

Pico Clock Project

Background

While looking through Thingiverse for something interesting to print my attention was drawn to a hollow clock design driven by a stepper motor under Arduino control. And then whilst looking at clock designs in general I came across a NeoPixel clock which used a sixty lamp NeoPixel ring to display the points on a clock face. I ordered a NeoPixel ring from Coolcomponents to see what I could do with it. My order also included a Grove light sensor which I hoped to use to detect ambient light levels. When it arrived I hooked it up to a Raspberry Pi Pico-w, which I already had, and found some nice Micropython example code to drive NeoPixels using the Pico's programmable PIO (leaving the main core free to do other things). I started to contemplate a design that had the following features:

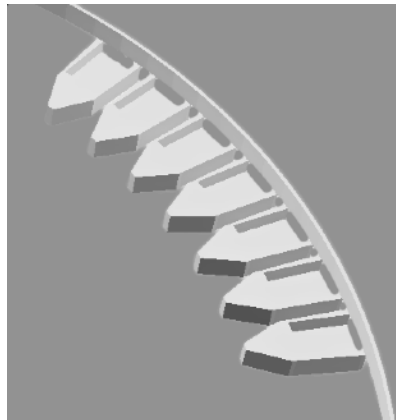
1. Hour and minute hands driven from their extremities, hollow clock style.
2. Seconds displayed on the NeoPixel ring.
3. Time set via a WiFi connection.
4. Based on RPi Pico-w running Micropython code.
5. Powered by USB connection.

Item 1 was all about designing some 3D printable parts that would mesh nicely together with a couple of stepper motors. And the stepper motors must be 5volt so that a standard USB could be used for power (item 5). To be sure of dimensions I ordered a set of 5 28BYJ-48 stepper motors and driver boards. These motors have a 64:1 gearbox and I believe are used in the ubiquitous Arduino hollow clock design. Although I only needed 2 of these a set of 5 seemed like the best deal.

Once the motors arrived from China I was able to proceed with some CAD. For no particular reason other than preference I always use OpenScad to capture my designs. The straightforward way that simple shapes can be defined for adding or subtracting suits my way of thinking. Also there is plenty of stuff available for generating gears and threads, etc.

The part of the design which caused me most trouble was making a light guide to turn the NeoPixels through 90 degrees. This entailed experimenting with transparent ('glass') PLA plastic filament. Although not entirely satisfied with the results it's good enough for what I wanted to achieve.

This render shows a section of the light guide which fits snugly over the NeoPixels.



Having found that the 'glass' PLA seemed more rigid than the normal stuff I decided to use it for the gears, gear tracks and motor mount. More by luck than judgement this resulted in the clock hands being like frosted glass which could be lit from their centre with an LED.

The only problem with the light guide as designed was that there was a lot of light spill from one light to its neighbours. This was partially solved by including small blanking plates in the front cover.

