Weekly Journal #03

Week Overview (Monday 25 - Sunday 01)

Planned Tasks:

- **Enable Tactile Buttons for Picture and Video Capture**
- **Complete the Web Server Implementation**
- **3D Print the Case for the Raspberry Pi and Camera Module**
- **Assemble All Components**
- **Finalize the Slide Presentation**

Monday, November 25

Completing The Web Server Implementation

1st attempt: Initially, we tried installing nginx as a reverse proxy and then setting up PiGallery2 to handle requests. The first method my colleague attempted was a direct installation, which meant downloading Node.js and running PiGallery2 natively. Everything seemed to be going well at first:

- She installed Node.js.
- She installed PiGallery2 from source via GitHub.
- She edited the configuration files.

However, she encountered an issue. During the setup, she got confused and turned to ChatGPT for help. ChatGPT mentioned that she needed to retrieve a file called `Index.ts`, which she had, but when she attempted to proceed, it kept telling her that the index file wasn't found. After more investigation, she suspected that she hadn't installed all the necessary dependencies, which caused this error. Despite researching this further, she couldn't solve the problem, so she decided to abandon this method.

2nd attempt: For the second attempt, she switched to the recommended and easier option: running PiGallery2 in Docker. These were the steps she followed, which are also outlined in the GitHub repository:

Install Docker:

```
curl -sSL https://get.docker.com | sh
```

Install Docker Compose and dependencies:

```
sudo apt-get install -y python3-pip
sudo pip3 install docker-compose
```

```
sudo apt-get remove python-configparser
sudo pip install docker-compose
```

Set up PiGallery2:

```
mkdir ~/pigallery2
cd ~/pigallery2
```

Create a `docker-compose.yml` file inside the PiGallery2 directory.

Run the following command to start PiGallery2:

```
docker-compose up -d
```

Configure nginx as a reverse proxy:

```
sudo nano /etc/nginx/sites-available/pigallery2
```

Enable the configuration:

```
sudo ln -s /etc/nginx/sites-available/pigallery2
/etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

After completing these steps, she was able to access the web server at http://10.0.0.247, the IP of the Raspberry Pi, and everything worked smoothly. This method was much faster than the first one. However, one drawback of using Docker is that it makes our server dependent on the person who created the PiGallery repository. If they decide to remove it in the future, our server will be affected. This wouldn't happen with the first option, where everything is installed natively.

Adding More Buttons:

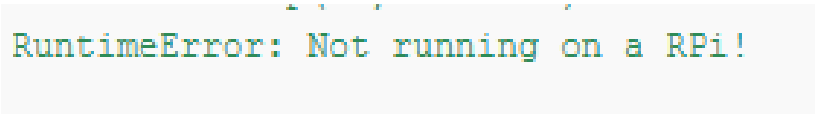
Previously, we installed a button that takes pictures when clicked. Now, we wanted to add a second button to start and stop video recording. We also considered adding a third button to turn the Raspberry Pi on and off. However, we realized that once the Raspberry Pi shuts down, it cannot be turned on again without manually plugging it back in. For this feature, we would need a power switch button, but since we couldn't afford to wait for the component to arrive, we abandoned this idea. We also considered putting the Raspberry Pi into a sleep or hibernation mode, but this feature is not supported by Raspberry Pi. The last option we looked into was shutting it down softly with the command `sudo shutdown -h now`, but it still fully shut down the Raspberry Pi, though more cautiously than using `sudo shutdown now`.

