Flyback Chronograph

CSCE 462 - Fall 2023

Department of Computer Science and Engineering

Team Members:

Allen Li - allenli512@tamu.edu

Vivek Amarnani - vivek5a@tamu.edu

Created under the mentorship and guidance of Dr. Jyh Liu

Table of Contents

Executive Summary

For our Fall 2023 CSCE 462 project, we designed a clock-sized stopwatch; in essence, a three-handed flyback chronograph. Hidden behind the disguise of a simple three-handed clock is a complicated multifunctional stopwatch and clock.

Using an IR remote, the user can switch the clock's modes depending on the use case. By default, the clock will be in time-telling mode. The hours and minutes hands mounted on the center axis work together with a subdial representing seconds (seen at the six o'clock position) to represent the time accurate to the second. When the mode is switched, all three hands quickly move to their zero position (at the twelve o'clock) and await user input to begin timing. When activated via remote, chronograph timing (or stopwatch functionality) can begin. When timing is finished, the clock can return to time-keeping mode.

Introduction

Purpose and Background

A chronograph watch is an implementation of the stopwatch function. Usually, chronographs contain 1, 2, or 3 subdials to contain all the information. There is customarily one subdial that contains the ‘running seconds,’ then others which may contain hours or minutes elapsed. When a reset button is pressed, the information on all the subdials is reset back to ‘zero’ mode, which usually means all hands resting at the 12 o’clock position.

What differentiates a flyback chronograph from a normal chronograph is the reset speed. A flyback chronograph will ‘fly back’ to the resting position (within 1 second), whereas regular chronographs will move at a slower pace. The flyback functionality is generally harder to manufacture in wristwatches due to the amount of torque and energy needed to be generated and then quickly released.

Figure 1: Image of a labeled chronograph from Google

We wanted to create a larger clock version of a flyback chronograph, essentially a working wall clock with a ‘hidden’ stopwatch functionality. When switching back and forth between modes, it will have the ‘flyback’ effect, although due to limiting motor speeds and torque required it won’t be nearly as fast as its wristwatch counterparts.

Our project stems from a genuine interest and admiration for the watch industry and community. This project has many potential applications such as a timer for sporting events or cooking, as well as an aesthetically pleasing home decor alternative to traditional clocks or wall clocks.

Constraints

Due to the highly mechanical nature of our project, there are some constraints that we will need to abide by. When it comes to watch/clock design, precision is key and must be reflected in the parts and tools that we use. The first constraint involves the speed and accuracy

to which our motors can be controlled. We will dive more into this issue in later portions regarding the design process, but we strive to have our motors operate fast and accurately. The second constraint involves the timing aspect. With our limited capabilities, keeping accurate time was a huge concern when starting off. Whether this was through accurate hardware, software libraries, or a combination of both, our tools played an impactful role in how specific we could keep time.

Design Process

Initial Ideas

Initially, we wanted our clock design to reflect how modern clocks look. We juggled many different ideas on how we could design our clock face to most effectively display the time, as well as implement a working stopwatch functionality.

We considered three faces (with three motors), for hours, minutes, and seconds. We also considered using 3D printed hands which would ‘catch’ the other hands when moving backward so that we would both reset the hands when prompted, as well as display any time on the 12-hour clock. Furthermore was the issue of how the hands would be controlled. Would the motor control the hands directly? If so, how many motors do we need? We knew we needed a motor with high enough torque as well as precision programming. If we wanted one motor to control multiple hands, how would we implement a gearbox or something equivalent so that the ratio of movements versus time stayed the same?

We settled on a design that was comprised of the many different execution methods of our project. We would use two motors: one to control a premade gearbox representing minutes and hours, the other to turn another premade gearbox meant to display seconds. Both gearboxes would be fixed within the first piece of wood. The front side would be the clock face, the backside would be connected to an equally-sized piece of wood that would hold up the shafts of the two motors, and the bodies of the two motors would be mounted to the back of the second piece. Then there would be a three-inch tall wall behind the second piece which would serve as the housing for all the electronics. We would use a Raspberry Pi microcontroller, 2 stepper motors, the gearboxes from a storebought clock, an RTC (real-time clock module), an infrared receiver/remote, as well as raw materials for binding (wood, fasteners, etc.).