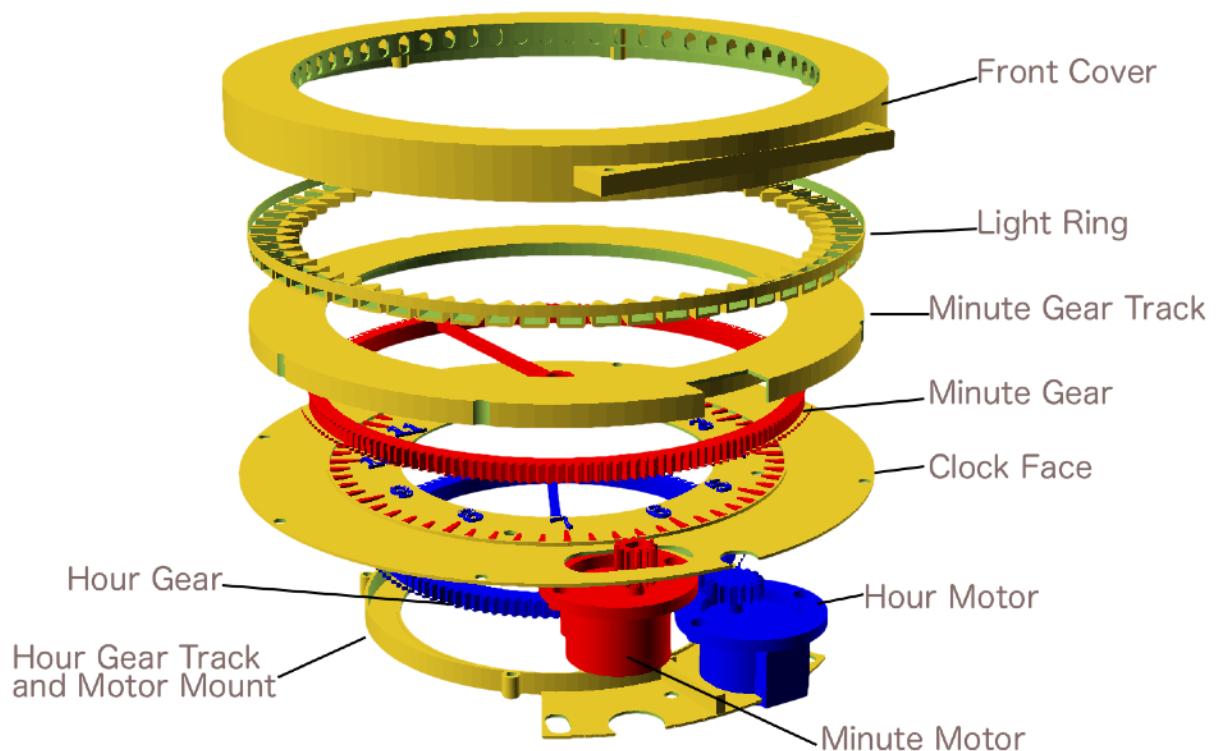
This render shows a section of the light guide fitted into the front cover.



The next problem to solve was the gearing. The motors required 512 full step pulses for a complete revolution which after some thought lead me to design a minute gear with 180 teeth. This meant that $\frac{1}{4}$ turn of a 12 tooth drive gear would turn the minute gear through 3 teeth. This gives $180/3 = 60$, i.e. 1 minute. A quarter turn of the motor requires $512/4 = 128$ full step pulses for 1 minute of movement. A similar sum resulted in an hour gear of 120 teeth with a drive gear of 16 teeth and a $\frac{1}{8}$ turn of the motor requiring 64 full step pulses per minute of movement.

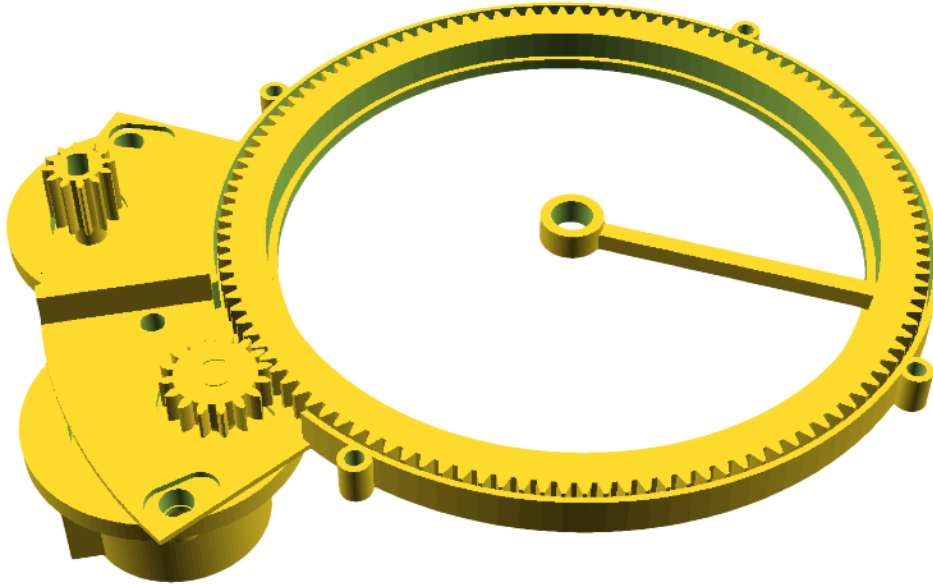
It may be obvious by now that the hands move 1 minute at a time i.e. 6 degrees. Once these gear ratios were fixed their diameter was adjusted by changing the gear module value (minute gear 0.8 and hour gear 0.9). Diameters, of course, were governed by the NeoPixel ring. During initial experiments and breadboarding of the design I decided to include 3 OLED displays which I already had. I considered one was necessary to show WiFi connection status on power up, and the others... why not? I also had a DS18B20 one-wire digital thermometer, the light sensor and a bright white LED to illuminate the hands from their centres. As the design progressed I couldn't really see how 3 OLEDs could be accommodated without overwhelming the clock face and I also couldn't think what an extra OLED would offer as regards information. Another consideration was that my OLEDs were SPI, 7 wire, devices that required a lot of interconnect. During the early breadboarding I swapped to 2 x I2C 4 wire OLEDs but these were slower and messed up the program timing. I'm not sure whether this problem was real or imaginary but I changed back to SPI.

