Ariana Haghighi (ah677), Jeff Nan (jjn48), Tiffany Chou (tlc234)

04 November 2022

ECE 4760: Digital Systems Design Using Microcontrollers

Professor Adams

Lab 3 Report: PID Control of an Inverted Pendulum with a Reaction Wheel

I. Introduction

An inverted pendulum has an equilibrium point that is inherently unstable. However, it can be balanced by applying the appropriate amount of torque. One way to do this is using a reaction wheel by spinning the wheel and generating conserving angular momentum for the pendulum-wheel system. In this lab, we balanced an inverted pendulum by collecting sensor data from the pendulum, processing that data on the Raspberry Pi Pico 2040 microcontroller, and rotating the wheel via a servo motor. The sensor contained both an accelerometer and a gyroscope which measure acceleration and angular velocity respectively. From the sensor data, we estimated the pendulum's angle using a complementary filter on the microcontroller. This filter low-passes the accelerometer measurements and high-passes the gyroscope measurements, such that the estimated angle is both accurate and stable for long periods of time. To actually move the wheel, we implemented a PID Controller with a servo motor. The servo is driven by an H-bridge motor driver which controls the servo at a lower level. This controller makes adjustments to the motor speed proportional to the pendulum's angle, angular velocity, and integral gain. To adjust the importance of each of these factors, a serial interface was implemented so the user can easily change them while the program is running. This serial interface was implemented using UART. Additionally, we graphed the pendulum's angle (actual and estimated) and the motor's duty cycle on a VGA screen for debugging purposes. This lab introduced mechatronics topics and explored how microcontrollers can interface with and control the physical world.

II. Design and Testing Methods

A. Concept

The complementary filter takes the best parts of accelerometer and gyroscopes and filters out the disadvantages of each of them. The gyroscope is very accurate for short periods of time, but accumulates a large bias over longer periods of time. The accelerometer is bad at filtering out

noise (like noise generated by the motor), but has zero mean so it does not accumulate large bias. At a high level the complementary filter is high passing the gyroscope and low passing the accelerometer in software. To implement this filter, we first collected data from an accelerometer and gyroscope mounted on the arm. Then we approximated the accelerometer angle by dividing the horizontal measured acceleration by the vertical. Next, we found the change in angle that the gyroscope measured. This difference was calculated by multiplying the raw measurement by the timestep (found by trial and error). Lastly ,we took a weighted average of the accelerometer angle and added the sum of the angle estimate with the change in angle to get our final angle estimate.

Control of the motor is done through the Pico's PWM module, which produces a square-wave-like waveform given a certain duty cycle, which approximates some average DC voltage based on the duty cycle. Thus, by updating the duty cycle, we can choose the voltage sent to the motor, which in turn determines the speed. Internally, the PWM is a number that counts up linearly (incrementing every time step) until some max value is reached (in our case, 5000), at which point it wraps back to 0. It outputs a HIGH voltage when the counter is below the defined duty cycle (a number between 0 and 5000), and LOW otherwise.

Ultimately, the pendulum-reaction wheel system is balanced using PID Control which stands for proportional, integral, and derivative control. To implement this algorithm, first determine the angle error. The desired angle for the pendulum is straight up, which we defined to be 0 degrees. The difference between the desired angle and estimated angle (from the complementary filter) gives the error. To implement proportional control, set the duty cycle of the motor to be the error multiplied by some scalar Kp. The value of this scalar needs to be determined through trial and error. Next, implement integral control by calculating error accumulation. During each iteration of the controller, the error accumulator gets incremented by the error. Note that the error can be negative or positive depending on the pendulum's position. The duty cycle is updated to consider the scaled error as well as the scaled error accumulation. This scalar Ki needs to be determined through trial and error, however a good starting point is a few orders of magnitude smaller than Kp. Lastly, derivative control is implemented by accounting for the rate of change of the error. The derivative error is calculated by subtracting the previous error from the current error. The duty cycle is updated to consider the scaled error, scaled error accumulation, and scaled derivative error. This scalar Kd needs to be determined

through trial and error, but should be a few magnitudes larger than Kp. The PID controller is integral to stabilizing the pendulum-reaction wheel system.

B. Implementation – Hardware

The RP2040 board is powered by the computer through the USB-C Peripheral. On the breadboard, we connect the RP2040 to the VGA port to display the boids as follows:

- GPIO pins 16 and 17 are used for the HSYNC and VSYNC, respectively. HSYNC and VSYNC are the horizontal sync and vertical sync.
- Pin 23 on the RP2040 is used to connect the Raspberry Pi ground to the VGA ground
- GPIO pins 18, 19, and 20 are used for red, green, and blue, respectively, which control the RGB color inputs to the screen.