Challenges Faced

Our implementation above would face various challenges. After figuring out the code for the stepper motors, we realized that they would be far too slow, and not nearly powerful enough to support the fast ‘flyback’ utility. Hence, we switched to NEMA 17 motors we found in the lab room. This would cause more problems than it would fix as a majority of the time spent working on the project was from debugging and learning how to use a foreign motor with little documentation.

We spent many days and hours trying to control the NEMA 17 motor with our Raspberry Pi. With the connected drivers (L298), we tried many packages, wiring methods, relays, etc. to even begin running the motor, but to no avail. While it was possible, almost all documentation and support was for our microprocessor’s cousin: the Arduino. At this point, our other hardware components such as the RTC and infrared receiver/remote were all functioning fine on their own.

We were forced to switch to the Arduino microprocessor, which we were all completely unfamiliar with after half a semester of working with the Raspberry Pi. Luckily, we were able to get the motor running, which meant now we had to learn how to control it with the many different libraries. This was our next challenge: find an online library that would allow us to both accurately control the positioning of the motor, as well as deliver enough power to quickly move between positions.

It was at this time we realized that the gearbox that would control the second's hands was unneeded, as it only resulted in more programming to calibrate the motor speeds with the rotations needed to accurately display time. We then needed to find out how to suspend two heavy-duty NEMA 17 motors in midair, in such a way that there is no movement when the shaft is spinning as well as staying in a precise position to not shift along the axis.

On top of hardware issues, we also encountered our fair share of software issues. We ran into many issues in regard to reading the RTC and controlling different IR signals. We wanted our design to correctly display the current time even if power to the Arduino is disconnected. Our RTC has a built-in battery so this was possible. Secondly, choosing and learning a completely new and under-documented library. Initially, we started with the “Stepper.h” library, which could operate the NEMA17, but proved to be far too slow. We then switched to the “AccelStepper.h” library. With initial tests we found that it enabled our motor to spin at much higher speeds, however, the documentation was extremely light and hard to understand. For example, the library would allow us to input the number of steps we want our motor to run at 1.8 degrees per step, however, if we want it to take 60 seconds to perform 200 steps (one full rotation), this would require steps that were not integers, which the library did not allow. To solve this, we would move in increments of 3,3, then 4 repeatedly so the average would be 3.33 steps per second. A similar process was used to control the movement of the hour and minute hands via the gearbox, though this algorithm was extremely complicated. Through many rounds of trial and error, we were able to get our motor functioning to our liking. One of the most difficult portions of the project was controlling the location of the motor and writing proper code so that the motor “knew” where it was. We had to do this twice, once for the second hand, and another time with different calibrations for the minute/hour hands.

Our largest and most time-consuming challenge was the connection between the motor and the dial that adjusted the gearbox. Due to the sheer power of the motor, the connection area would often vibrate, loosening the connection over time. We tried many rounds of glue, bindings, and heavy preventative measures, but there was nothing that could prevent the loosening of the glue over time. The glue that was allowed to set for a few hours would provide smooth movements, but as the connection deteriorated, the hand movement would begin to slip and miss steps. This would completely undermine the amount of math we put into the program as all movements are random and subjective to how much the gears slip.