Plastic Components



Motor Mount

This shows the back plate with the motors and the hour gear in place. Note the adjustable mount for the motors which are also raised (lowered in this picture) with plastic spacer rings.



Clock Construction

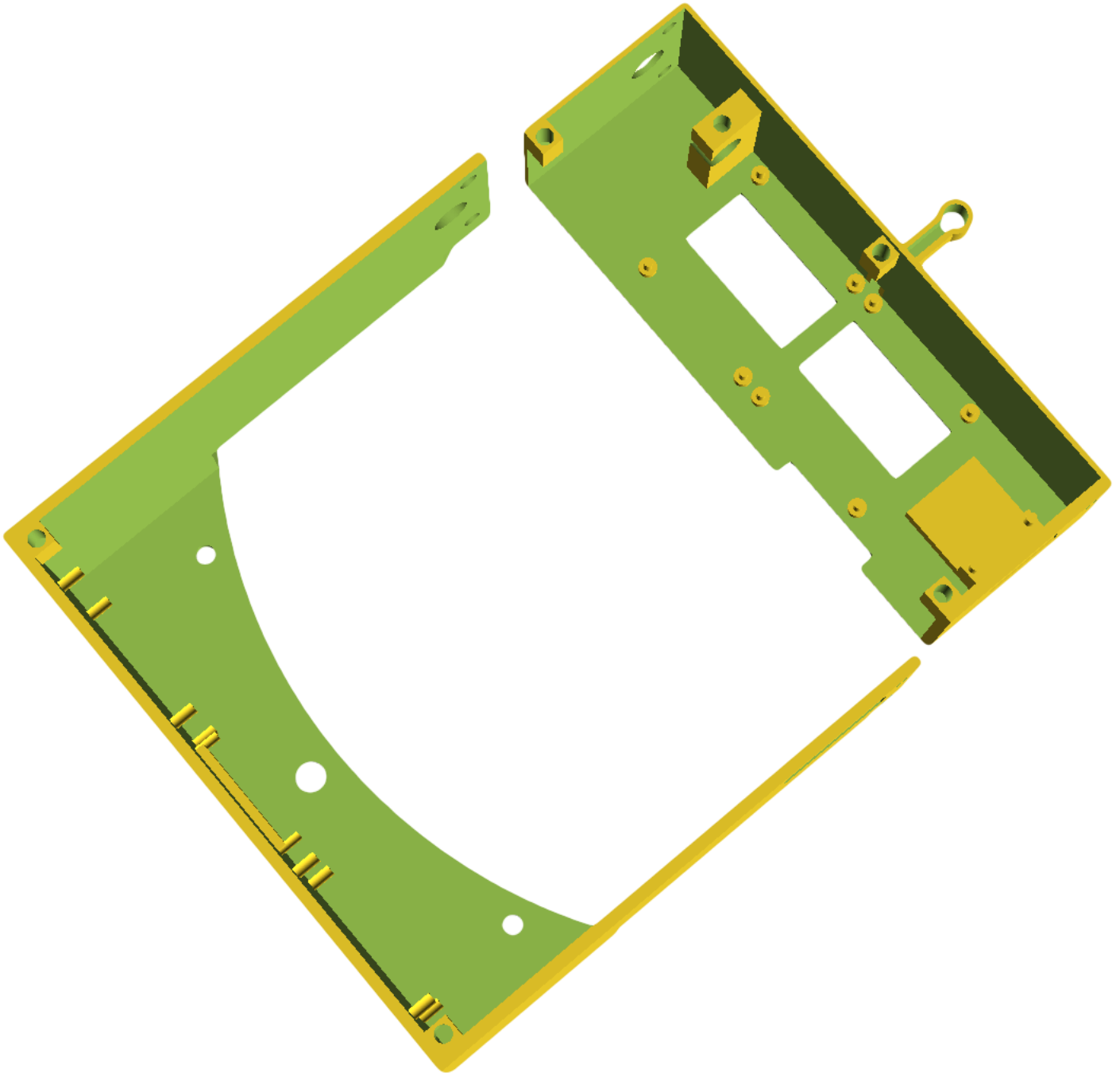
All parts are screwed together with 5mm long domed hex head M3 screws. These self tap into the plastic 2.8mm holes as printed. Most parts have been designed to lay flat on the bed with no support with the exception of the front cover which has an inset for mounting to the case. The case itself also has some support requirements.

