Smart Thermostat/Knob


Description

This smart knob offers precise control over various appliances using a smooth, 360-degree spinning knob and a push button. Its on-board Wi-Fi allows seamless integration with existing IoT ecosystems. For example, instead of directly managing your home's temperature, the knob can connect to smart thermostats like Ecobee to adjust settings. The device dynamically adapts its function based on the connected device. When plugged into a laptop, it controls elements like volume and brightness, while wall-plugged, it manages room lighting via Wi-Fi. If time allows, it will also feature either a desktop or mobile app to edit these settings and add a customized user experience. This will be done via Electron for a desktop application, or Expo for a mobile application. Electron allows traditional HTML, CSS, and Javascript to interface with a computer’s native API, while Expo allows for HTML, CSS, and React Native to run on iOS and Android devices.

The integrated display enhances the user experience by indicating the selected mode. When connected to a laptop, it could show the volume percentage. The display can also function as a standalone infotainment screen, displaying the time, weather, or other data. A 0.96-inch OLED

module with 128x64 resolution, connected via I2C, will be used for graphics. Seeing as this screen is a rectangle shape, it will be essential for me to create a visually appealing bezel that shapes it into a circle and masks out its pcb. This will make the device look more like a knob, rather than a rectangle.

The project utilizes an Adafruit QT Py ESP32 Pico as the microcontroller, chosen for its built-in Wi-Fi, Bluetooth Classic + BLE, and ample GPIO pins. A KY-040 rotary encoder handles the knob's rotation, and a tactile push-button switch, located beneath the knob, provides another input method. Finally, a NeoPixel Ring (16 x RGB) will add an underglow effect, visually indicating specific settings. The Ring will have lights dynamically turned on or off to indicate a progress bar that is used for things like showing the user which direction they are turning in, or indicating the temperature.

Links & Resources

[DIY Multifunctional control knob using Pi Pico]

The creator of this project demonstrates how it is possible to use a raspberry pi pico to control the brightness or volume of a computer. While this isn’t using an Arduino, the concept is the same and it can easily be adapted to work with my ESP32 Pico. I will use this resource to learn how a rotary encoder is utilized and read from a microcontroller

[DIY haptic input knob: BLDC motor + round LCD]

[Building a haptic input knob from scratch! - YouTube]

The creator of this project is very talented and was able to create a knob that has haptic touch feedback through motors. This is out of my expertise, however the creator shows their design process, as well as how the rotation piece works. Since I will be using a Rotary Encoder, I will not be implementing haptic feedback, which will make this much easier to replicate. My vision is to have a similar layout where the underglow is able to provide information, and the display tells the user what is happening. This video will teach me how to implement a beautiful user interface into a display.

[(15) They Never Shipped, So I Built My Own Focus Dial from Scratch - YouTube]

This device actually doesn't rotate, however the creator documents how they created a case for a knob, as well as using the same NeoPixel Ring to show progress. The creator tells us how they were able to design a case on a CAD software, prototype the circuit, as well as provide us with code for the display. I will use this resource to structure my design, as the way they stacked each component is very compact and user friendly.

[ecobee API]

The thermostat in my home is from ecobee, and fortunately they have an API that is free to use and interface. I can use this to read and write temperatures to my house over wifi. This will allow the smart knob to change the temperature in my house.

Materials (Total Cost of ~ $66)

[ESP32-S3 2.8inch Round Display]

$39.99

This is the microcontroller that will be powering the smart knob. It also has a display that is directly connected to the microcontroller, which will make it easier

for case design, as I will no longer have a thick wire connecting between a microcontroller and a display. I also will have an easier time creating a circular design, as I am not stuck with a rectangle display.

[KY-040 Rotary Encoder]

$9.99

This is what will allow the knob to rotate 360 degrees. The Rotary Encoder is how the smart knob will know which direction the user is rotating, as well as how much they have rotated. I will only need one rotary encoder, and it doesn't matter which one I use, so I am more than willing to use the ones that the school has.

