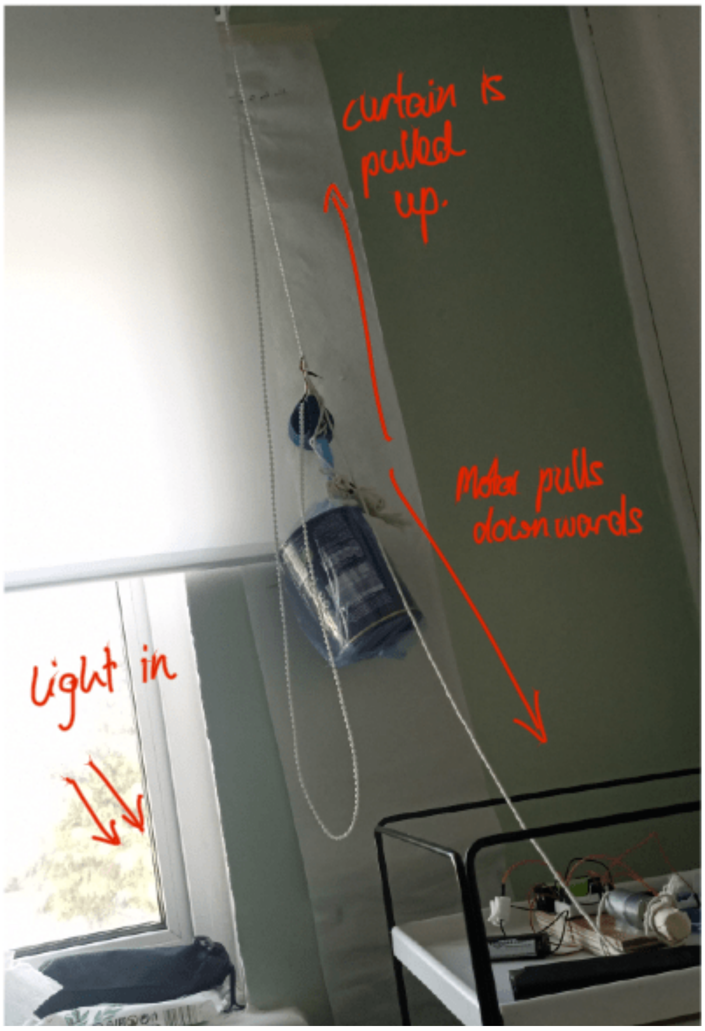
Motor Alarm Design

By Clement Jin

General Design



Essential Requirements

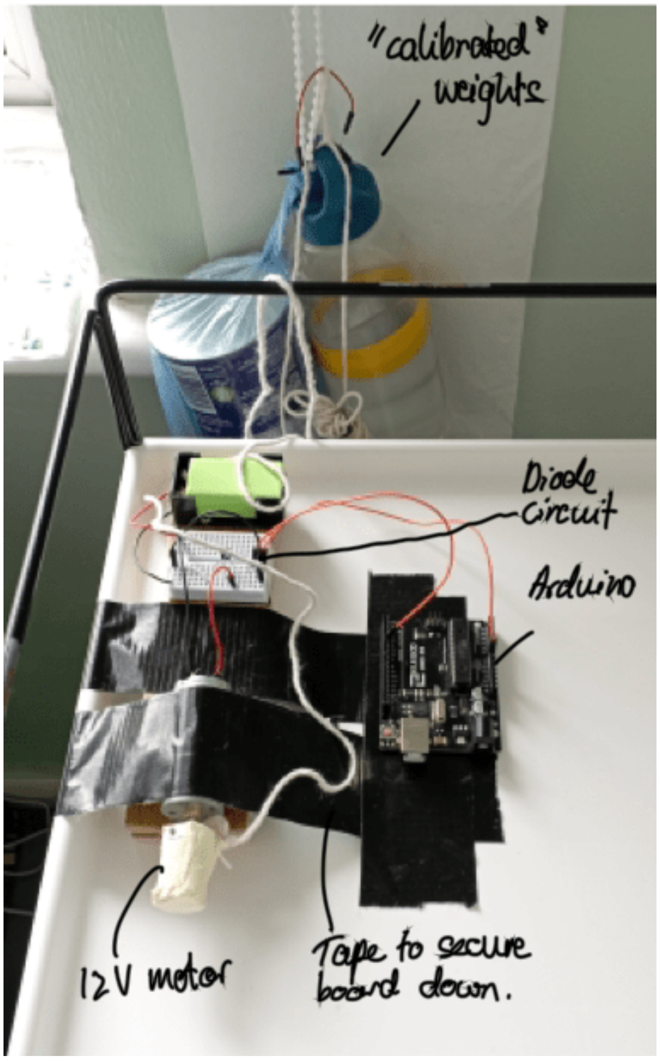
1. The device should be able to lift the curtains autonomously at the specified alarm time.
2. The device should be able to display current time and alarm time.
3. The device should allow users to configure current and alarm time independently (i.e., without a computer).