The gears have to be dressed once printed due to first layer spread. The better this is done the smoother they will run. The drive spur gears are a tight fit on the motor shafts and will need relieving. The minute and hour hand guides should have any spurious bits of plastic removed. And to ensure smooth running the gears and guides benefit from an application of PTFE spray. The clock face numbers and minute markers are embossed for easy painting.

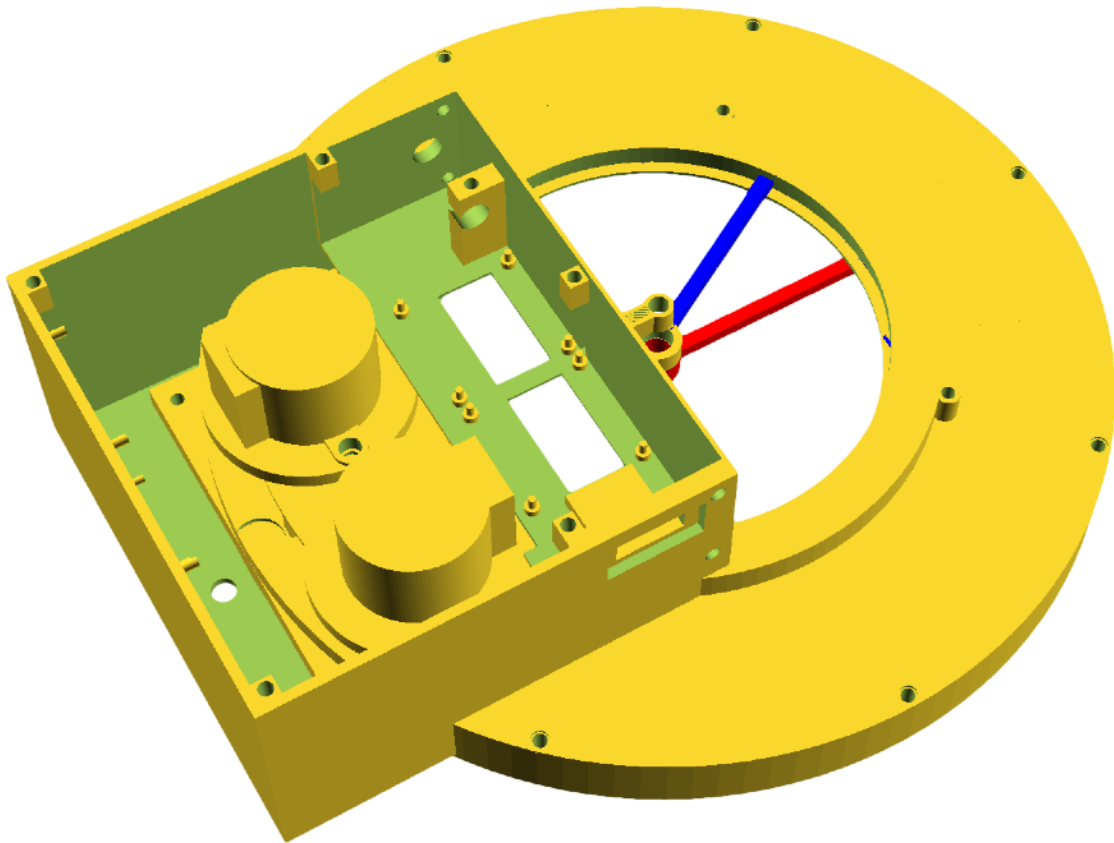
The light ring is fiddly to insert into the front cover. I accidentally broke the outer ring while trying to do this which made it much easier to manipulate. The trick is to feed it in at an angle making sure to align the next two or three sections as you go. It's a very satisfying fit once the circle is completed and clicks nicely into place.