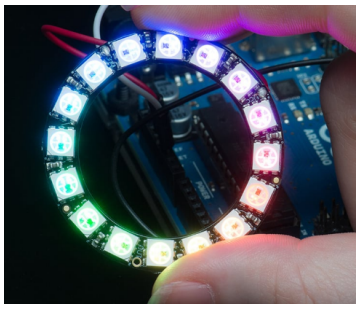
[Tactile Push Button 20pcs]

$5.99

This is what will give a click input to the knob. This will be utilized as a second input for conformation or mode switching. I will only need a single push button, and this is something I know the school already has.

[NeoPixel Ring - 16 x 5050 RGB LED]

$9.95



This will add underglow to the smart knob to make the aesthetics more pleasant, as well as show the direction that the knob is turning

Housing for the smart knob will be 3D Modeled then printed

Timeline

1) Obtain the parts either online or/and from the school's collection of parts
2) Set up a breadboarded prototype that is able to utilize the rotary encoder to change volume on computer
3) Look into most efficient and compact way to package into a single device
4) Begin CADding the device
5) 3D Print Cad Files to house all electronics
6) Create the production code that has all features (working underglow, realtime info on OLED display)
7) Solder everything to microcontroller and begin building final product
8) If time remains, create a desktop / mobile UI to edit and manipulate the smart knob.

Progress Update #1

Over the Spring break and the week following, I began the initial construction of my Smart Knob. I started with breadboarding the circuit on my Arduino UNO, as that way I could have a test device that had the functionality I was looking for. This circuit contained the ring light and a potentiometer, as I wanted to make sure that I would be able to control each individual light. The final version of this project will utilize a rotary encoder, however being that I was on spring break, I did not have one on me in my kit. Attached below is the breadboard + video example.

Breadboard + Video

Challenges

- Having the ring light remain connected properly

    This was the only main issue I had. The Adafruit ring light requires you to solder directly to the back of it in order to connect with an arduino. I only had GPIO cables, so I had to bend them in order to have them stay connected to the ring light. This made them flicker occasionally, as they weren't fully connected.

Lessons Learned

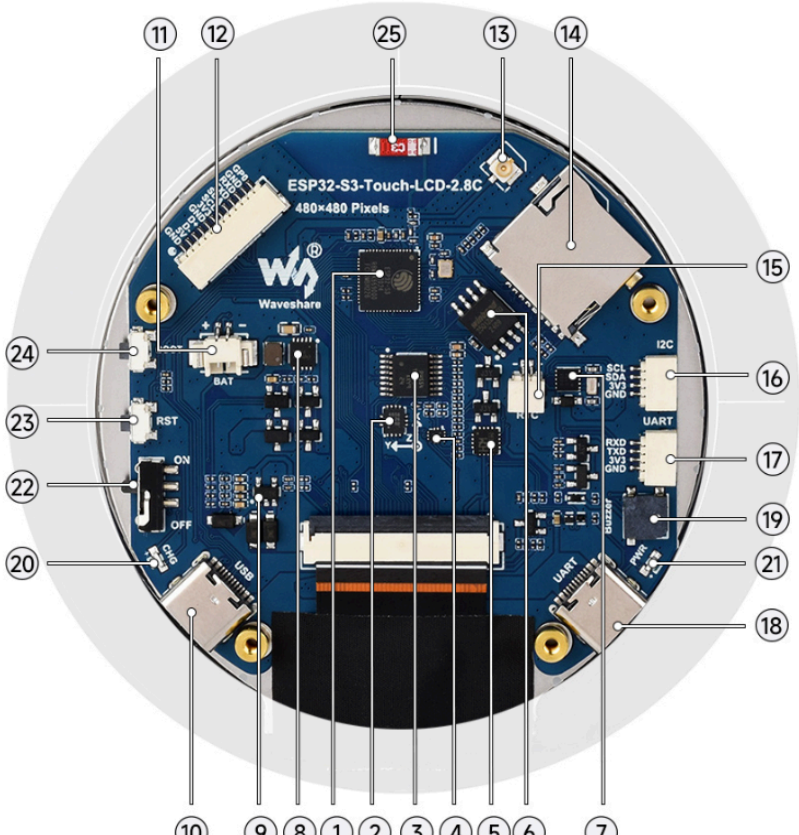
- ## Utilizing foreign libraries in Arduino IDE

