>Increases noise &amp; response time</td>
</tr>
</tbody>
</table>

Manual tuning of the PID controller required extreme caution as the upset of one constant could disrupt the entire system unknowingly. To track forward progress, the PID was tested on a flat surface attached on top of the labyrinth maze to create an obstruction free surface. The PID was tuned using the position of the ball and the position of a single target on the flat surface. The distance between the ball and the target was referred to as the error term.

Because the mechanical subsystem was separated into two axes (and servos), $X$ and $Y$, two separate PID algorithms were used. Coincidentally, further research reported that the identical weighting constants $K_p$, $K_i$, and $K_d$ for both $X$ and $Y$ were sufficient.

After significant amount of tuning, the PID values that represented the system are as follows:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_p$, proportional</td>
<td>14</td>
</tr>
<tr>
<td>$K_i$, integral</td>
<td>0.12</td>
</tr>
<tr>
<td>$K_d$, derivative</td>
<td>200</td>
</tr>
</tbody>
</table>
Target Advancement

The target advancement algorithm was a simple algorithm derived from typical path advancement algorithms. Once the ball is within a confidence range from the target, the current target is achieved and a new target is set. This new target is simply the next point in the sequence of path points.

By combining the PID controller with this advancing target algorithm, the system is complete. Figure 1.12 shows the importance of the target advancement algorithm in the overall system.
1.3.4 User Experience

The experience generated by a standalone labyrinth solver was an important consideration during the project design and implementation phases. The resulting interaction scenarios are covered in this section.

Graphical User Interface

The most important interaction component of the labyrinth solver was the graphical user interface, or GUI. The GUI allowed user input to physically test and interact with an otherwise isolated system. The GUI was created using Apple’s Cocoa API and Apple’s integrated development environment, Xcode, in both Objective-C and C++ programming languages.

As the core of the entire system, the GUI controlled and allowed input to every aspect of the system from receiving and processing input from the camera (Section 1.3.3), calculating the necessary servo tilt (Section 1.3.3), and sending serial output to the embedded system.
As the hub of the entire system, the software received, processed, and sent almost every piece of data necessary for the system to function. As this data was freely available within the code of the software, it was an important design decision to output this data to the GUI. In some cases, it was also important to allow this data to be user editable, such as the values for PID.

Debugging information was prevalent in the GUI, including measurements such as PID calculations, physical degrees of the servos, and intensive path data for the various path modes available.

**Physical Input Device**

The graphical user interface included a button to allow manual control of the labyrinth using a Nintendo® Wii Nunchuk controller. The Nunchuk controller features a joystick and an accelerometer, both of which were used for manual control of the Labyrinth.

The Wii Nunchuk fed joystick and accelerometer data directly into the Arduino embedded system. The Arduino then processed that data and fed servo coordinates directly to the servo. This data was also sent back to the GUI for means of debugging.
The Nunchuk is a proprietary device, therefore most of the specifications were found through reverse engineering. The following Nunchuk specifications were listed by a popular Wii homebrew community wiki called WiiBrew.org [7]. The Wii Nunchuk is an accessory that plugs into a 6-pin expansion port located at the bottom of the Nintendo® Wii Remote. This expansion port allows bidirectional synchronous $I^2C$ communication between remote and Wii accessories. Wii extension peripherals have a slave address of 0x52. The Nunchuk has two membrane switches that act as the C and Z buttons. Each axis of the joystick uses a 30KOhm potentiometer. The exact model of the Nunchuks accelerometer and microcontroller is unknown. The Nunchuk has a three-axis accelerometer. The accelerometer model is in the LIS3L02 series from STMicroelectronics. The Wii Nunchuk requires a 3.3-Volt power supply to work properly [7].

The Wii Nunchuk is connected to the Arduino by using a small PCB adapter created by Tod E. Kurt called the WiiChuck. This adapter plugs into the 6-pin expansion port of the Nunchuk. This PCB exposes the ground, power, SCL, and SDA pins and avoids the need to cut the wire cable open. The figure on the left shows the pins on the Wii Nunchuk adapter [7]. The figure on the right shows the WiiChuck connected to the Arduino board, with pins highlighted in different colors [8]. On the Arduino Duemilanove board, analog input pin 4 is the $I^2C$ SDA pin and analog input pin 5 is the $I^2C$ SCL pin.

