Raspberry Pi Pullup Tracker

20 April 2017

Sam Dowd

sdowd1@lsu.edu

Table of Contents

Introduction

Motivation

I was motivated to choose this project for multiple reasons. My roommates and I got a pullup bar in our apartment over winter break. We started informally competing to see who could do the most. This sparked the idea that we could make the competition a little more formal, but this would be hard to do without the help of technology. That's where the second motivation comes in. The Raspberry Pi is something that has interested me since it was originally introduced in 2012. But I had never taken the initiative to buy one as I never had a project idea involving one which I thought I would have time to do. So the Raspberry Pi being offered as a special topic for this class gave me the perfect opportunity to solve two problems at once, both making a way to formalize the competition and giving me the final push to try out the Raspberry Pi.

Project Description

The project in its final form is an embedded system on the wall of my apartment consisting of six main parts. The Raspberry Pi itself, a breadboard to attach all of the components, a four button input pad, an LCD screen, a pair of USB speakers, and an infrared distance sensor mounted above the door. These components work together to provide an interface for performing pullups and keeping track of the competition. Users log in by keying their personal code on the keypad. The LCD screen then tells them to begin their set. While performing the set of pullups, the infrared sensor is read by the Raspberry Pi to determine the number of pullups done. The speakers indicate each time a pullup is counted with an audible signal so the user knows they are being tracked. At the end of the set a different audible signal is played and the LCD screen shows the user how many pullups they did, then updates them on their progress in the monthly competition. The Raspberry Pi then stores that information in a PostgreSQL database system. This database is then read by a Python script each night which updates a Django web app hosted on Heroku and accessible at http://pullups.mdowd.me/. The web app displays the progress of the users as well as some set records that are found using SQL queries to the database. Progress updates are also made using the speakers at the end of each night, where the Google Text To Speech API is used to read out the winner and loser of the day and of the current month. All of this is done automatically by the Raspberry Pi using cron jobs. The end result is a web app where users can track their progress from anywhere as well as a wall-mounted interface that tracks pullups and updates users.

Division of Roles

As discussed with Dr. Lee, this project was undertaken alone. But an honorable mention to my two roommates who provided massive amounts of data to work with as beta testers.

System Design

As my project was on the Raspberry Pi special topic, I did not choose any big data technologies or datasets. I will instead describe the design of my system in terms of hardware and software, as this project involved a good bit of both.

Hardware

The hardware component of the build is a system mounted to the wall outside of my room. The Raspberry Pi sits in the center of the system, pinned to the wall with thumb tacks. Attached to it via the GPIO pins is a breadboard. On that breadboard is a small LCD screen and a potentiometer to control the brightness of the LCD. To the other side of the Raspberry Pi is a four key keypad, also connected to the Pi via the GPIO pins. Power to the board runs from the MicroUSB port down to the outlet below. Running from the 3.5mm jack on the board is a pair of computer speakers that are conveniently, if dangerously, hung around the apartment's fire alarm. The final component is the infrared sensor mounted above the door right in the middle of the pullup bar, with three braided wires ran all the way down the door frame to connect to the Pi and breadboard. The typical use case is as follows: a user approaches the system. The LCD prompts the user, "Please enter code:" This is where the keypad comes into play. The user enters their predefined code (1, 2, or 3) followed by 4, which has been repurposed as an enter key. The LCD then prompts the user to begin their set.

The user grabs the pullup bar and begins performing pullups. The Raspberry Pi begins watching for input from the infrared sensor. When it senses the user pass the point where the sensor is mounted, a pullup is counted. The speakers play a quick noise to inform the user a pullup has been counted, the LCD screen shows the current pullup count, and the Raspberry Pi waits for the infrared sensor to read that the user has lowered below the sight line. The process of checking for a pullup then begins again and keeps going until the user stops for three seconds or presses any button on the keypad.

The LCD then displays the user's total for the day, total for the month, and maximum number of pullups done consecutively. This is a useful way to track progress, but does not give the user context for how their competitors are doing. So I have also implemented a function that displays the current progress of all users when a user keys the code 11 on the keypad. The other additional function is the ability to perform an untracked set by keying 32 on the keypad. This comes in handy for guests to the apartment who want to try the strange computer contraption on the wall. It was also integral to the testing and fine tuning of the system. My roommates wouldn't allow the results to be skewed by me testing things, so I needed a sandbox mode to be able to test adjustments to the timings and infrared calibration.

Software

The software for this project was written entirely in Python, as there were readily available libraries for the Pi and I already had a familiarity with the language. The software consists three scripts, a PostgreSQL database, and a web server.

The first script is the main script, main.py. I set up a cron job to run the script every time the Pi powered on. This is the script that sits in wait for the user to begin entering buttons on the keypad, then processes those inputs as well as the pullups. At the end of each set, it updates multiple relations in the database to record all the data possible. The main script consists of an infinite loop that checks the status of the GPIO pins where the keypad is attached forever. When one of the pins reads input, the main script listens for more keypresses, storing each of them, until the 4 key (repurposed as a return key) is pressed. At that point it checks the keys entered against codes it knows. If there is no match, it updates the LCD to display "Sorry! Unknown code." If there is a match it calls the associated function. Either displaying information pulled from the SQL database or getting ready for the user to do their pullups. If the code passed is a user code or the untracked set code, the script runs a while loop waiting for the user to flip a flag, which is flipped either when they have done pullups and then stopped or when they press another key. Inside that while loop, the script reads the input data from the infrared sensor to determine how many pullups are done and sends the appropriate signals to the LCD and speakers. When the set is complete, the script displays the progress of the user for a few seconds, before returning to the "Enter code please:" prompt once more. This continues indefinitely.