I had to properly install the Adafruit NeoPixel Library in order to communicate with the ring light. This skill became essential in later portions of my project

- ## Mapping potentiometer values to a custom scale

I learned the correct way to utilize the map function in Arduino IDE, and used it to scale values from 1 - 1023 to a new scale of 1-16

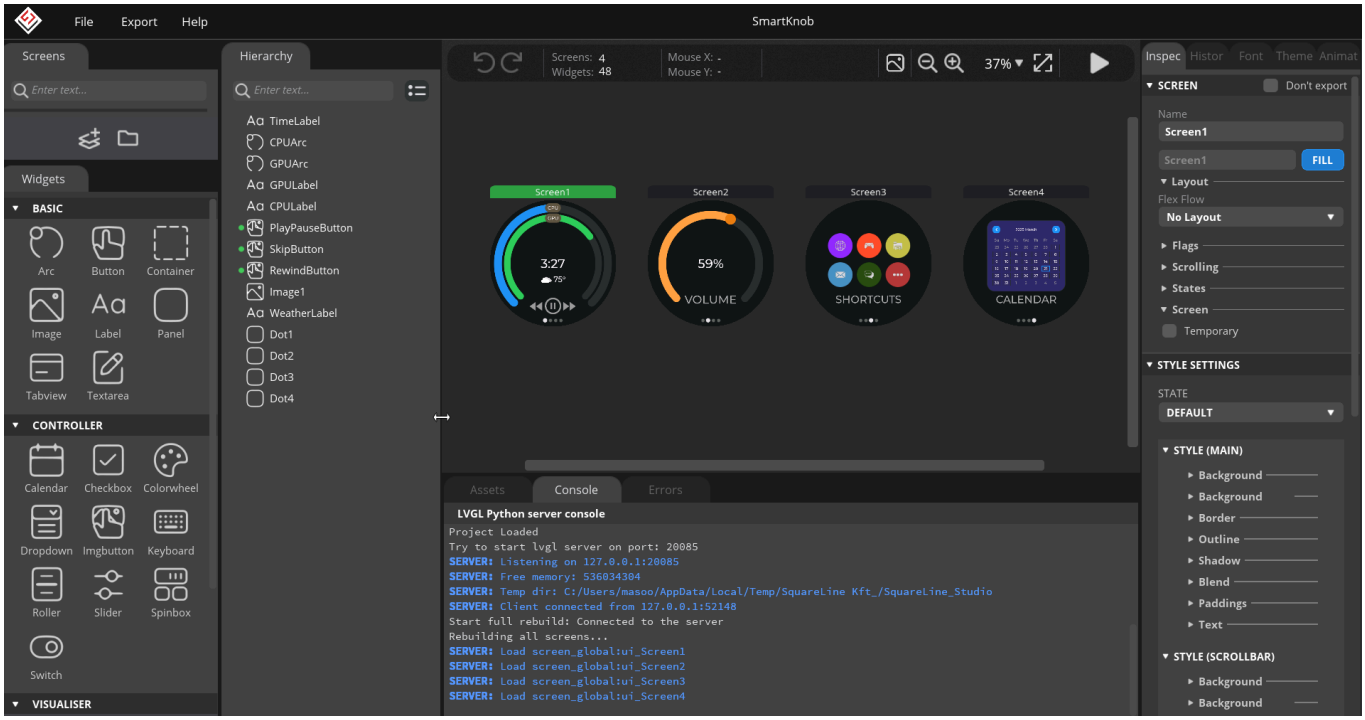
The next thing I worked on was learning how to utilize my ESP32-S3 board + TFT Display. This proved to be a major roadblock, as I was constantly having the board disconnect on me due to power issues, and being unable to communicate with it, as commands such as Serial.println were not working. I began to research online and ask some of my peers who had experience with microcontrollers, and after 2 days of digging, I realized that the lack of communication was due to me plugging into the wrong port on the ESP32-S3. As seen in the diagram below, my microcontroller has 2 usb type c ports: A

