Final Project Report

Mason McGuire

6/06/2024

Motorcycle Wheelie Control

Background/intro:
My goal with this project was to implement wheelie control (a feature that some newer motorcycles have) onto my bike that was designed in the 90s. Below is the general overview of what parts the project includes and how it works:

The wheelie control system consists of a Raspberry Pi 5, IMU sensor, solid state relay, red and blue LED, red and blue button, LCD screen, and 3D printed mounting plate. The buttons/LEDs/LCD screen are used for user interface, the IMU sensor is used to read the angle, the solid state relay is used to limit engine power by cutting the signal to the spark plug, and the Raspberry Pi runs the whole operation. The system is always on when the bike is on.

Functionality:
When powered on, the system defaults to a mode (function in the code) “start_screen” that toggles through some text on the LCD screen. This text reads “System Disabled Angle: 20”, “Blue - angle Red - enable”, and “Current angle 2.83 degrees”. The angle on the first text block is referring to the current set wheelie angle, the angle on the third text block is referring to the current angle of the IMU. The second text block means that you press the blue button to set the desired angle, and the red button to enable the wheelie control.

If you press the blue button to set the angle, the system then goes to a mode (function in the code) “change_angle” that lets you change the desired angle. The LCD displays "Change angle: 20 Blue+ Red- HOLD*". This means press the blue button to increase the angle, red to decrease, or hold either button to select the current angle and go back to the start screen. 20 is just a placeholder for the current angle, it will be whatever you set it to.

If you press the red button to enable wheelie control, the system then goes to a mode (function in the code) “enable_system”. The LCD displays “System enabled Angle: 20”, where 20 is whatever the angle is set to. The blue LED will turn on to indicate the fact that wheelie control is enabled. If the IMU sensor detects an angle greater than the desired angle, it will disable the relay to cut the spark and turn on the red LED to indicate that spark is being cut. When the IMU detects an angle lesser than the desired angle it will enable the relay to enable the spark again, and turn off the red LED. You can press any button to go back to the start screen. When this happens, the blue LED will turn off indicating that the system is no longer enabled.

Software selection:
For this project I chose to use adafruit circuitpython/blinka. This was because they had a library for both the BNO085 IMU sensor, and the generic 16x2 LCD screen. I will admit this took a while for me to set up and install and use. This is probably because I am a beginner linux user. However, these libraries ended up working for my purposes (with the exception of an OSError, but I'll get into that in the challenges encountered section).

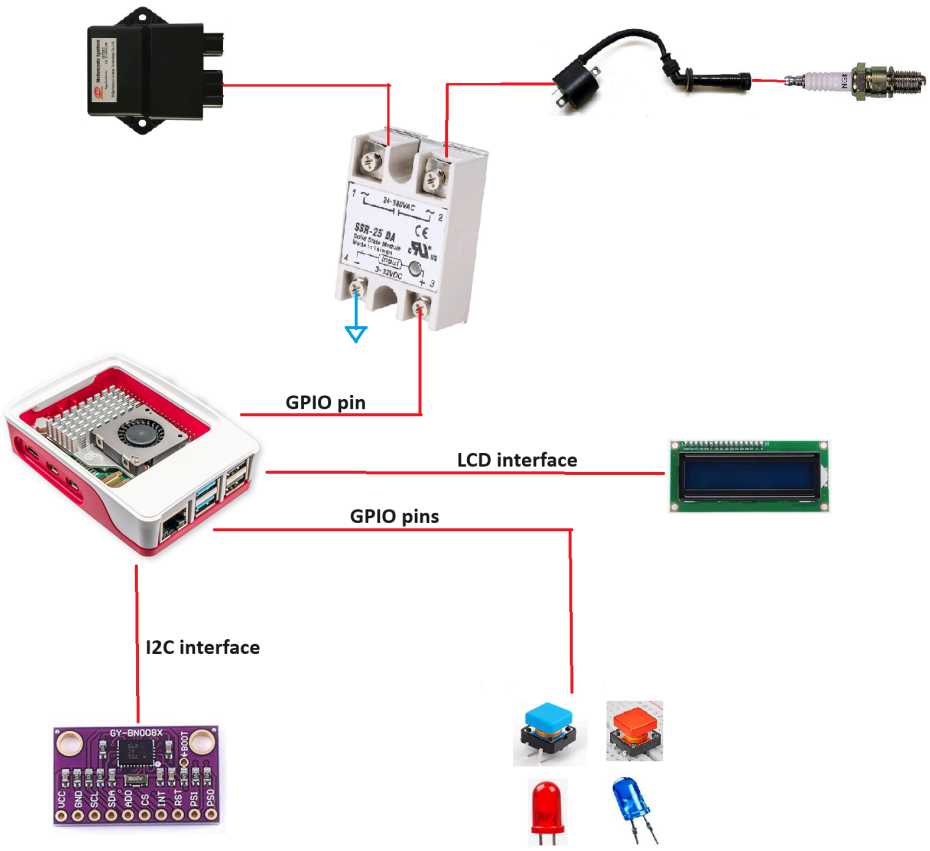
Component selection:
For this section I will not get into the selection of all of the components such as the LEDs, LCD screen, or buttons, as those were generic. I will focus on the components that enable the main functionality of the project such as the IMU sensor and relay.