Non-essential Requirements

1. The device could include a sound alarm.
2. The device could be sensitive to light intensity instead of a specific time.

Motor Design (right)

Before adding gears, the force from the motor proved to be inadequate to lift the curtain. Thus, as a temporary solution, I attached a water bottle to the curtain so a smaller force was required from the motor. I used a water bottle so I could 'calibrate' the weight of the bottle by drinking the water!) This worked, but was clunky and difficult to use.



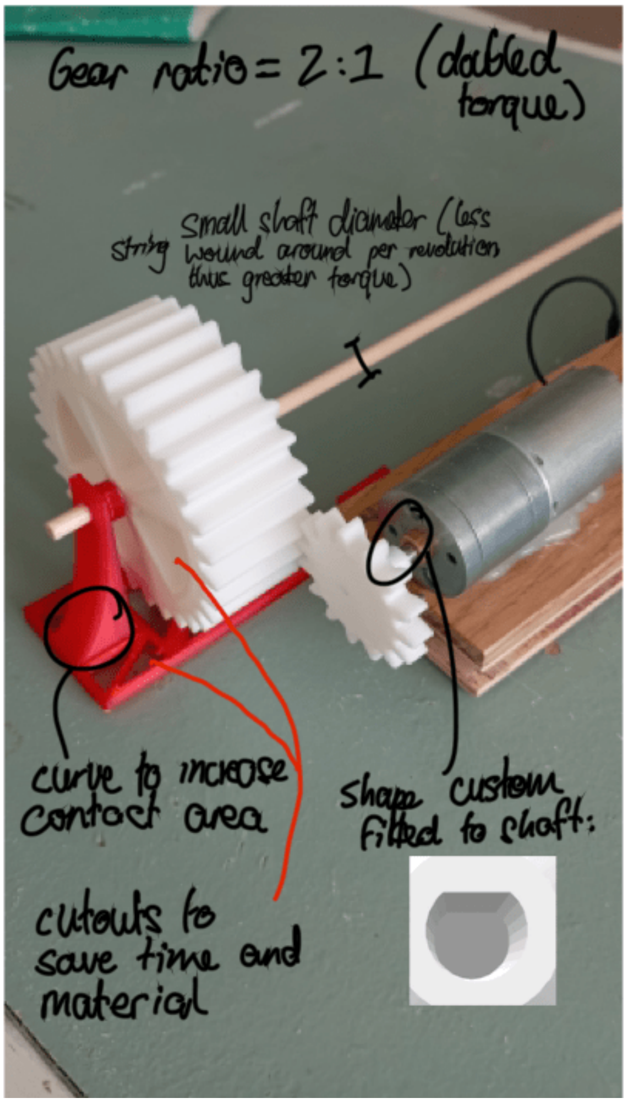
Gear Design

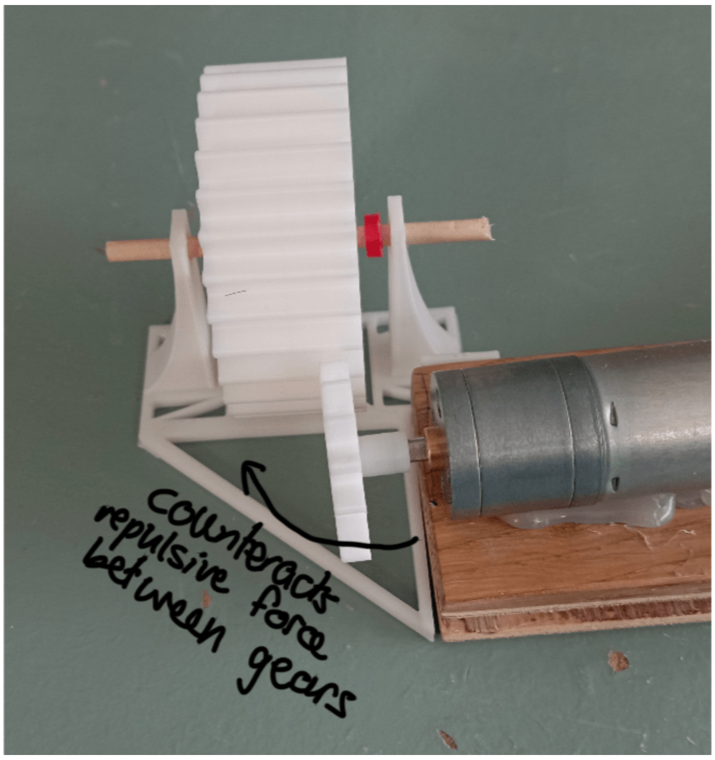