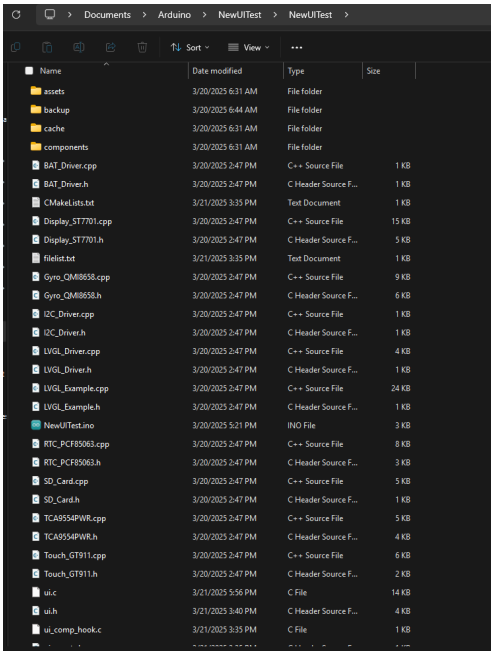


generic one, and one that has the text UART printed under it.

I learned that when the UART port is available, one MUST plug into it for communication between a host device and the microcontroller. This is due to the other port only being used if native USB / CDC ACM is being utilized, which can only be done on microcontrollers with only a single usb port. Boards like the ESP32-S3 include two ports so that UART communication can be used while still having a spare usb port open for HID or other usb functionality.

After understanding this crucial information, I was able to begin creating the user interface for my display. I researched online what the best way to do this was, and I found out that a library called LVGL is what most professionals use when creating an interface for arduino applications. I also found out that there are softwares such as SquareLine Studio that allow you to create such an interface in an Adobe Photoshop-like manner, where one can drag frames or elements, and scale them to one's liking. After doing so, one can export this into native code, which can then be compiled in Arduino IDE to send to the ESP32-S3. After tinkering around with this application for a few hours, I was able to create the following UI for my smart knob.





After pressing export, the native folders and files are generated, and can then be implanted into the Arduino project directory, as seen in the next screenshot. The challenge arises when compiling this, as it is vital that the .ino file contains the native code required for your specific microcontroller, and unfortunately, the board I am using from waveshare had little documentation, so I had to spend a bit of time understanding how this worked for other boards, and puzzling my way through for doing this on my own board. After a while, I found out that I had to first download the Demo project, then replace all of the files with the new ones generated by SquareLine, then alter some of the settings in the configuration files in order to

have a successful compile. This seemed to do the trick, as now I am capable of utilizing this interface on my physical arduino, which I will demonstrate in a bit.

Challenges

- ### Communicating between my laptop and the arduino

  The solution to this was not obvious at all, I had to spend countless hours researching online, only to find out that the solution was to simply plug into the alternate usb c port.

- ### Compiling the User-Interface in Arduino IDE

  Waveshare did not provide any explanation on how to do this, only 3 sentences in the entire documentation explained how to do this, and that didn't even work. I had to youtube this for a different board, and follow along with a custom installation in which I replaced the files in the Demo Project provided by waveshare.

Lessons Learned

- ### The difference between UART and USB CDC

I was not aware of the difference this made in communication with commands like Serial.println(), and responding to this from my desktop with python code that called functions like Serial.write()

- ### Creating UI and having it properly scale to the physical world.