Case

This is printed in two parts for ease of printing.

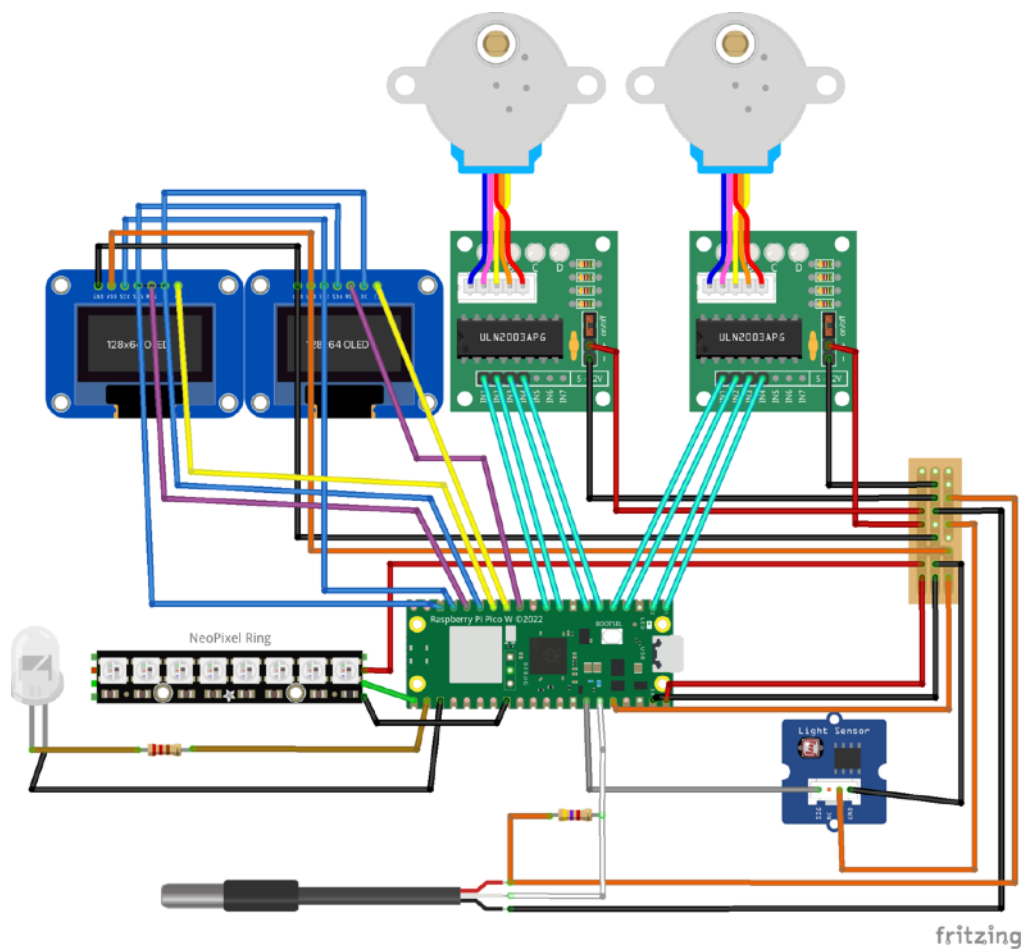


Case with clock mounted



Besides having space for all the components the case needs to give access to the Pico so that a USB cable can attach to it. In this regard the current design shown above leaves something to be desired, as the arrangement leaves the Pico reliant on two plastic lugs to hold it in place. Also the wiring together of the various parts proved somewhat fiddly with this design. Ultimately a PCB would be the answer, but for now this is it. The lugs on the bottom hold the stepper drivers and light sensor. The lugs around the OLED windows hold the OLEDs in place. The hole on the right side (as shown) is for the USB connection to the Pico which sits on two lugs on a small shelf. Also I mounted the Pico on a piece of matrix board to give space for the couple of resistors and their interconnect. The hole on the left and the adjacent bracket is for the temperature sensor. The hole at bottom centre is for the light sensor which I had to modify by extending the diodes reach with a couple of wires. It's intended that the lugs are melted with a soldering iron to fix components in place.

The circuit



The above diagram shows all the component parts and a connection solution used in the current design. As GPIO allocation is very flexible on the Pico this solution could undergo change where required as long as the software reflects the final interconnect.

As well as extending the light sensor as mentioned, I also removed the output wires from the NeoPixel array and used their socket to mate up with the plug on the input wires. This helped with the clock's construction.

Software Description

This is written using Micropython which is a subset of Python 3.

The basic program loops around checking the the real time clock, the temperature and the ambient light value.

Before the loop starts the Pico attempts to connect to a local WiFi connection. If this fails after 30 attempts the program stops. If it connects successfully a call is made to an NTP server to get the current time. If this fails the program stops.

If the time request from the NTP server is successful it's written to the Pico's real time clock, but regardless of this success an asynchronous client server is started.

The program loop contains a 0.6 second asynchronous sleep delay. During this delay the client server gets a look in and checks for client connections. If a client connects it's sent a web page containing various forms. The client server then waits for requests from the client, but hands back control to the clock until the next sleep.

The `get_time` routine, formats the OLED display strings and sets up the NeoPixel display for minutes, hours and seconds. It also checks for hand synchronisation, which should be correct after the first full minute of running.

The NeoPixel ring is wired such that position zero is where the input/output wires connect. Because the clock needs the wires to exit at the 6 o'clock position there is a NeoPixel offset applied to the PIO feed array.

If the clock hands aren't synchronised the `set_hands` routine is called. This reads a file stored in Pico's memory to get the last known location of the hands (this is updated every minute during normal running). The shortest route to their correct locations is then calculated and passed to the motor step routines.

It is worth mentioning at this point that the Pico has two cores. This is taken advantage of via Python threading to pass motor movements to the second core. The program waits for the first complete minute to start before hand synchronisation, allowing enough time for 30 minute hand adjustments.