To create a large gear reduction that was compatible with my design, I required the precise manufacture of the gears and the gear holder. Thus, I chose to 3D print these parts to ensure this precision, while also allowing me to make prototypes quickly and efficiently.

This was favourable over laser cutting, as I could easily make thicker gears, whilst creating complex curves to increase structural strength (especially on the stand).

Prototypes 1 & 2

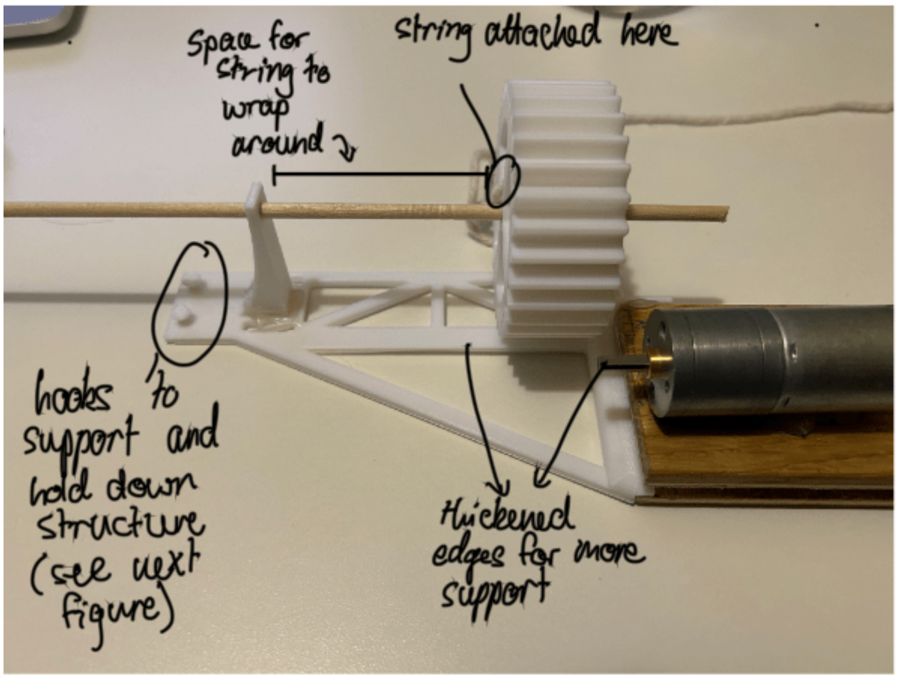
In my first and second prototypes, the gears ran smoothly, but I severely underestimated the force exerted by the gears on the holder. These forces pushed to tear the stand from the motor board both horizontally and vertically, ultimately pulling the 2 components apart. However, the gear reduction did prove sufficient to pull the curtain down.