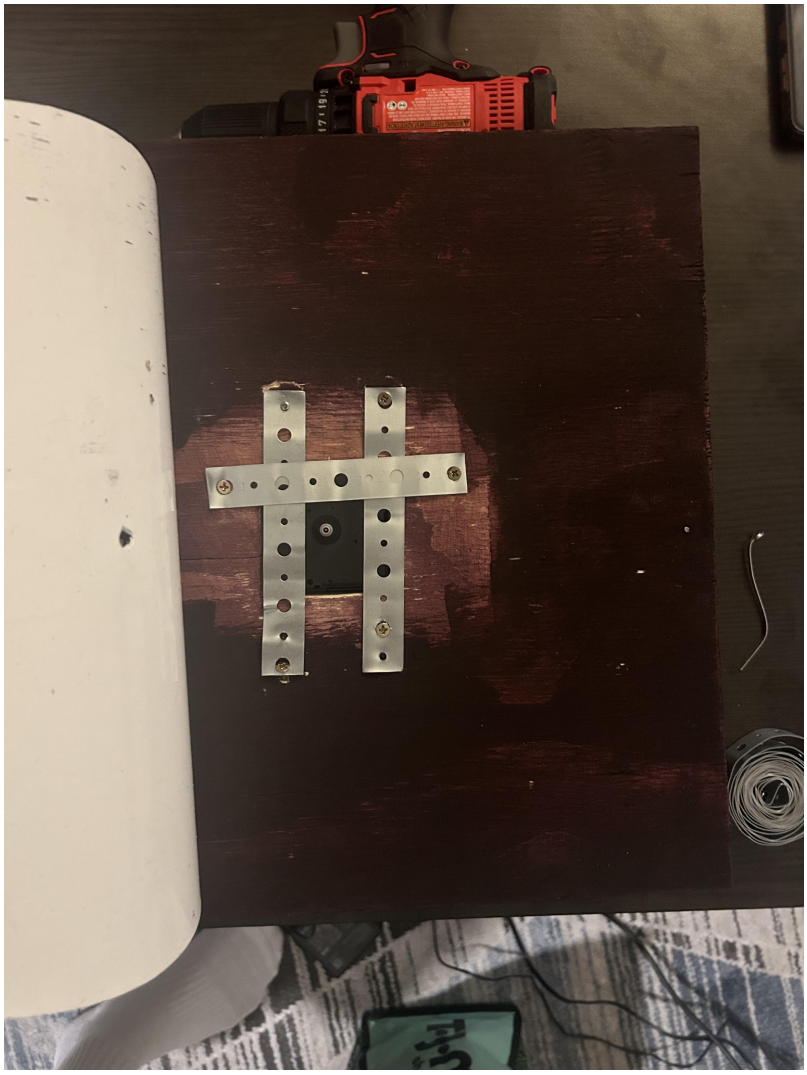

Figure 2: Measures to prevent gearbox movement due to motor vibrations

Design Adjustments

Initially, our flyback chronograph was to be powered by a Raspberry Pi, we ended up switching this for an Arduino Uno due to the larger amounts of documentation as well as the compatibility with our NEMA 17 motor. The switch to an Arduino meant we could use our stronger and faster NEMA 17. With that, we also incorporated the NEMA 17 driver: the L298. Our physical casing originally had two gearboxes that would be controlled by motors, however with the accuracy of the “AccelStepper.h” library we were able to get rid of the seconds-motor. This meant there now had to be a cutout in the middle wood section to allow for the second motor to reach all the way to the front wood while the two pieces of wood lay flat against each other. Our RTC did not need to be changed as it was fully compatible. Due to time constraints, we were unable to implement the IR remote functionality as we did not have time to debug the issues with the receiver responding to the signal from a remote source.

System Design

Physical

The physical product consists of three portions: the front wood', the middle wood', and then the clock box. The box's dimensions are 14''x14''x4.5''. To suspend the NEMA17 motors in their correct positions, we use two angle brackets that would support the sides of the motor.

The front wood holds the gearbox in the middle of the box (which turns the minute and hour hands), and then halfway between the middle of the face and the bottom, the second-hand motor is suspended (on the back).