Raspberry Pi 5 - I had this from CPE442 and decided to use it for its faster processing capability instead of the Pi 3 I also have.

SSR-25DA Relay - I chose this relay for a couple of reasons. First it is a solid state relay, this should mean fast switching capability (it actually wasn't fast enough, but I'll get into that in the challenges encountered section). It also has an input voltage of as low as 3 volts, so it should be compatible with the 3.3V output from the Pi GPIO pins. It has an output voltage of up to 380VAC which is necessary for it to cut the spark (I used a scope and measured that wire at up to 300VAC).

BNO085 IMU sensor - I chose this IMU sensor because of the integrated processor with great sensor fusion capabilities. This means that instead of having to do a bunch of math to get the angle from raw gyro data, I can just ping the sensor over I2C and get the angle. The calculated angle updates at a rate of 400Hz. While this seems great, it may also be a reason for failure, but I'll get into that in the challenges encountered section.

System architecture:
Below is an image of my system architecture. The pinout can be found in the github files under SCHEMATICS. The Pi talks to the IMU sensor over I2C, and interfaces with everything else using the GPIO pins.



Challenges encountered:
Multiple challenges were encountered during the design and implementation of this project. They will be individually addressed below.

3D printing/designing the mounting plate - To mount all of this hardware to a place on my bike that already had two screw holes proved to be a challenge because of the curvature of the bike at that point. It took six iterations to get this curvature correct. It took more than six 3d prints because I was having issues with bed adhesion. However, I was able to overcome these issues and design/print a mount plate that worked perfectly. The revisions can be seen in the CAD folder of the github files.

Frying the Pi - When I was connecting the GPIO extension board and ribbon cable from the Pi to the breadboard, I decided to test it to make sure I had the ribbon cable oriented correctly.

While testing, I accidentally shorted 5V to GND on the Pi and it went dark. Luckily it has a self repairing fuse that allowed it to work after sitting for a couple days, but this definitely put a damper on my progress.

IMU sensor delay - After getting the system all wired up and such, before testing on the bike, I wrote some code to output the IMU angle to a file, along with a timestamp. Then I moved the system rapidly back and forth, and then read the file. Even though the angle was technically written to the file at 400 Hz, there were large gaps in the angle, it would even jump a few whole degrees at a time. While I did move the system rapidly, there's no way I could move it a couple of degrees in 1/400th of a second. I think that the sensor fusion software running on the IMU must have a small delay in it so the angle data that it reads out at 400 Hz is not perfectly real-time. This was noticed when testing on the bike as well, as sometimes the spark cut would happen at a later angle than expected. To fix this, I would instead use the raw gyro data from the IMU, and do my own math to get the absolute angle data and the angular velocity. Then I would use these to try and predictively cut the spark a little early so that the angle is not exceeded. The raw gyro data can also be read at 1000 Hz.