Additionally, we allow for user input with a serial interface. A USB port is connected to GPIO pins 0 and 1 for TX0 and RX1, respectively. The USB port is plugged into the computer, and allows the user to type their inputs into the keyboard connected to the computer. Additionally, the ground from the USB is connected to the ground on the RP2040. To set up the serial interface, we use PuTTY with a baud rate of 115200.

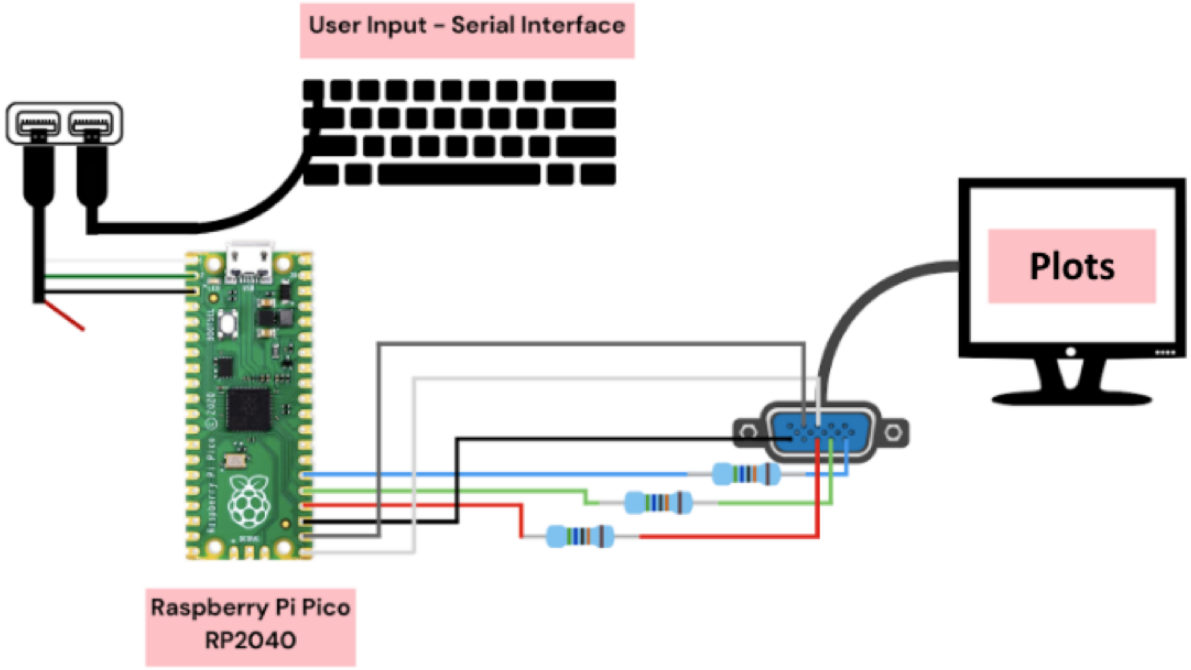

Figure 1. Wiring diagram, showing connections between the RP2040 and the VGA display and USB cable.

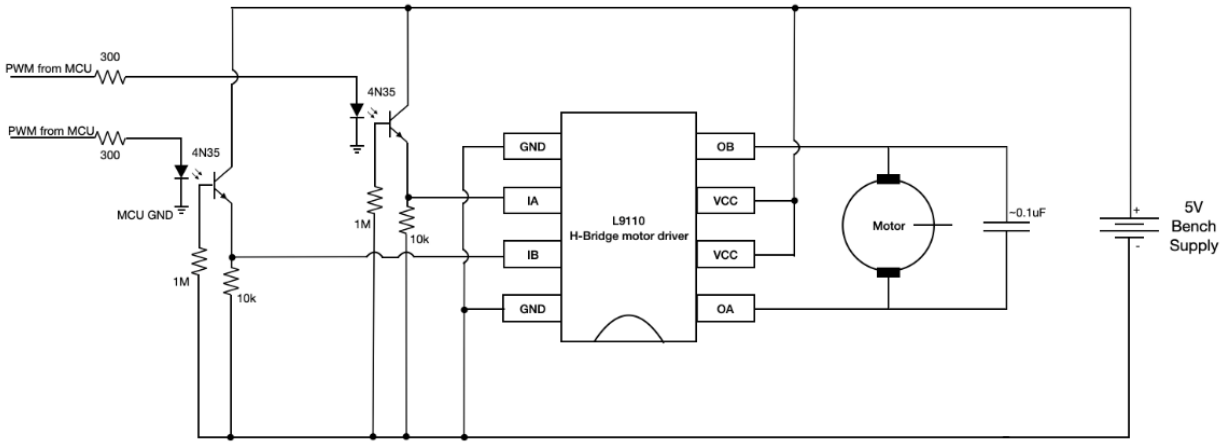

Figure 2. Schematic of the motor H-Bridge circuit.