Gear ratio = 2:1 (doubled torque)
Small shaft diameter (less string wound around per revolution thus greater torque)
curve to increase contact area
shape custom fitted to shaft:
cutouts to save time and material



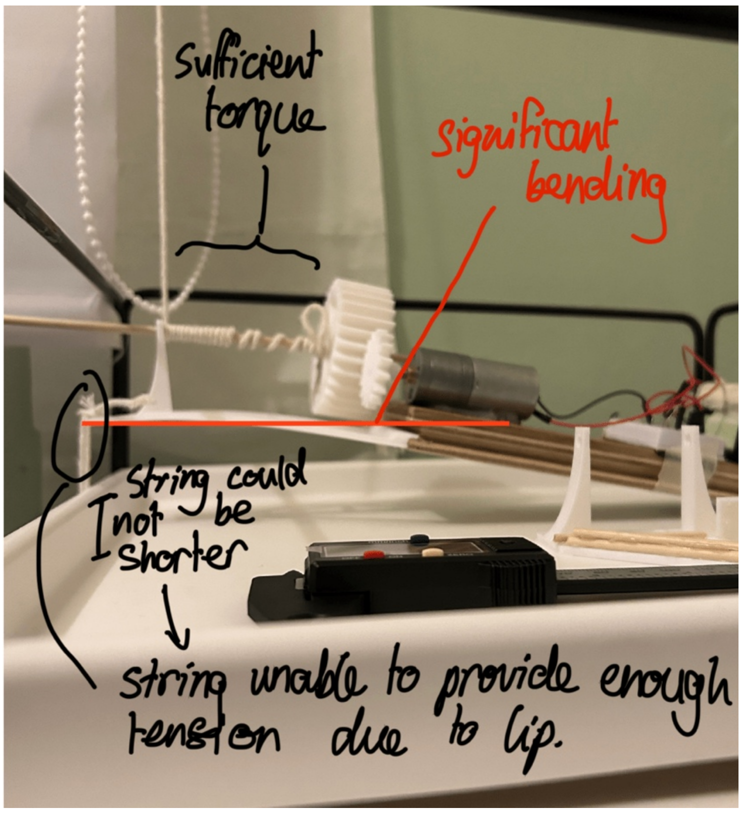
There were minor inaccuracies regarding dimensions, but these were extremely easy to remedy in Fusion 360 as a result of the timeline feature and setting well designed constraints.

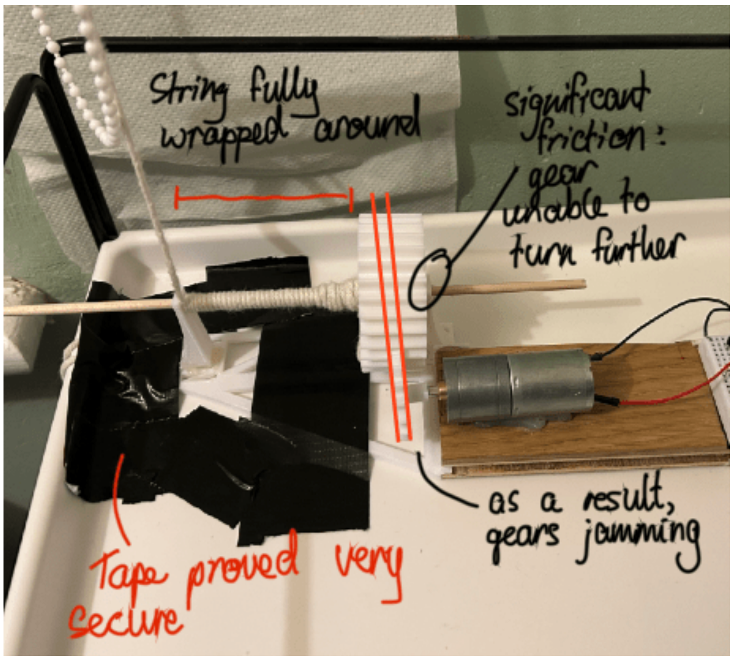
Prototype 3


Space for string to wrap around
string attached here
hooks to support and hold down structure (see next figure)
thickened edges for more support