Relay delay - When testing on the bike, I noticed that after cutting the spark at the desired angle, it would then take far too long for the spark to re-engage, fully killing the wheelie. I'm not quite sure if the relay delay is the issue, or the IMU sensor delay. However, I checked the relay max frequency specification (regretfully after already installing the relay in the bike), and its maximum switching frequency is 100 Hz. While this may still be enough, I would like to have a faster switching relay for peace of mind.

OSError when configuring GPIO pins via adafruit library - I had set up my program to run on the Pi on boot/reboot so that it would work when I plugged it into the motorcycle. When I tested this functionality when plugged into my monitor/keyboard/mouse, I would get an OSError about GPIO pins already being in use (this did not always occur when running the program with the Pi already booted up, but still did sometimes). It did not actually affect the program functionality because the program still ran, but I ended up spending a lot of time trying to fix it as it was annoying me. I tried adding program functionality to "clear" all of the GPIO pin settings on program start and exit, but that did not work. Researching the problem showed me that it was an issue with an adafruit library "libgpiod", and other people were also having it as well, specifically on Pi 5s. I ended up just ignoring the error.

Closing thoughts:
As you can see in the demo video the wheelies when the system was enabled (all wheelies except for the first one), the system did not re-engage the spark cutting relay fast enough. This was the main problem, but I still thoroughly enjoyed building and testing this project. I wish I had more time to work on it and perfect it, but I am starting a job soon after graduation and likely won't have the time.

Demo video:
https://youtu.be/qjaUx9cg5qc

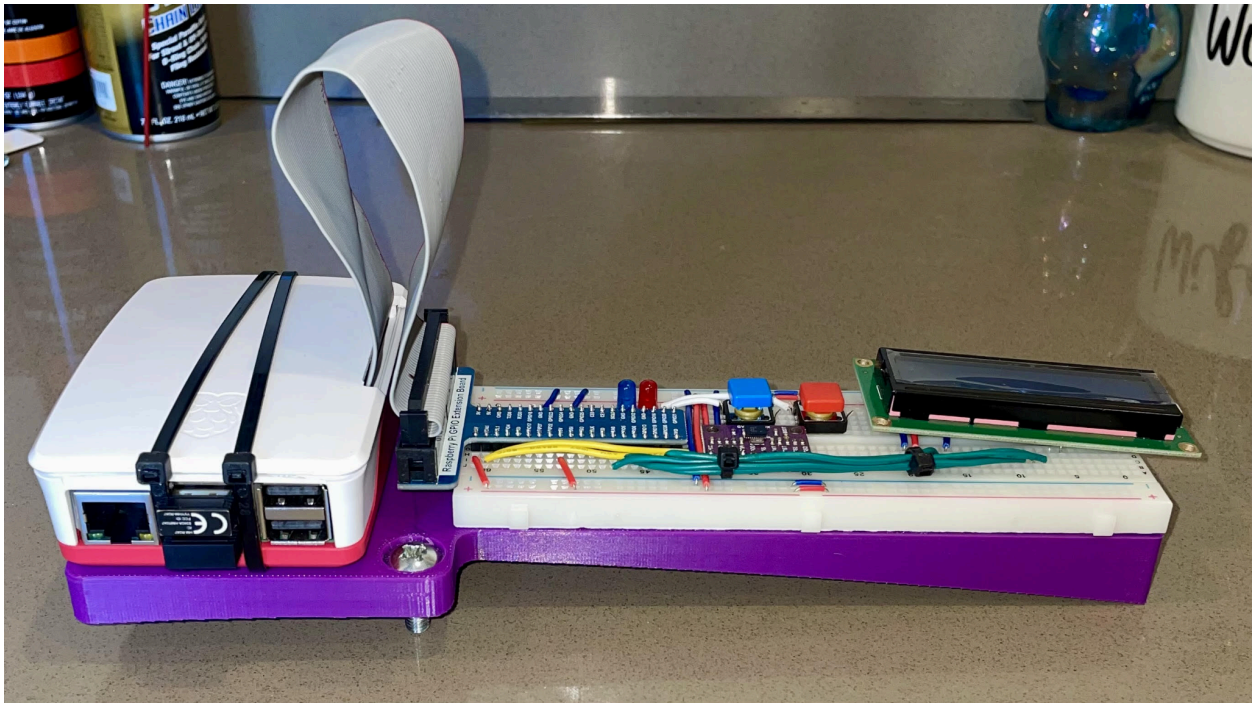
Code:
https://github.com/masonmcg/SquidSafe