If the Pico's PWM outputs were directly connected to the motor, there would be a number of problems, including high noise and high voltage spikes going into the microcontroller. Therefore, to properly control the motor, a circuit (Fig. 2) is built around the motor's H-bridge driver. This circuit has a few functions:

- The H-bridge L9110 driver allows high-voltage spikes coming off the inductive motor elements to go to ground, protecting any susceptible devices.
- 4N35 opto-isolators isolate the Pico from the motor, which protects the board from any high voltages, and allows the motor to be independently powered by an external DC power supply, which is good for configuring input voltage.
- A capacitor is placed in parallel with the motor to filter out high-frequency noise signals being sent to the motor.

To measure the tilt angle of the arm, an MPU6050 device is attached to the arm, which contains both a 3-axis accelerometer and 3-axis gyroscope. These two sensors, as described above, are used in the complementary filter. The MPU6050 is wired to the Pico's GPIO pins (8 and 9), and communication occurs between the devices using the Pico's I2C protocol peripheral.

C. Implementation – Software

Calculation of the complementary filter output and PID control all happens in an ISR which runs when the PWM module wraps from its max value. The output of the complementary filter gives an angle, which the PID controller uses in its calculations.

First, readings are made from the MPU6050 to get the accelerometer and gyroscope measurements, and some computation is done on the readings to get the useful data (angle of acceleration from the accelerometer's horizontal and vertical acceleration values, and change in gyroscope angle from its angular velocity measurement). Then, the complementary filter is computed:

```
complementary_angle = multfix15(complementary_angle - gyro_angle_delta,
zeropt999) + multfix15(accel_angle, zeropt001);
```

Figure 3. Code Snippet of the complimentary angle calculation.

This code assigns a weight of 0.999 to the gyroscope angle, 0.001 to accelerometer angle, and implements feedback to the gyroscope input. The output of the complementary filter is used as the arm's current angle, to be used in PID.

To do all of these calculations, as well as to express the results in useful units (we use degrees in fixed point), we have a few fixed point constants defined, including the numbers 0.999, 0.0001, and 180/π.

Then, the PID controller is implemented:

- Proportional: the error of the angle is simply the difference between the current and desired angles.
- Integral: in a static variable, `error_accumulation`, the approximate integral error is calculated by adding the error to it each cycle.
  `error_accumulation` is limited to be between [-2500, 2500], which is half the maximum duty cycle (5000 for the PWM).
- Derivative: we store the angle error from the previous cycle, and the difference between the current and previous errors is the approximate change in error, `error_deriv`.

- Dithering: We increment or decrement the desired angle by some angle increment (chosen to be 0.001) in the direction of the angle error.
- The final control value is calculated from each of the three terms:

```
Control = Kp*error + Ki*error_accumulation + Kd*error_deriv
```

If the new control value (duty cycle) is out of the bounds of the PWM duty cycle [-5000, 5000], we limit it to that maximum value. Each of the parameters is a defined value, which can be edited through user input, as described later.

Now that we have a new duty cycle, it needs to be sent to the PWM module. In initialization, one slice (slice 2, corresponding to GPIO pins 4 and 5) of the PWM is chosen, and the channel outputs A and B are connected to the H-bridge circuit as above. If a duty cycle is written to channel B, the motor spins clockwise; if written to A, it spins counterclockwise. The overall speed of the wheel is given as the difference between these two outputs, so it makes it much simpler to send a desired duty cycle, and thus speed, to one channel and set the other to 0. So, if the duty cycle is positive, it is written to PWM channel B, and A is set to 0, and vice versa for negative duty cycle.

The user interface is done using a thread on core 0. A user input state machine allows the user to change each of the PID parameters, as well as the desired tilt angle. Other options are to edit the speed of VGA plotting, and to directly edit the duty cycle for testing purposes.

A thread running on core 1 controls VGA display, which displays two plots, one of the complementary angle output (from -25 to 25), and one of the duty cycle (from -5000 to 5000), each scaled and shifted to fit on the screen well. As well, the PID parameters and desired angle, as well as numeric values of the above plots, are displayed. The plots are plotted in real time, at a user-defined speed.

D. Testing

Verification of the results in this lab was done visually, through trial and error, by looking at the stability of the pendulum after hitting it with a pen at different strengths (light, medium, and hard “thwacks”) and changing the different parameters until the pendulum was sturdy enough against the thwacks.

We had a few problems with the physical setup of the pendulum system, including the wheel falling off when it was spinning and the forward and backward movement of the pendulum where it was attached to the table. To fix these problems, we replaced the wheel and tightened the nut and bolt together before each trial, while still making sure that the pendulum was able to freely move.

Additionally, there was an error with the program when running on the microcontroller, where everything froze, including the VGA screen display and the serial interface. We think that this error was due to motor noise messing up the I2C communication transaction, and were unable to fix this other than by turning the power to the RP2040 off and on again to reset everything.