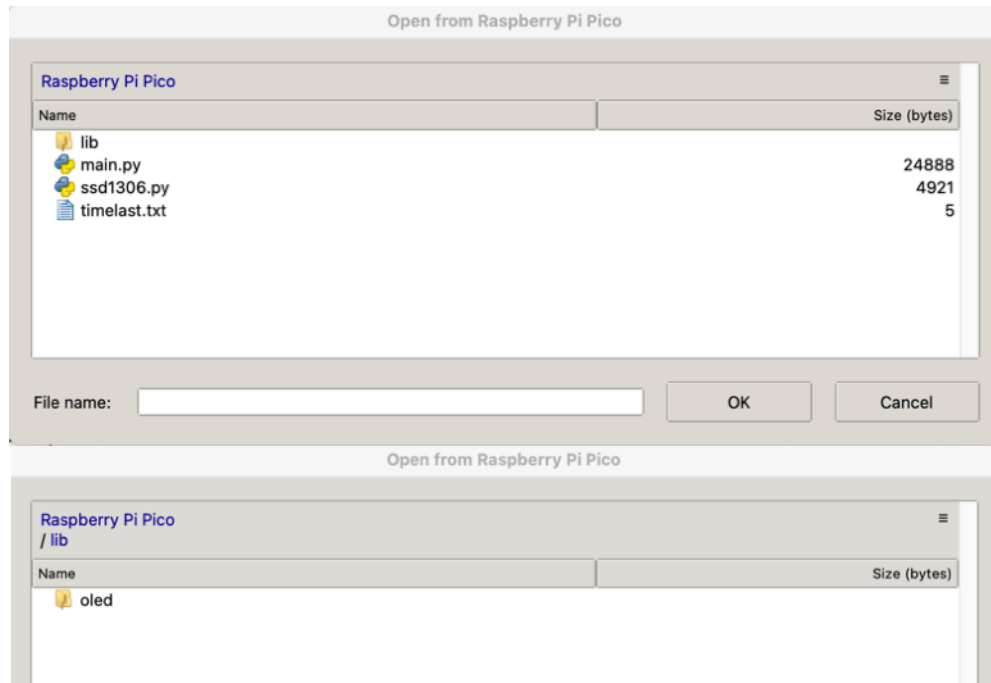
I've read that passing processes to the second core via threading can leave a trail behind so garbage collection is called in the main loop.

As an aside I tried using Pico's programable PIO to drive the steppers. After a lot of heartache and searching I managed to get this working. The advantage being that running the steppers via PIO not only takes the load off the Pico cores but also enables the simultaneous running of the two motors. This was full of promise and demonstrated the advantages mentioned well. However, I couldn't get the motors to run without mis-stepping so I ended up returning to the original solution. A real shame and probably worth a revisit sometime.

Setup

Before the software can be run on the Pico the Micro-python environment must be set up. This is a case of downloading the latest UF2 file from e.g.

<https://www.raspberrypi.com/documentation/microcontrollers/micropython.html> and loading onto the Pico. There are also some extra modules that are needed.



Shown above are the top directory and the /lib directory. The oled directory loaded into /lib and ssd1306.py shown here at the top level, but could also reside in /lib. These files are included in the software download but must be placed using Thonny IDE. The main.py file is the program itself, and timelast.txt file is written by the program and records the last position of the hands and the state of BST.