To provide more stability, I created a platform from which I could tie the gear stand securely, while making more space to wrap the string around. I also increased the length of the stand to accommodate more string.

However, the shape of the platform made it impossible for the string to provide enough tension to hold the build down, while the base was too flexible and bent significantly during loading.


Sufficient torque
significant bending
String could not be shorter
string unable to provide enough tension due to lip.

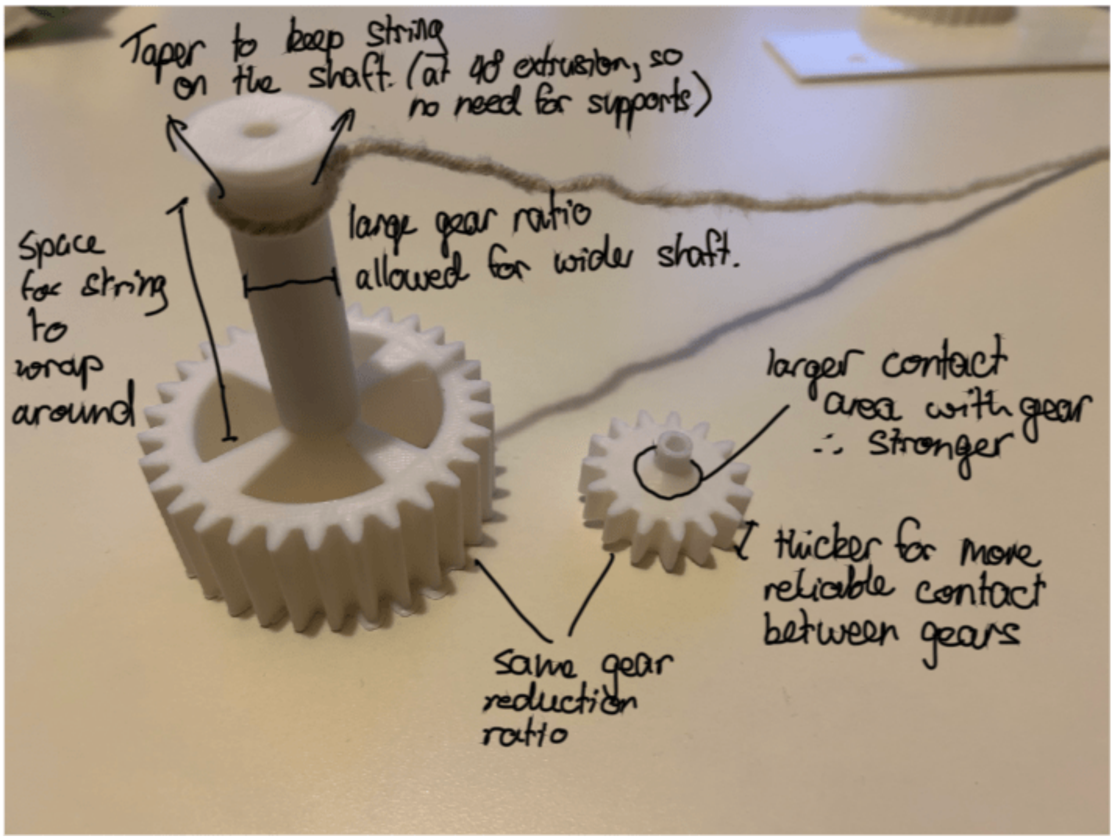

String fully wrapped around
significant friction: gear unable to turn further
as a result, gears jamming
Tape proved very secure