To balance the wheel, we found that our biggest issue was that the wheel did not react fast enough. To fix this, we increased the voltage, ending with a voltage of 6.89 Volts and changing the derivative value to 3000.

To verify that the algorithm was correct, we plotted two graphs, the duty cycle and the pendulum angle, to see whether or not the values from the duty cycle were inverse and proportional to the pendulum angle. This confirmed that our calculation of the duty cycle was correct, and that it was our parameters that needed to be adjusted to get the pendulum balanced.

III. Results

To select the PID gains, we found that the biggest issue was the reaction speed of the wheel, so we increased the Kp value to help improve this. In addition, we adjusted the other different parameters of the parts of the PID control algorithm, and ultimately found that the following values created a balanced pendulum:

- Kp = 3000
- Ki = 1
- Kd = 10000
- Angle = 0

Overall, the pendulum was able to withstand medium-hard levels of hits, and remained balanced. To view a video of our pendulum demonstration, click here.

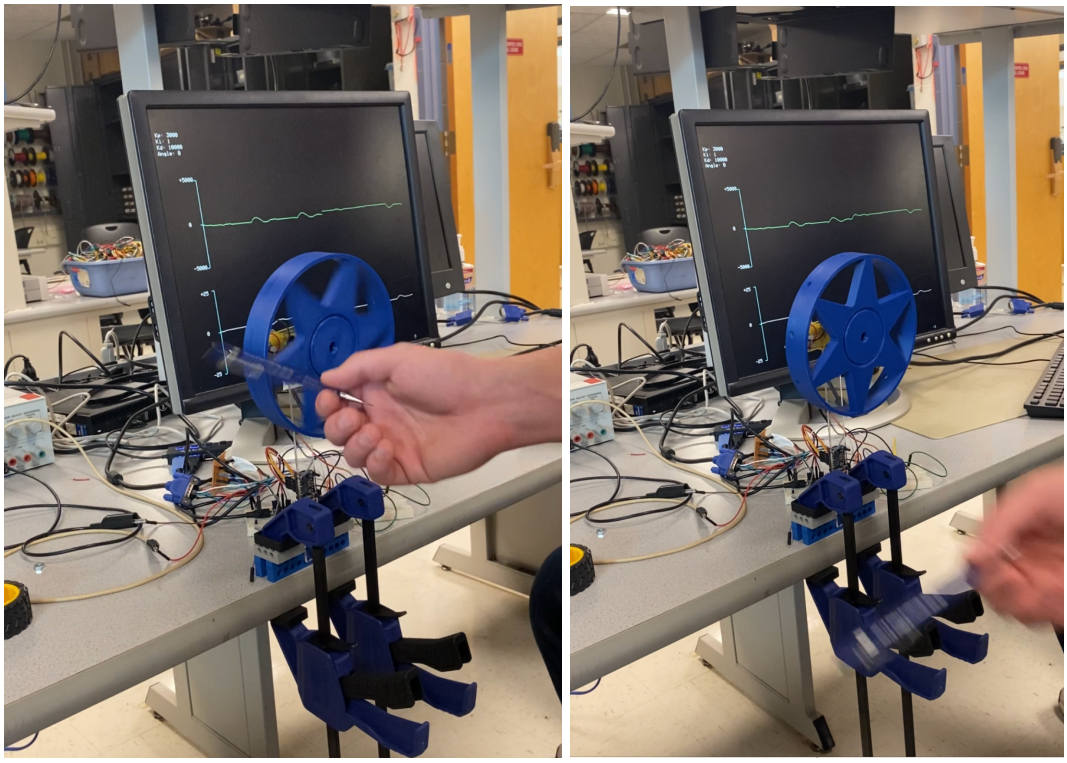

Figure 4. The pendulum remains balanced during and after the pen hits it.

IV. Conclusion

Overall, we were able to successfully implement the PID control algorithm and adjust the parameters to create a balanced pendulum. It was interesting to see the real world applications of the system in spacecraft, and allowed us to model it in the classroom setting. We used more features on the RP2040 microcontroller, including the I2C polling, and used what was implemented in previous labs such as the VGA display and serial interface to allow for user inputs. Since this lab was primarily hardware focused and dependent upon the behavior of our pendulum and wheel, there was a lot of trial and error involved in getting it to balance, which was an interesting requirement since it was important to know what each parameter did and what the effects of changing them would be. Finally, we found that the lab was reasonable, especially since our past issues with things like the VGA screen were solved in previous labs, through trial and error based labs can always get a bit frustrating when knowing that the algorithm is correct, and it requires small adjustments to values until the desired result is achieved. In conclusion, this lab was a great way to learn about how spacecraft movement works on a smaller scale, and gave us the opportunity to implement many of the things we have discussed throughout the class.