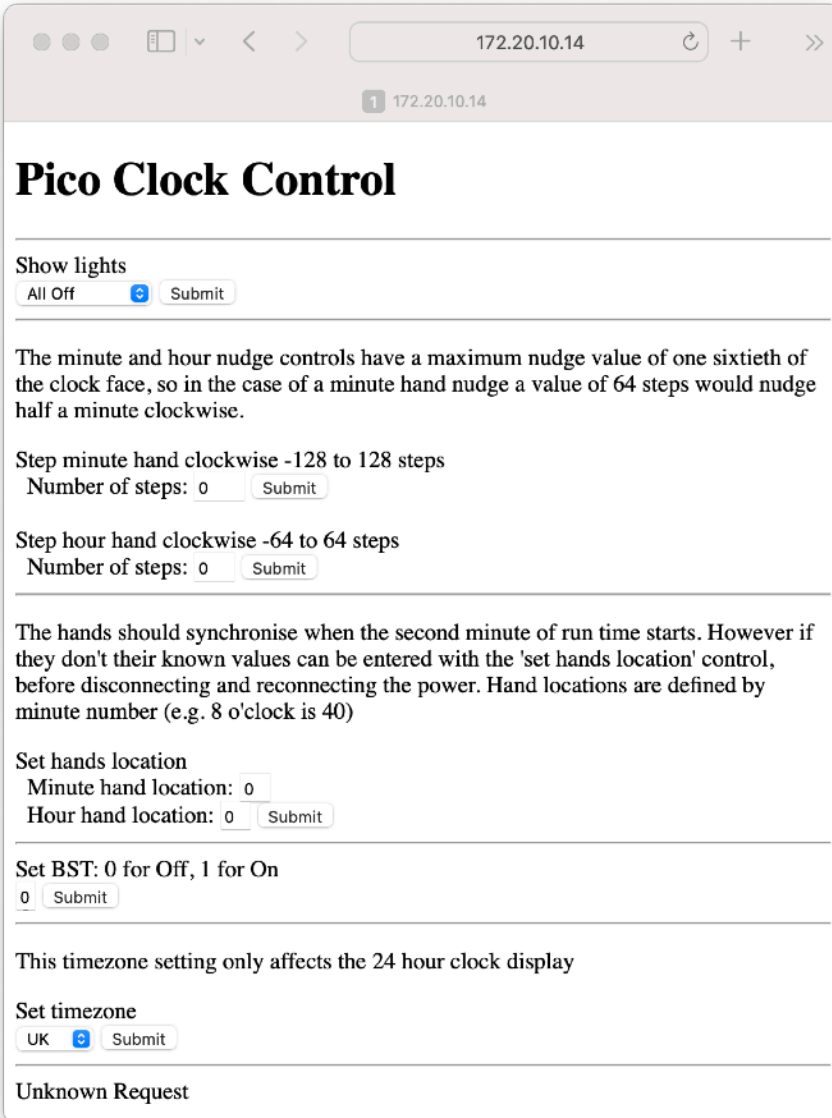
Commissioning

Your WiFi SSID and PASSWORD must be filled in (line 30).

There are some routines hashed out just before the main loop for testing the motors, recording the hand locations and nudging for proper alignment. The nudge and set hands controls are available on the web interface which you can connect to via a browser on the address shown when the clock connects.

Web Page

On connecting a browser to the clock's address a web page is loaded as shown below.



The screenshot shows a browser window with the address bar set to 172.20.10.14. The page title is "Pico Clock Control". The interface includes several control sections:

- Show lights:** A dropdown menu currently set to "All Off" and a "Submit" button.
- Step minute hand clockwise -128 to 128 steps:** A text input field with "0" and a "Submit" button.
- Step hour hand clockwise -64 to 64 steps:** A text input field with "0" and a "Submit" button.
- Set hands location:** Two text input fields for "Minute hand location" (0) and "Hour hand location" (0), followed by a "Submit" button.
- Set BST: 0 for Off, 1 for On:** A text input field with "0" and a "Submit" button.
- Set timezone:** A dropdown menu currently set to "UK" and a "Submit" button.

At the bottom of the page, there is a message: "Unknown Request".

This displays several submit forms which are self explanatory and offer NeoPixel display options, Minute and hour hand nudge values, initial hand location setting, BST status and timezone setting.

These are my selection of what seemed useful and can be easily changed in the program.

The nudge and hand setting forms are very useful when commissioning and correcting the clock.

The BST setting enables the overwriting of the programs 'on' state and the new state is stored along with the hand locations. This change immediately moves the hour hand but doesn't correct the display until the next minute is entered. When the clock is restarted BST goes back to its 'on' state regardless which requires a couple of minute cycles to sort the display out.

Set timezone drops down a list of timezone names which are stored in an array in the program along with their shift from UK time. This has no effect on the hands, just the time of day display. The time of day in words stays aligned to UK time.

The 'Unknown Request' line shown above changes to show the results of a valid request.

What Next?

This project must be seen as experimental in nature. Although I've produced a working version there's still plenty to do.

There are many possible enhancements that could be made to the clock considering its WiFi connection and web interface. I wondered about passing it extracts from my Apple calendar's events for display but at first glance it looked a bit complicated.

The way that the NeoPixels are used is down to artistic imagination. The current modes and ambient light levels are open to improvement.

A PCB would be a bonus and I might work on that one day.

The software is definitely in need of a rationalisation and general tidy up. It was an organic development based on what worked as I picked my way through possibilities.

I hope however that this project inspires others to embrace the Pico and all its capabilities.