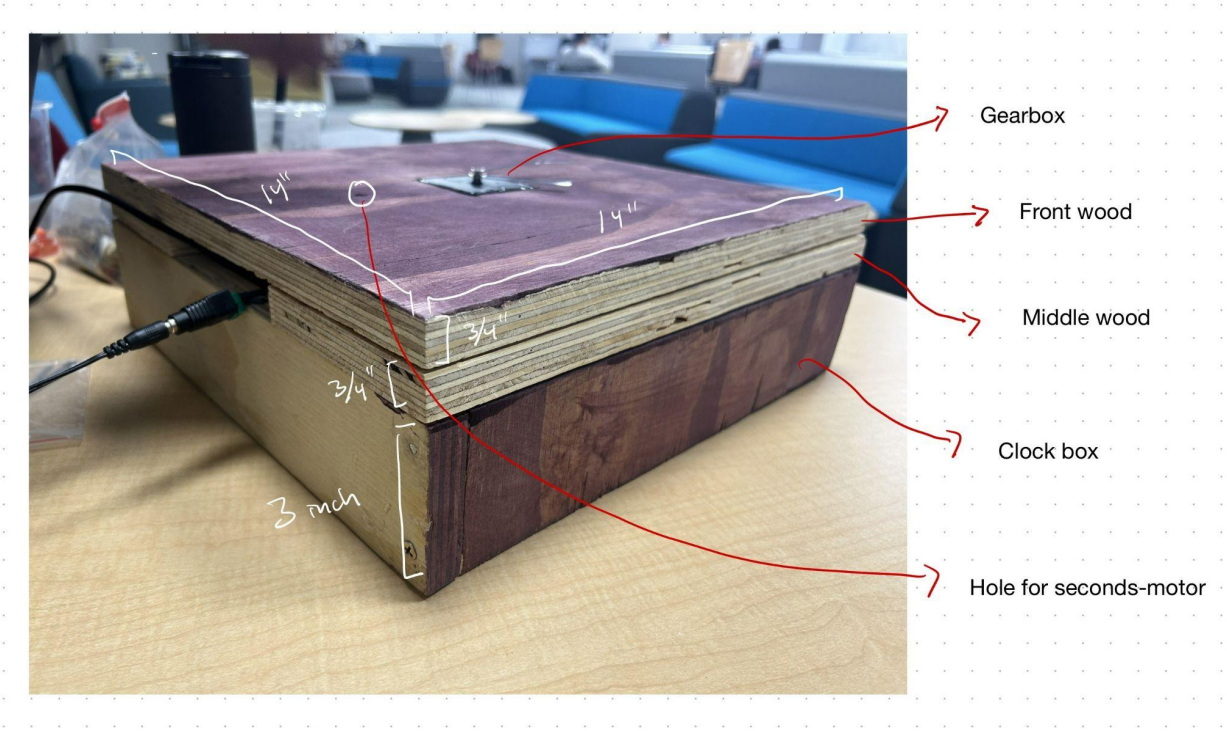

Figure 3: Labeled side view of Flyback Chronograph

The middle wood simply serves as a barrier between the body of the hour/minute motor and the gearbox that it is supposed to be connected to. On the backside of the middle wood are two more angle brackets to hold the hour/minute motor up. The front and the middle wood sit perfectly flat against each other, with the front wood in front. On the back of the gearbox (nested in the front wood), there is a small dial meant to adjust the time on the original clock it came with. This dial should align perfectly with the shaft of the hour/minute motor. There is a square cutout beneath the hour/minute motor in the middle-wood to allow for the second motor to sit flat against the front-wood.

We stacked washers on the protruding portion of the hour/minute motor shaft (to increase surface area), then superglued the end washer to the time-adjustment dial of the gearbox. This is how we control the hour/minute hand. This system was not entirely durable and would often become unstable in our testing - to counteract this, we used malleable stainless steel binding to fasten the gearbox to the case and ensure no slipping between the motor and gearbox.

Behind the middle wall sits the encasing of our hardware components. We built essentially a 3-inch wall around the perimeter of the box, creating a hollow box behind the motors to hold our microprocessor, wires, motor drivers, etc.

On the frontmost surface of the clock is a 12”x12” printed dial designed by our team. Hands were salvaged from other clocks, but modification was required to fit these hands to the shafts. The entire wooden construction was stained in a dark brown treatment to be aesthetically pleasing and complement the appearance of the clock’s dial and hands.