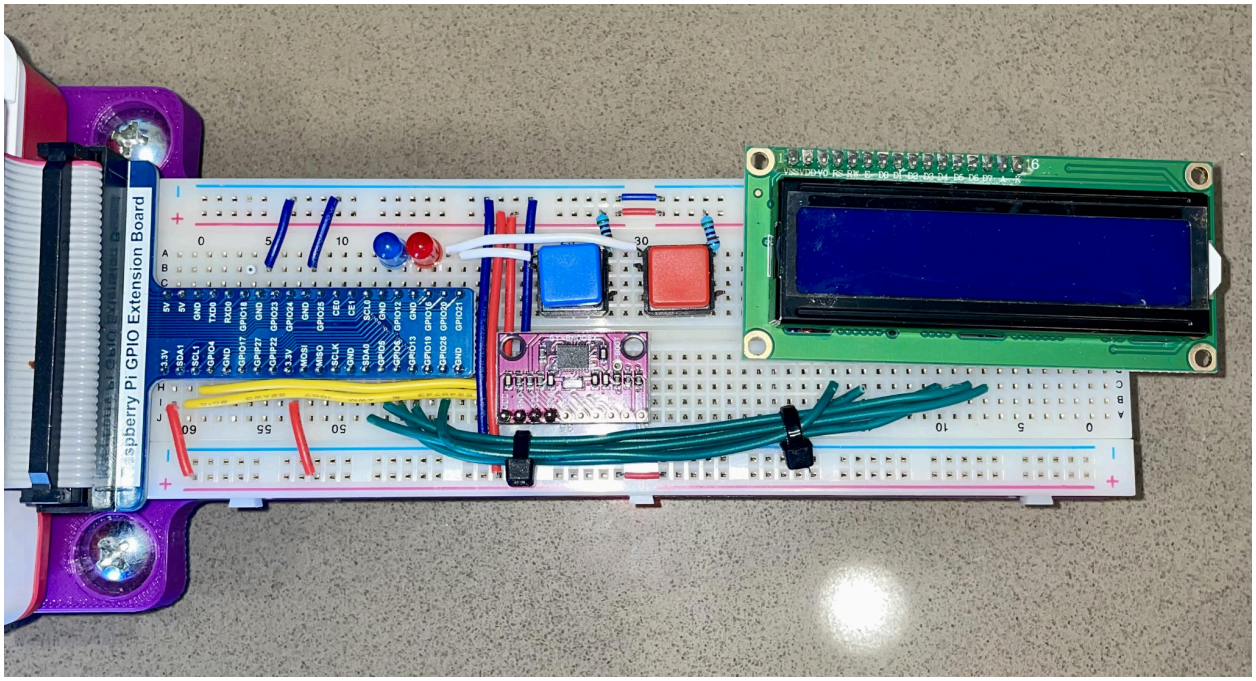
Pictures of system:

System placement on motorcycle

Relay placement on motorcycle

Note: the LCD screen in the pictures below is not connected and doesn't fit. It is just a placeholder to demonstrate where the LCD used to be. The actual LCD fell off of my bike this morning during testing (after the above pictures were taken).




Raspberry Pi GPIO Extension Board