I learned how many real products in the real world are able to create such beautiful interfaces that work so well. I thought this was something super complicated that required only pure code, but having a drag and drop editor seemed to simplify the process, and made me more confident in my skills.

After creating the UI, I moved on to adding functionality to this interface. When I created this, all of the text was all dummy code. For example: The CPU and GPU utilization bars did not update according to the status of my laptop, the time did not auto update, the volume bar did not function, and the weather was not accurate (I still have not coded this part, I will be finishing that soon). I realized that if I wanted to accomplish this, I would need to send information from my laptop the arduino, and learned that this was possible through python. I began writing a python script that responded to the messages sent by the arduino, and this is attached below:

Raw Python Code

Essentially, my code is constantly listening for certain commands, while also providing the arduino with information such as CPU and GPU utilization, and the Arduino is then saving this data locally (in RAM, not to Storage) and then displaying this in various manners.

Below is a video demonstration of the ESP interacting with the host device.

Demonstration

Note: Currently I have not coded the weather, or the play/pause/skip/rewind buttons. That will be done soon.

Challenges

- Having the volume on the ESP32 update when the user changes the volume

    It was relatively simple to have the volume on the laptop update when it was changed from the ESP's Screen, but I had to figure out how to have the ESP update if the volume was altered from the host PC through the windows volume bar. This was solved by detecting a change in volume when it wasn't through the code, then sending a command to the ESP, then detecting that command and updating the values.

Lessons Learned

- Proper allocation of resources for python scripts

I realized that if the loop ran too frequently, the host device would have a negative effect on performance, and this could cause utilization to spike, and reduce the performance on the laptop. This was countered by adding some waits in the code, and not constantly updating the values if they weren't required. Originally, the time was sent to the ESP every second, however I realized that the increment of time could be done locally, rather than needing to send this to the ESP. Doing so significantly boosted the performance of my script.

The next step in my project was to get the scrolling between pages done through a rotary encoder. A rotary encoder is able to read the rotation of a rotating component, and with the use of my microcontroller, I can use these values to detect direction, and speed. My plan was to use

these values to scroll between the pages, similar to how a thermostat works. Unfortunately, I was stuck for a while because I lacked the knowledge on how to connect this to my specific microcontroller. Traditionally, the power, ground, and switch pins are connected normally, and the direction and distance pins are connected to PWM. I realized that the PWM-specific pins on my board were constantly being occupied, because I was using the I2C pins for my display. Connecting to these pins caused errors in the Serial Monitor, and after a while I found out that the solution to this is to just connect to the general pins, or GPIO. The ESP32-S3 is capable of using any of these pins for PWM, which upon connecting to, made my code work instantly. Because of this issue, I had first constructed my design on an Arduino UNO, and when I noticed that my code was working perfectly fine there (as noted in the video below) but incorrectly on my ESP, I had to look online for a solution. A member of the Arduino Discord Server had informed me of the use of GPIO, and by following his instructions, I was able to get my code to work.

Issue with ESP32-S3 + Rotary Encoder

Working Demo on Arduino

Final Working Demo

Challenges

- ### Allocating pins to make sure I wasn't wasting any

  Seeing that I had also purchased a Ring Light, I have yet to attempt to connect this to my microcontroller, however I have had success connecting this to the Arduino. Now that I know that I can use any of the GPIO pins, I should have enough to power my right light while still being able to read values from the potentiometer. I am not sure if

there will be power issues while running all of these components at once, which is why I will be focusing on this next.

Lessons Learned

- ## Learning from the experience of others

Frequently, I try to attempt solutions on my own but rarely ask others for help. In this scenario, I had tried everything that I could do on my own, and realized that the only solution would be to ask someone who is more experienced with microcontrollers than me. After a bit of debugging with this person, I was swiftly able to find the solution to this troublesome issue, and continue on with my project. This is an important skill as I continue in my development career, and I recognize that this is one of my weaknesses.