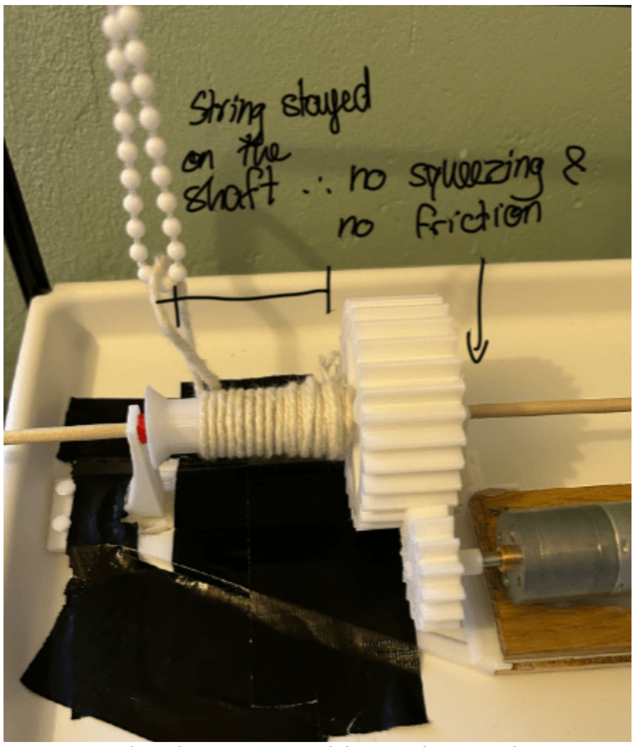
I found the best solution to be simply securing the base with gaffer tape, which was surprisingly stable but not very elegant.

Another issue with my design was that the string shaft was too thin, causing the string to take up the whole shaft, ultimately pushing the gear against the sides of the stands.

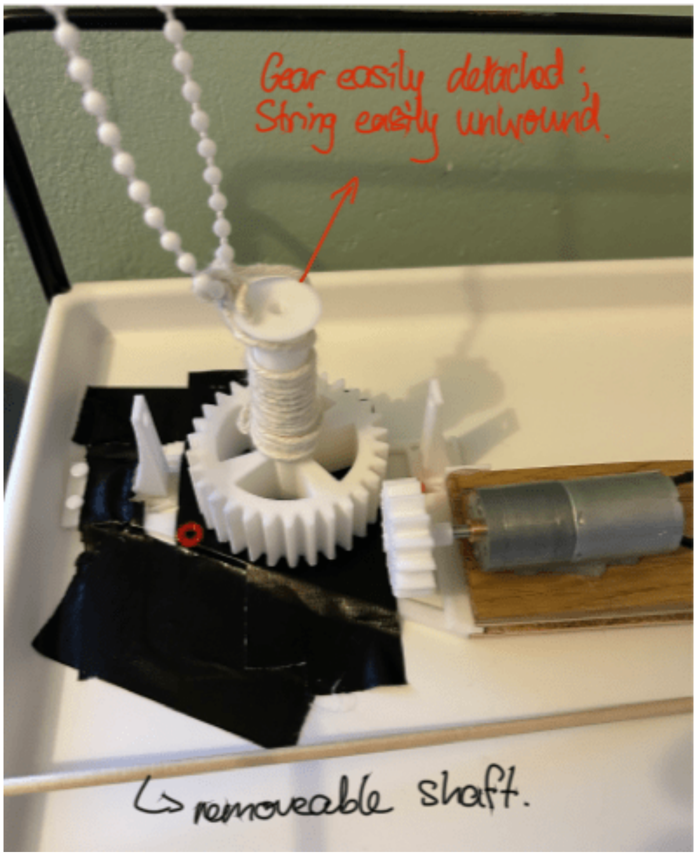
Prototype 4

I solved the problem of the string jamming the gear by designing new gears that kept the string on the large gear itself, and away from the sides of the stand. My use of 3D printing was perfect for this design, as it would have been extremely hard to make in any other way. Moreover, I increased the thickness of the shaft, so more string could be wound around it, while the gear reduction would still provide enough force to lift the curtain.