Hardware

Our hardware components consisted of two NEMA 17 motors, both with L298 drivers, a DS3231 Real Time Clock module, an infrared receiver + remote sender, Arduino Uno, and 3 12V power supplies. Both of the two NEMA motors are controlled by a L298 driver which is then hooked up to the Arduino Uno. The RTC and infrared receiver are also both wired to the Arduino. Due to the lack of GND ports, we soldered two wires together in a Y-shape to put the last remaining GND in parallel. We needed two female barrel connectors for the 12V power source to connect to the L298.

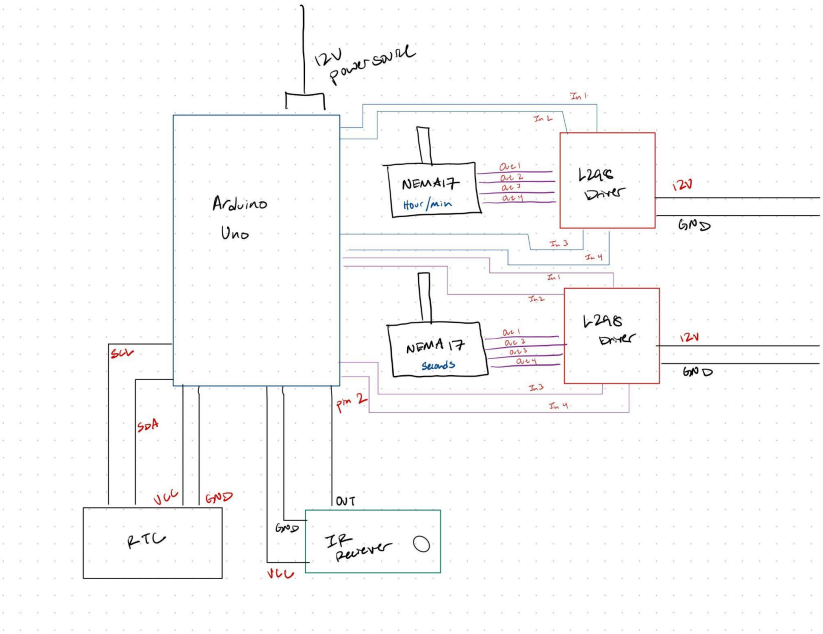

Figure 4: Rough wiring diagram of the Flyback Chronograph

Between the NEMA 17 and the L298 driver, there are 4 connections: 2 connected to the 5V enable, and 2 for output B. The 12V and GND of the driver connect to the power source (after using the barrel adaptor). We then connected A enable and B enable to themselves, then the four logic input pins were connected to the Arduino.

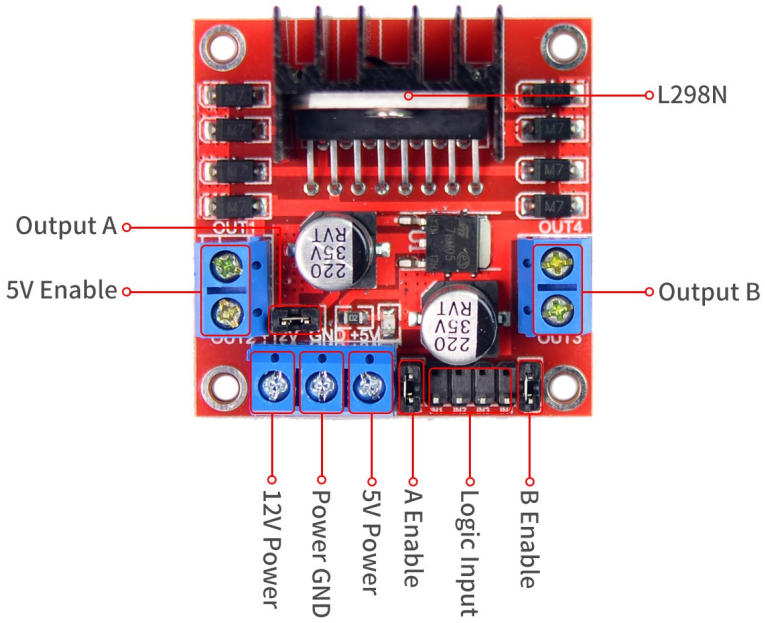

Figure 5: Pin Diagram of L298 Motor Driver

Both the RTC and infrared receiver connect directly to the Arduino Uno. The RTC requires the SDA and SCL pins (on top of GND and VCC), and the infrared receiver only requires an output pin (along with GND and VCC).