Encountering a Big Problem:

While working on a 6-pin switch button, she used a resistor and tested a Python script that was supposed to start recording when the button was pressed and stop recording when clicked

again. However, it wasn't working. She switched to a 4-pin button, which we had previously used, and ran a different script. This time, she received an error that didn't make any sense.

Image of the RuntimeError:



After researching the error, she found that it could be related to either the Raspberry Pi itself or read-write permissions for accessing the GPIO pins. She found an article that provided a solution, but her attempts didn't work. One possible reason for this could be that SSH wasn't enabled, but she didn't know this at the time. The issue was still unresolved, and we continued troubleshooting:

Link to error explanation and guide

AT HOME:

While Lydia was working on the Raspberry Pi setup, I started working on the slides for the presentation. I focused on creating a clear and organized structure for the presentation, making sure to highlight key aspects of our project, including the Raspberry Pi setup, the installation of PiGallery2, and the challenges we faced with the buttons and camera module. I also included steps from our troubleshooting process to demonstrate how we identified and resolved issues along the way. The slides were designed to be informative and easy to follow, with visuals that would help convey the technical aspects of our work.

Conclusion:

At this point, we had one SD card that recognized the camera module but couldn't recognize the Raspberry Pi. On the other hand, we had another SD card that could detect the Raspberry Pi and GPIO but not the camera. This created a serious problem for our project since we needed both the camera and Raspberry Pi to work together.

Tuesday, November 26

Researching for advices

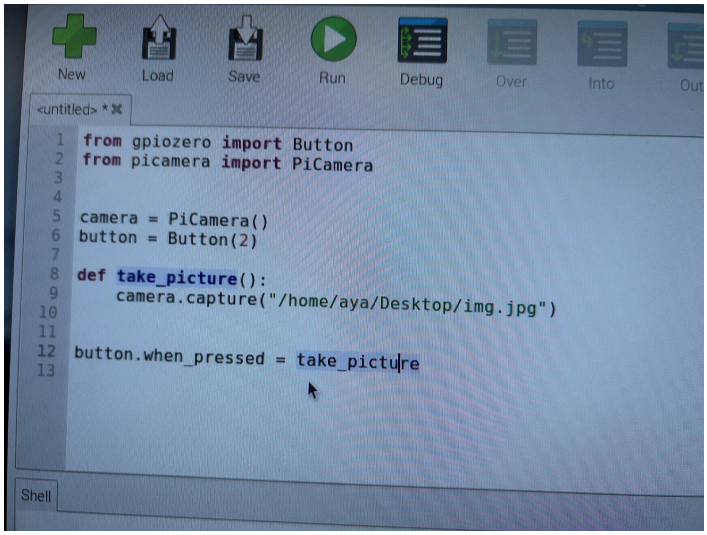
We visited the STEM center at Vanier College to see if anyone from the Computer Engineering department or the robotics club could help us with the problem. A student there mentioned that Angelos has a strong knowledge of Raspberry Pi. He suggested that we reflash the Raspberry Pi to resolve the issue.

Friday, November 29

Completing the button setup by reactivating the camera module on the final microSD card.

When working with Danat's SD card, the camera module was being detected, which indicated that the hardware was properly connected. However, we faced issues when running the script—the buttons weren't responding as expected. The script reported that it was not running on a Raspberry Pi, even though we were clearly using one. This led us to suspect that the issue was with the script's environment or configuration rather than with the hardware.

Image of the script for taking an image and the error output:




```
RuntimeError: Not running on a RPi!
```

In contrast, when we tested the script on my own SD card, it correctly recognized the Raspberry Pi and the buttons worked as expected. However, the camera was not detected in my case, pointing to an issue with the camera module settings or the operating system.

The root cause appeared to be a mismatch between the software environment and the hardware configuration. While Danat's SD card detected the camera module, the script failed to recognize the Raspberry Pi. Conversely, my SD card recognized the Raspberry Pi, but the camera module wasn't detected. This confirmed that the problem was related to the operating system and missing libraries.

To avoid risking data loss on Danat's SD card, we continued using my SD card for testing, as it was the original card where we had issues enabling the camera. For the test, we used a different, simple script that printed a message when the button was pressed. This script worked

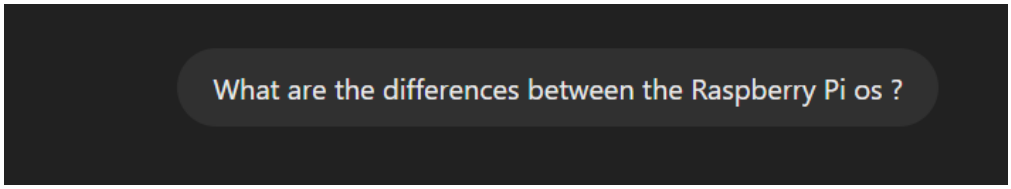
as expected, confirming that the buttons and breadboard were functioning properly. At this point, we ruled out hardware issues and focused on the software setup.

The script:

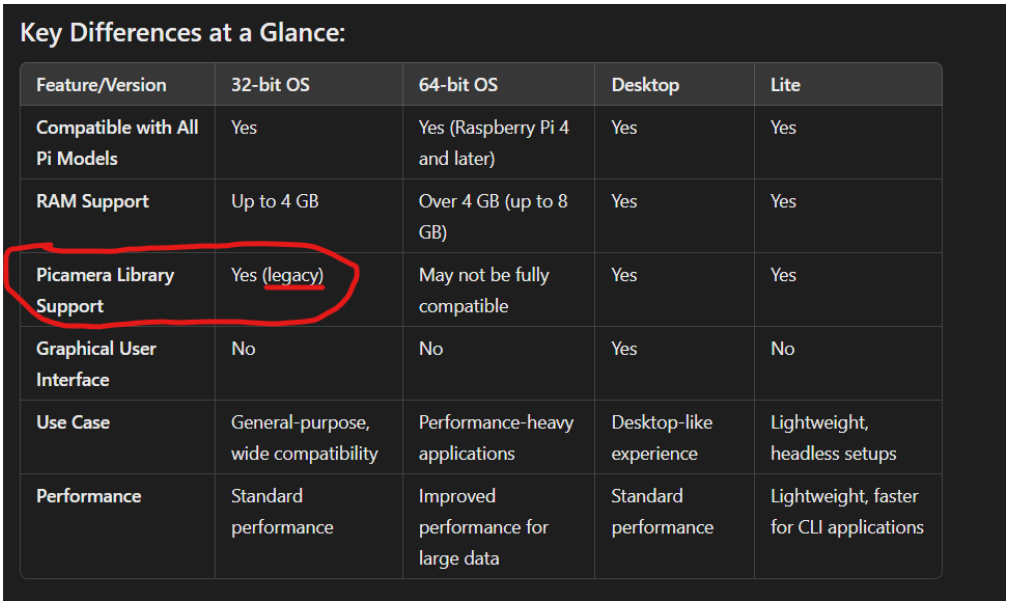
```
from gpiozero import Button
button = Button(2)
def button_pressed():
     print("Button was pressed!")
button.when_pressed = button_pressed
```

We began troubleshooting by ensuring the necessary libraries, specifically the python3 library, were installed. We installed the library, but the issue persisted. We then considered that the operating system might be the cause since Angelos told us Tuesday that maybe we needed to reflash the SD card, so we consulted ChatGPT about the differences between OS versions. It turned out that the 32-bit legacy OS included the required picamera library, while the version we were using (32-bit) did not.

Input to ChatGPT:



Output from ChatGPT:



Key Differences at a Glance:

Feature/Version	32-bit OS	64-bit OS	Desktop	Lite
Compatible with All Pi Models	Yes	Yes (Raspberry Pi 4 and later)	Yes	Yes
RAM Support	Up to 4 GB	Over 4 GB (up to 8 GB)	Yes	Yes
Picamera Library Support	Yes (legacy)	May not be fully compatible	Yes	Yes
Graphical User Interface	No	No	Yes	No
Use Case	General-purpose, wide compatibility	Performance-heavy applications	Desktop-like experience	Lightweight, headless setups
Performance	Standard performance	Improved performance for large data	Standard performance	Lightweight, faster for CLI applications

To resolve the issue, we reflashed the SD card with the correct OS version and proceeded to enable the camera module in the Raspberry Pi configuration menu (raspi-config). After rebooting the Raspberry Pi, we ran the script again, and both the picture and video capture functions worked perfectly. Finally, we combined both scripts into one to streamline the process and complete the button setup with the camera module reactivated.

Steps to Flash SD card:

1- Flash: Go to the raspberry Pi Official website

2- Go to the raspberry pi imager page

3- Pick your computer OS (Windows, MacOS, Ubuntu and even Terminal)

4- Download

5- Open the Downloaded file and accept all the terms

6- Raspberry Pi Imager should now be downloaded and available to use

7- Click on it, and accept that it should make modifications on your computer

8- Insert an SD card using an adapter or by directly inserting it to your computer

9- First:Pick your raaspberry pi devive

10- Second: Pick your OS system (64-bit, 32-bit, legacy 32-bit, etc...). We took the Legacy 32-bit since it comes with the picamera Library

11-Thirdly: pick your SD card storage

12- Flash

---------**Enabling the camera**----------

1- Go to the terminal: sudo raspi-config

2-On the Rspberry Pi configuration tool, select interface options

3- select legacy camera and enable it

4- reboot (sudo reboot or using the user interface)

Image of the script to take pictures:

```
from gpiozero import Button
from picamera import PiCamera
from datetime import datetime

camera = PiCamera()
button = Button(2)

def take_picture():
    timestamp = datetime.now().strftime("%Y-%m-
%d_%H-%M-%S")

camera.capture(f"/home/danat/Desktop/image{timesta
mp}.jpg")

button.when_pressed = take_picture
```

Image of the script to take video:

I don't have it anymore since we merged the picture script and video together but you can find it below in the final script.

Final Script:

```
from gpiozero import Button
from picamera import PiCamera
from signal import pause
from datetime import datetime
import time


button = Button(2)
deux = Button(3)
def gettimestamp():
    return datetime.now().strftime("%Y-%m-%d%H-%M-%S")

#Picture
def takepicture():
     timestamp = get_timestamp()
    file_path = f"/home/aya/Desktop/img{timestamp}.jpg"
    camera.capture(filepath)
    print("took picture")

deux.when_pressed = take_picture

#Record
recording = False
clicks = 0
last_click_time = 0

def toggle_recording():
```

```
def toggle_recording():
    global recording
    recording = not recording
    if recording:
        timestamp = get_timestamp()
        video_file = f"/home/aya/Desktop/video{timestamp}.h264"
        print(f"Recording: {video_file}")
        camera.start_recording(video_file)
    else:
        print("Not recording")
        camera.stop_recording()

def button_clicked():
    global clicks, last_click_time
    current_time = time.time()

    if current_time - last_click_time <= 0.5:
        clicks += 1
    else:
        clicks = 1

    last_click_time = current_time

    if clicks == 2:
        toggle_recording()
        clicks = 0

button.when_pressed = button_clicked
print("Ready for double click")
pause()
```

When pressing the first button twice quickly toggles video recording, while the second button captures a picture and saves it with a timestamp.

About the Webserver

We also realized that we need to start the Docker container when the Raspberry Pi boots up. To do this, we can create a systemd service that will automatically start the Docker container including a timestamp to ensure it always has a unique name. Also, running PiGallery in a Docker container has some downsides, as it depends on the person providing it. If they decide to shut it down, our server will also go down, meaning our gallery is heavily reliant on them.

Here are the options we considered:

1. Keep using Docker and create a systemd service to start it on boot.
2. Reinstall PiGallery2 natively on the new SD card and restart the webserver.

Saturday, November 30

Printing the 3D case for my Raspberry Pi and camera module

When I tried to 3D print the case template for my Raspberry Pi and camera module, I downloaded the design from a website and started the printing process. Unfortunately, the printing stopped unexpectedly after one hour, and I couldn't figure out the reason why it happened. I asked my father to help me troubleshoot the issue. He checked the 3D printer and the setup, but he couldn't identify the exact cause of the problem.

Since we weren't able to resolve it ourselves, my father contacted one of his friends who has experience with 3D printing and has one at home. His friend agreed to help and printed the case for us. Thankfully, the print was completed successfully, and I was able to move forward with my project.

Link of the 3D print case template: https://www.thingiverse.com/thing:4617605

Why did we chose this template?

We chose the **Raspberry Pi Camera Module case template** because it was specifically designed to accommodate both the Raspberry Pi 4 and the Camera Module 2, which are essential components of my project. This template offers a compact and secure design that ensures all components fit neatly together, providing durability and protection for the hardware.

Additionally, the template's design is well-documented, making it easy to download and adapt if needed. Its user-friendly features, like precise cutouts for the camera and buttons, align perfectly with the functionality I'm implementing in my portable camera project. These factors made it the ideal choice for housing the Raspberry Pi and ensuring the camera module's stability during extended use.

Video of the 3D printer printing the case:

https://drive.google.com/file/d/1Vd_aoE0e_0eQd6z9Tkke1UwKzg1Hr-f5/view?usp=sharing

Image of the Final Product:

Sunday, December 01 - Monday, December 02

Conclusion:

Unfortunately, we have so many things on our plate that we were not able to finish the web server as we would have liked, but we set a goal to finish it Monday night so we could present on Tuesday.

So here is what's left to finish our project:

1. Finish the Webserver

2 . Publish the pictures to the webserver

3 . Install the case

4 . Create a firewall

5. Create SSH keys

6. Create systemd processes

7. Finish the slides for the presentation

8. Finish the scripts

Expectations VS Reality

Finally, we have decided to proceed with two buttons: one to take a picture and the other to start and stop video recording. After careful consideration, we have decided not to implement additional features on the web server, such as filters or cropping for pictures and videos. While

these features would be valuable, they aren't crucial for our current prototype and would require additional time and resources.

We are also aiming to add portability to the system, but this depends on whether the portable battery we have works. If it doesn't, we will remove this functionality. If the battery doesn’t work, we plan to keep only the 3D-printed case and remove the harness.

As for the cooling component, we decided against including it. Since this is still a prototype, and given the constraints on our budget, the cooling system is not essential at this stage. We opted to allocate resources to other parts of the project that are more critical for its success.

Similarly, we are not adding a connectivity module. For non-portable use, we can connect the camera to the web server via an Ethernet cable. For portable use, we won't need to connect to the server at all, as we plan to upload images manually. The addition of a ToF (Time of Flight) sensor has also been removed from the scope of the project due to the increased complexity and cost.

We initially planned to have the server fully functional by week 3, but we have fallen behind schedule. We have not yet started testing media files on the web server, and we now realize that we should have allocated more time to ensure the server was up and running on time, while also factoring in potential setbacks.

Despite these delays, I am pleased with how my colleague and I have organized ourselves. Even with our other ongoing projects, we have made good progress. While we couldn't achieve every aspect of our original vision for the project, I am happy with what we've accomplished. Our camera now successfully captures photos and videos using tactile buttons, and we have a functioning web server. Given all the challenges we faced, I am satisfied with the results and hope we can continue to improve this prototype in the future.

Link GitHub repo:https://github.com/ayakharchafi/Unix-Project24