Taper to keep string on the shaft. (at 48 extrusion, so no need for supports)
Space for string to wrap around
large gear ratio allowed for wider shaft.
larger contact area with gear ∴ stronger
thicker for more reliable contact between gears
same gear reduction ratio



Conveniently, the gear could now be easily removed, and the string easily unwound.



Display Design

Display Requirements

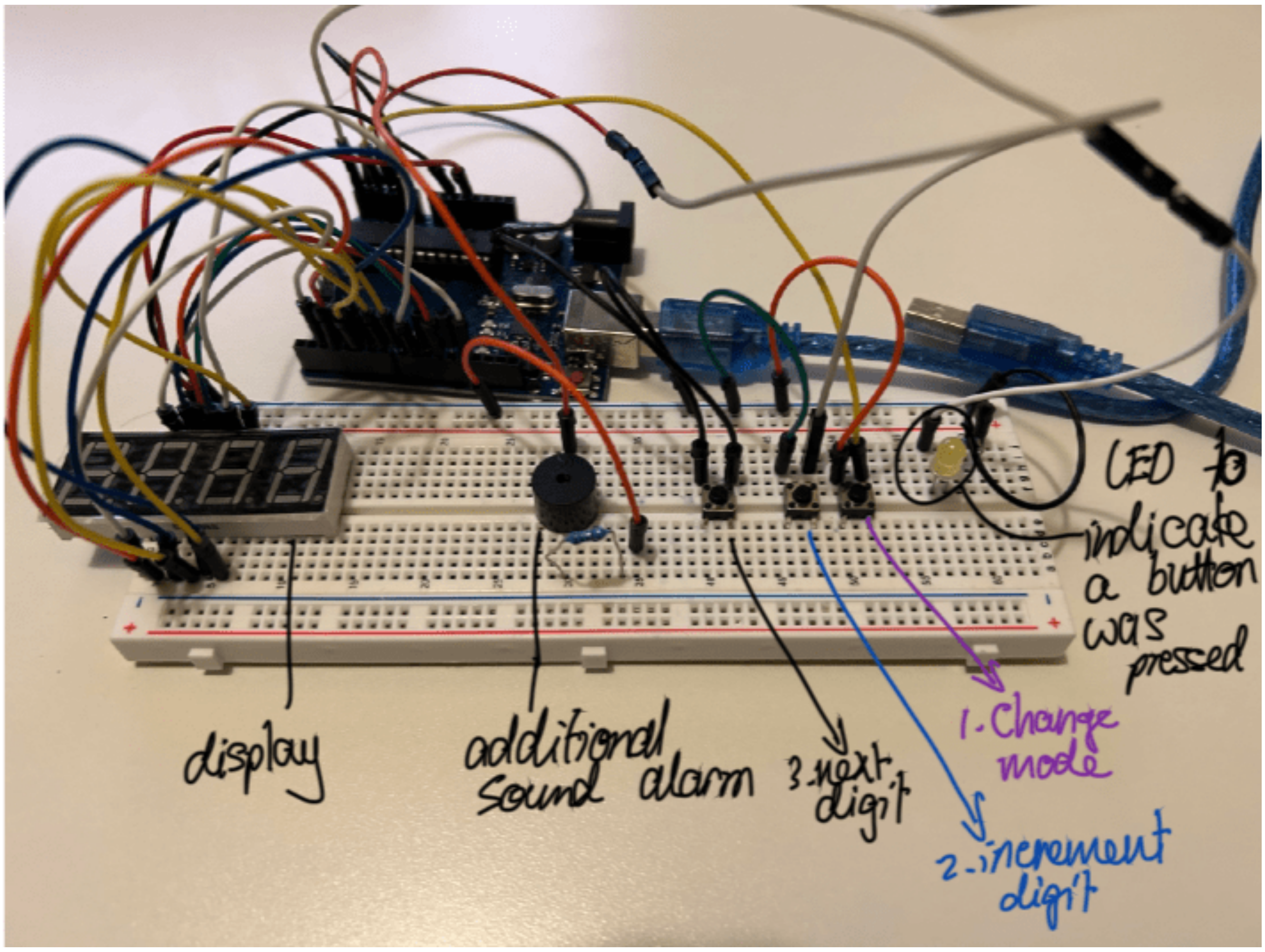
In line with my essential requirements, I developed some extra requirements for my display:

1. The display should be able to display current time accurate to ± 1 minute.
2. The display should be able to turn off while still keeping track of time.