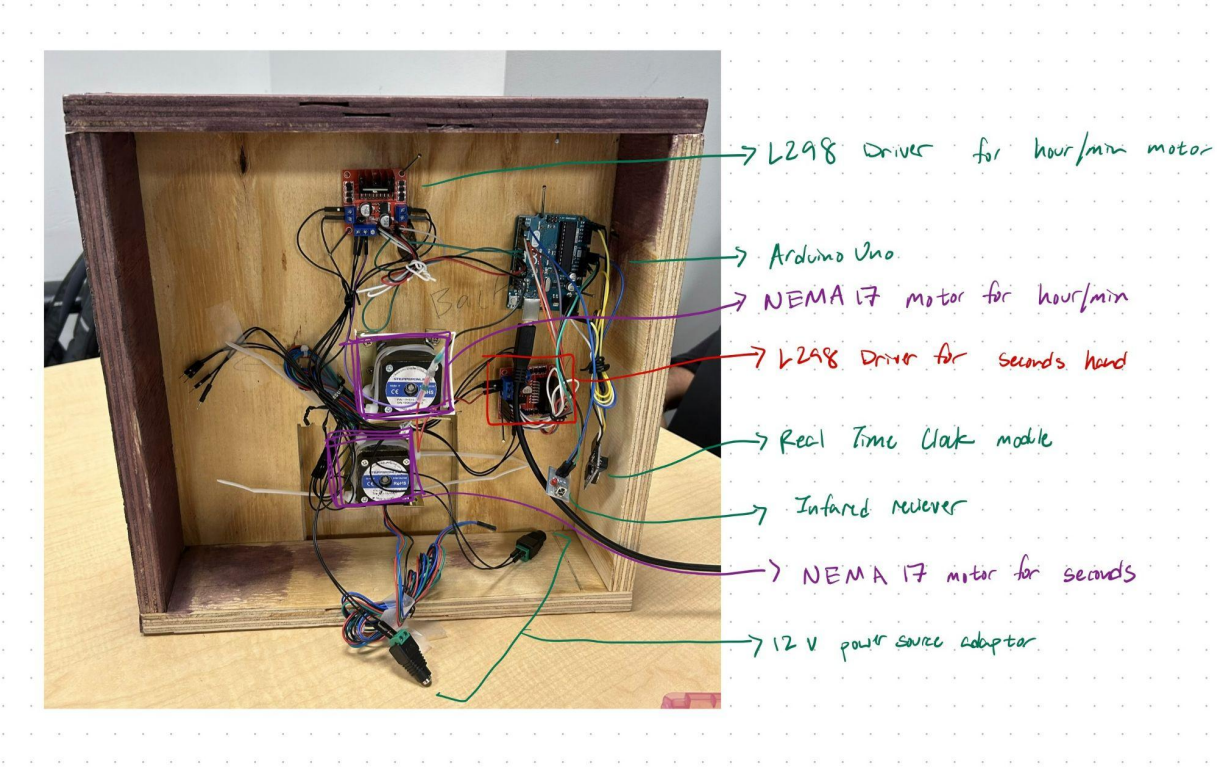

Figure 6: Labeled backside view of Flyback Chronograph

Software

As is customary with Arduino sketches, there are two major functions setup() and loop() to control the bulk of the program. Several help functions were created to be called within these functions. The setup() function enables the use of the RTC and IR remotes, sets the positions around the clock, and more. This function also handles switch cases to allow for the mode switching, and many conditional statements to prevent the user from providing unwanted inputs. The loop() function controls the movement of the hands, updating several times a second for the time-keeping mode via the RTC, as well as enabling the chronograph mode.