Figure 1.14: WiiChuck

Figure 1.15: Wii Nunchuk
Path Building

A build-a-path feature was included in the GUI to provide another means of manual interaction. This feature, activated by the click of a button in the GUI, allows the user to place five targets on the flat surface of the labyrinth and watch the ball move to each target quickly and accurately.

Figure 1.16: Build-A-Path

1.4 Results

The initial objective of this project was to create a cost-efficient automated labyrinth. Unfortunately, due to the underlying mechanics of the labyrinth, an efficient proportional-integral-derivative controller was not achievable. The mechanics of the labyrinth, including the wire-equipped tilt mechanism, were not sustainable for the constantly shifting tilt scenarios enacted by the electro-mechanical subsystem. However, the project did generate a positive user experience through the graphical user interface and its build-a-path feature, which accurately guided the ball to each of the five points specified by the user, quickly and accurately.
1.5 Constraints

Several constraints were considered prior to designing the system. These constraints included budget requirements, manufacturability requirements, and sustainability requirements.

1.5.1 Budget

This project was intended to be completed under a budget of $100 per research member. Due to this constraint, it was imperative that we create a sustainable labyrinth solver using inexpensive hardware components and compensate for these components using inexpensive software.

1.5.2 Manufacturability

This project consisted of off-the-shelf hardware components that can be easily combined and manufactured in high quantities. The software was written as any indie developer would write software, through Xcode in highly portable C/C++/Objective-C. The Arduino embedded system is easily manufacturable as it is open source and its schematics are readily available on the Internet.

1.5.3 Sustainability

This project was built using several hardware and software components. The manufacturing process for these hardware and software components does contain hazardous chemicals; however, these chemicals are generally regulated by organizations and are not typically exposed to users.
1.6 Organization

The research and development team as well as their distinct skill sets are present in this section.

1.6.1 Team

The team consists of four highly motivated engineering students.

<table>
<thead>
<tr>
<th>Name</th>
<th>Major</th>
<th>Concentration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Niraj Desai</td>
<td>Computer Science &amp; Engineering</td>
<td>Embedded Systems</td>
</tr>
<tr>
<td>Christopher Riviere Escobedo</td>
<td>Computer Science &amp; Engineering</td>
<td>Computer Graphics</td>
</tr>
<tr>
<td>Patrick Murtha</td>
<td>Computer Science &amp; Engineering</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>Michael Sevilla</td>
<td>Computer Science &amp; Engineering</td>
<td>Embedded Systems</td>
</tr>
</tbody>
</table>

Niraj Desai

Niraj was responsible for team planning and organization. He also provided significant research towards image recognition, embedded systems, and user experience scenarios. Niraj also expanded upon the PID control algorithm to include target advancing as a means of automation.

Patrick Murtha

Patrick was responsible for project planning and organization. He was the driving force behind the electromechanical subsystem and research direction. The initial GUI code with video feedback and serial communication was written by Patrick. Patrick wrote the PID implementation and also proposed transparency as a means of generating positive user experience.

Christopher Riviere Escobedo

Chris worked on the embedded processor hardware and target detection. Chris wrote the Arduino interface source code to connect the MacBook to the servos via USB protocol. Chris worked on the Wii Nunchuk and WiiChuck connections to provide manual control. Chris also implemented the ball detector program.

Michael Sevilla

Michael was responsible for significant intelligence algorithms, including the Kalman filter implementation of the labyrinth solver. He was also instrumental in developing a sustainable electro-mechanical subsystem to support the overall software implementation.
1.6.2 Skills Requirements

- Carpentry and Basic Construction
- Electronics
- Image Processing
- Signal Processing
- Makefiles
- Distributed Revision Control System
- C++ Programming
- Cocoa Programming
- Control Theory
Bibliography