A 4-digit-7-segment display and push buttons were the obvious choice, as they were relatively simple to set up (compared to an LCD) and could display all 24 hours of the clock. One drawback, however, was that the display used up all of the digital pins on the Arduino, so I was reduced to the 6 analogue pins (A0-A5) for my remaining circuitry. To reduce pin usage, I designed a User Interface that satisfied my requirements with just 3 buttons.

User Interface Design

1. Change mode. Available modes were {display current time, display off, modify current time, modify alarm time}
2. Increase digit. When modifying current or alarm time, this increased the value of the target digit by 1.
3. Next digit. When modifying current or alarm time, moved to editing the next digit.



Software Design

Splitting my program into multiple files helped significantly with readability and debugging.

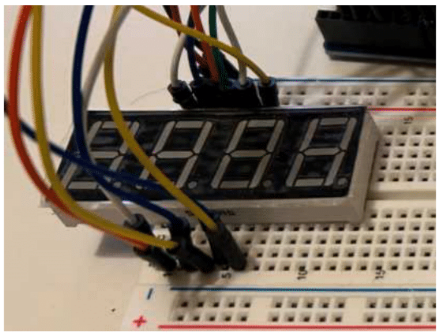
Black is the main loop of the main file.

Red functions are part of the UI file

Blue functions are part of the display file

Purple function (update-clock) is its own clock file, due to the handling of digits rolling over, eg. 10:59→11:00.

Green function (drive_motor) is its own alarm file.



Main Program

```
display_mode = 0  // can be: 0 : display on
                             1 : display off
                             2 : edit current time
                             3 : edit alarm time

void loop { // main arduino loop
- - - - - - - - - - - - - - - - - - - - - - - -
    // switching display modes

     if display_mode = 0:
          run_display_on_function()
     elif display_mode = 1:
          run_display_off_function()

     elif display_mode = 2:
          run_change current_time_function()

     elif display_mode = 3:
          run_change_alarm_time_function()

     else:
        raise error.
- - - - - - - - - - - - - - - - - - - - - - - -
    // edit current or alarm times

     if (next_digit button was just pressed):
          move_to_editing_the_next_digit_along()

     elif (add_one button was just pressed):
          increase_current_digit_by_one()

     elif (next_digit button was just pressed):
          display_mode += 1  // changes mode
- - - - - - - - - - - - - - - - - - - - - - - -
    // run alarm
     if  current_time = alarm_time:
          drive_motor()
```

Other key values and functions

current_time : an array of the current time

alarm_time : the alarm goes off when alarm_time = current time

update_clock() : increments the value of current_time by 1 every minute according to a clock, eg. 10:59 → 11:00

refresh_display() : this implements the 7-segment display library to display the current time to the screen.

date_display(digits) : this makes the display show digits. Used when modifying current_time and alarm_time

NB. the UI functions heavily use the above functionality. However, they are not included in the main file due to the different display requirements of each mode. e.g. display_on needs to update the clock, and refresh the display. change_alarm_time should only increase the clock, while displaying alarm_time instead of current_time

Initially, I tried to write my own code to drive the display, but my code proved to be slow, and caused the screen to flicker. Thus, I changed to using a library to handle display code for me. This was in fact quite simple as my programs were loosely coupled and independent: I could simply replace my display functions with the library functions with minimal modification.

Integration of Parts

The integration of parts was relatively straightforward, as the gear and system were already connected. I connected the transistor pin on the Arduino with the transistor on the motor board, while remembering to wire both breadboards to common ground.