We have one function that takes an RTC input and will move the hour/minute hands to their correct position. The position is based on a matrix of the number of turns needed per hour (averaging 266.66) in addition to the number of turns needed for each minute. These were determined through specific algorithms designed based on the dial turn-to-gear spin ratio. We also have a position matrix for the second hand, which evens out to 3.33 turns per second. Because our library did not allow for non-integer inputs for distances we wanted to move, we had to create algorithms that would yield the correct averages at the end of every hour and minute.

Whenever the second hand would pass by zero (go from 59 to 0), we would set the current position to 0 so that the flyback function had a reference position to go to. We tried to do something similar with the hour/minute hands, but due to the large number of steps needed, it

was much easier to calculate the number of steps “left” to get back to 12:00. 3200 steps would take us 12 hours, the rest can be calculated ourselves.

Outcome and Improvement

Ultimately, we were able to create a clock with the three functional modes that we had initially designed, including the time-keeping mode, the resting chronograph mode, and the active chronograph mode. Our final implementation was attained after much hardship with the motors, gearbox and gear ratios, physical construction issues, and a wide range of software-related problems. The final design for our clock proved to be aesthetically pleasing and highly legible, and therefore a true functional tool for a wide array of uses. We set out to create a three-handed ‘flyback’ chronograph, and we succeeded in doing so, but were unable to implement the ‘extra’ features that we aimed to include.

There are several areas of improvement in our implementation. To prevent unusual behavior, a limit sensor at the twelve o’clock position could have been implemented as a foolproof way to distinguish the zero position in the software. Additionally, using EEPROM would allow the clock to save information when the clock is detached from the power source and read this information when the power source is reconnected, creating a reliable way for the clock to always keep proper time without any user adjustment. We had aimed to implement IR functionality as well, but were unable to do so by the deadline because of the finicky nature of the receiver. Implementing the IR remote as well as a sort of WiFi-connect remote functionality would be a surefire way to elevate our prototype into a true ‘smart-home’ product. Lastly, a greater fit-and-finish for the physical construction could be highly beneficial. Adjusting the dimensions, especially the thickness, would lead to a more polished product. Adding a glass cover and a plastic/metal bezel to the front dial surface would serve to elevate the entire appearance by adding dimensionality and lead to a more finished look.

Appendix

Bill of Materials

Item	Price	Quantity	Description	Link
Stepper Motor	$11.98	1	set of 5 stepper motor with driver board	https://www.amazon.com/GeeekPi-Stepper-28BYJ-48-Uln2003-Compatible/dp/B087B5NWY4/ref=sr_1_3?keywords=uln2003+driver+board&qid=1697578380&sr=8-3
DS3231 RTC for pi	$9.99	1	Another RTC for Pi, this one higher precision.	https://www.amazon.com/AITRIP-Precision-AT24C32-Arduino-Raspberry/dp/B09KPC8JZQ/ref=sr_1_3?crid=3VKW3LSMC10MK&keywords=DS3231%2BRTC&qid=1697578784&sprefix=ds3231%2Brtc%2Caps%2C124&sr=8-3&th=1
RTC battery	$5.99	1	Battery for RTC	https://www.amazon.com/SKOANBE-Lithium-Button-CR1220-Battery/dp/B08CMJ8J6K/ref=sr_1_5?crid=BLR7JZTQJY45&keywords=CR1220%2Bcoin%2Bcell%2Bbattery&qid=1697579069&sprefix=cr1220%2Bcoin%2Bcell%2Bbattery%2Caps%2C223&sr=8-5&th=1
Remote Control + Receiver for Pi Infared	$8.99	1	2 pack of a remote control and pi reciever for infared signals	https://www.amazon.com/DIYables-Infrared-Controller-Receiver-Raspberry/dp/B0BXKJ7QHY/ref=sr_1_5?crid=2GVWGB1TP7BTR&keywords=basic%2Binfrared%2Bremote%2Bcontrol%2Braspberry%2Bpi&qid=1697579264&sprefix=basic%2Binfared%2Bremote%2Bcontrol%2Braspberry%2Bp%2Caps%2C143&sr=8-5&th=1
Wood	$51.00	4	4 pieces of 2x2x(23/32) wood. Each piece is 12.75. Two were used for prototype, two were used for final	Home Depot or Lowes
CR2025 button batteries	$5.00		battieres for remote control	https://www.amazon.com/Duracell-Lithium-Battery-Lasting-Count/dp/B07G7L5M4J/ref=sr_1_5?crid=2XV3KTJANWZP6&keywords=CR2025+button+batteries&qid=1697832356&sprefix=cr2025+button+batterie%2Caps%2C129&sr=8-5
Angle	$4.81		To hang the NEMA 17	Home Depot

