Engineering Addendum

Goals and Progress

The goal of the Smart System for Visually Impaired is to improve its users' abilities to navigate safely. The first way it achieves this goal is by providing users with more visual context about their surroundings through other mediums. Visual context is transformed into text descriptions in audio and semantic haptic feedback. A binary assessment of whether the current surroundings are navigationally safe or unsafe is quickly transmitted to the user through haptic feedback. This way, users can quickly make a decision to not proceed before knowing the full description of their surroundings which would take much longer to propagate than a simple buzzing signal. The system also illuminates the user's wrist in dark light conditions. Though this does not directly help the user navigate, it helps others navigate around the user. In its current state, the system achieves all of these goals, but its performance in terms of latency, accuracy, and over helpfulness can always be improved. Each aspect of the system comes with a few things to keep in mind while developing them.

Stereoscopic Setup

The stereo vision system is the backbone of this project. At times, getting both cameras to work simultaneously can be tricky. This task poses both software and hardware challenges. During development, cameras are often put in more vulnerable positions considering that housing may be improved or the cameras themselves are being swapped in and out. Because of this, it is important to always verify that the hardware is working, before trying to solve any potential software issues. Cameras can be verified individually, by plugging in one at a time and capturing an image. When connecting or removing devices from the Raspberry Pi, always make sure to power the Raspberry Pi off first. This prevents any potential damage to the devices. Once the cameras are verified to be working, it is also important to verify that the connection ports are working. These can be verified by plugging in a known working camera into an unverified port and taking a picture. Once the cameras and ports are verified, any issues with the stereoscopic setup can be attributed to software.

Another potential problem that may arise when working with the stereoscopic setup is the device numbers used to reference the cameras. On many setups like with a laptop, when two cameras are plugged in, they can be referenced through device numbers 0 and 1. However, in our experience with the Raspberry Pi, the device numbers were not always consecutive and in order, leading us to use reference numbers 0 and 2. These references are determined by the underlying operating system, so even if a different library is used to interact with the cameras, this referencing guide will remain relevant. An example of establishing the video capture references to the cameras can be found below.

```
    # Example of initializing video captures
    cap_left = cv2.VideoCapture(0)  # Adjust index if needed
    cap_right = cv2.VideoCapture(2)  # Adjust index if needed
```

Stereoscopic Calibration

When iterating on the camera selection or the mounting system, it is important to always calibrate the cameras and their stereoscopic setup. Optimizing the calibration error is pivotal for ensuring that the depth mapping output is accurate and useful. Our calibration scripts use a checkerboard pattern with 10 by 7 internal corners as the reference calibration image. Each of the squares of the checkerboard When using the calibration scripts, make sure to only settle for a calibration error below 1 pixel. Ideally, the mean error should be below .5 pixels. Any higher mean error will not yield a helpful result. To improve this calibration error, make sure to take the calibration images in well-lit conditions. It is important to also have the calibration reference image fully in frame and in multiple different angles across all of the captured calibration images.

Depth Mapping

Even after performing calibration, there are many parameters to tune that can affect the quality and usability of the depth map. The main things we modified when working with OpenCV's stereo block matching algorithm for depth mapping were the parameters for the number of disparities and the block size. Other features we modified were the speckle window size, speckle range, and texture threshold. The reason we modified these aspects was to find a balance between making the depth mapping sensitive enough for low texture objects without having a large, distracting amount of noise.

Working on the ESP32

The ESP32 deals with many hard coded constants. These range from WiFi credentials and the IP of the Raspberry Pi to the GPIO pins and parameters involved with sensors and output. When iterating on the ESP32 code, it is important to ensure that the constants used in the code reflect what is mapped out in the hardware. It is important to make use of the TinyPICO pin map for this. Ensuring that devices that require PWM or ADC are plugged into the appropriate ports is pivotal to the success of the project. Though the code handled by the ESP32 and the wrist wearable is not required to be concurrent, it would be best to avoid long sequential blocks of code that would prevent other functions from updating, when iterating on the functionality of the wrist wearable.

Prompt Engineering

Though it is good to save on tokens when using LLMs, we found that including more tokens in the prompt can be beneficial in many cases. A longer prompt that does not increase the requirements of the output can improve the specificity of the train of thought of the LLM and its concision. Often, the bulk of the latency when interacting with an LLM comes from the LLM producing its output. By reducing the length of the output by having specific constraints in the prompt, latency can be reduced. Especially since we are passing images to the LLM alongside the text, the extra tokens in the text portion of the prompt are well worth the savings on output tokens.