Bracket			motors	
Super Glue	$3.98		Connecting hands to motor, gearbox to motor	Home Depot
Flat washer	$2.50		Increasing surface area on motor shaft	Home Depot
Clocks	$43	6	Bought 5 small clocks to test gearbox and hand movement. Bought one larger clock as final prototype tool	Bought at target

Tentative Work Schedule

- Week 1
  - Finalize parts needed and a rough sketch
  - Should have a budget finalized for approval.
- Week 2
  - Begin testing parts incrementally. Make sure all parts work as expected.
  - Test infrared signal pick up. Test changing motor speeds along with store-bought gearbox.
  - Determine voltages for different motor speeds and how that will affect the watch hands spinning.
- Week 3
  - Begin coding individual parts. Picking up different button signals. Make sure the clock can display whatever time we want. The hands should move correctly in 'time-telling mode'.
- Week 4
  - Begin putting different portions together. Flyback functionality should be working. Make sure chronograph mode timing is correct.
- Week 5
  - Presentation should be complete.

References

- https://www.youtube.com/watch?v=SUziz1zupGk
- Explains stepper motor, gear ratios
- How to control a DC motor with an encoder
- https://www.youtube.com/watch?v=dTGITLnYAY0

- How i Made this Amazing Hollow Clock | Arduino
- https://www.youtube.com/watch?v=jvoOgxK4EvI

- Analog Clocks using Servo Motor
- https://www.youtube.com/watch?v=TL7jMiQ42s8

- clock using a servo motor
- https://www.youtube.com/watch?v=ktay08mUI6g
- Using 28BYJ-48 Stepper Motor Push button Speed with 8 projects: Ultimate Video Tutorial Lesson 107
- https://www.youtube.com/watch?v=TQ7R2bY-MWU
- Servo motor clock with two buttons
- https://www.youtube.com/shorts/y9a1SYxLnEE
- A Unique clock model powered by Arduino Servo motors
- https://www.youtube.com/watch?v=6JrwaI-NqDM
- Servo motor counter clock and clock wise
- https://www.youtube.com/shorts/Zcpna4ndyFk
- 3D printed WiFi sync analog clock
- https://www.youtube.com/watch?v=rGEI4u4JSQg
- Stepper and Servo Motor Clock Demo (Mod 3, Task #1)
- https://www.youtube.com/watch?v=isYkBRXG-YI
- How To Use An RTC (Real Time Clock) With Arduino Uno R3 | Make an Accurate Clock!
- https://www.youtube.com/watch?v=gv1h-3kK6SU
- Making a clock spin faster
- https://electronics.stackexchange.com/questions/70196/making-a-clock-spin-faster
- How fast can a Quartz Clock spin?
- https://www.youtube.com/watch?v=XzXfadQXRn8
- RTC with raspberry pi
- https://pimylifeup.com/raspberry-pi-rtc/
- https://www.seeedstudio.com/blog/2020/07/07/raspberry-pi-rtc-tutorial-using-ds1307-and-ds3231-rtcs-with-raspberry-pi-m/
- https://www.circuitbasics.com/what-are-real-time-clocks/
- IR with raspberry pi
- https://www.digikey.com/en/maker/tutorials/2021/how-to-send-and-receive-ir-signals-with-a-raspberry-pi
- https://circuitdigest.com/microcontroller-projects/raspberry-pi-ir-sensor-tutorial
- Limit switch w/ pi
- https://newbiely.com/tutorials/raspberry-pi/raspberry-pi-limit-switch
- Lots of tutorials
- https://newbiely.com/tutorials/raspberry-pi/raspberry-pi-servo-motor