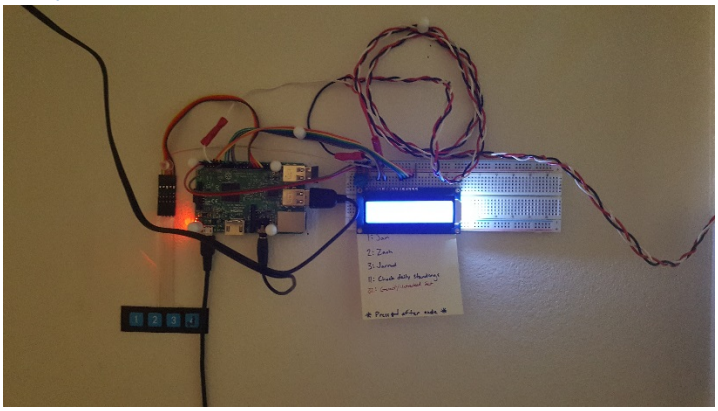
The second script is the midnight script, midnight_checkin.py. It also has a cron job, but instead of running at reboot, it runs at 11:55 pm every night. This script is more straightforward. It makes queries to the SQL database to determine a number of statistics about the progress of users. It then takes some of this data and feeds it to a data visualization function that saves it as a graph. Other data is simply fed directly to the web server to be displayed in text. Yet more of the data is used immediately by the script. Google's Text To Speech API is used to play a speech through the speakers indicating who the winners and losers of the day and all time are. The last thing the script does is take all the data and the graph and uses Heroku's git CLI to push everything to the Django server which triggers a server restart and updates everything on the website: http://pullups.mdowd.me/.

The final script is a modification of the midnight script, monthly_checkin.py. It is run via its own cron job, which runs on the first of every month at 9 am. It behaves very similarly to the daily checkin script except that it does not interact with the web server. It simply moves data around in the SQL database. It saves all the data from the month that just ended, makes comparisons with it to determine a winner and loser, announces that winner and loser using Google's Text To Speech API again, then resets everyone at 0 so the new month can begin. This completes the data tracking portion of the project. The data displaying portion is then done by the web server.

The web server is a substantial portion of the project. It is run on a Django backend hosted on a Heroku dyno. To make all the data collection useful, there needs to be an enjoyable way to access that information. The site itself is very simple but it is easy to navigate and find information. Stats displayed include the all-time champion, the champions for each month, the progress of the month graph, the highest maximum and most daily wins, and finally the individual stats for each user. All of this is updated automatically by the midnight script. Django is a View Model Controller system. The script saves all the information which is then read and passed into the template by the view. With such a simple site that just displays information, the majority of the Django framework is not taken advantage of, but it is an efficient way to display the changing information.

Detailed Description of Components

Raspberry Pi Model 3



The brain of the system is the Raspberry Pi. Every other component connects to this board either via the GPIO pins or its other ports. It is running the Raspbian Linux operating system, which is where the Python scripts are all run from.

4 Button Keypad

The first input device is the keypad. It has four buttons, labeled 1, 2, 3, and 4. The 4 button has been repurposed to work as a return key. There are five wires running from the keypad. The first is for power, and is connected to the power rail on the breadboard. The remaining four are signal wires, one for each button on the pad. These are attached to GPIO pins on the Raspberry Pi board so it can read the signals each time a user presses a button.

Infrared Sensor

The second input device is the IR sensor. It is mounted directly in the middle of the top of the door frame, where it is triggered only by users' heads moving above the pullup bar. The sensor consists of two components encased in one unit. An infrared emitter and an infrared receiver. The emitter is constantly emitting an infrared signal. When there is something within about 12 inches in front of the sensor, the infrared signal bounces off that object and hits the infrared receiver, sending a signal down the signal wire to the Pi. The IR sensor has three wires, the aforementioned signal wire, a power wire, and a ground wire. I braided these three wires together and tacked them along the door frame so that I could continue to use my doorway as a portal to my bedroom without getting tangled in wires.

16x2 Character Liquid Crystal Display

The first output device is the LCD. It provides the quickest way for users to obtain information and understand their interactions with the system. It can display characters in 32 spots split into two rows. It is limited to white letters, numbers, or symbols on a blue background, but that was enough to express the information needed. The LCD was difficult to wire because it actually has 16 pins. Power and ground for the back light, power and ground for the logic controller, and 12 different data pins for controlling everything. These pins are inserted into the breadboard behind the LCD's board, then routed to their various destinations.

Speakers

The second output device is a set of simple USB computer speakers. There was no good way to mount them on the wall, so they're hanging over the fire alarm. These have no complicated wiring because they simply attach to the Raspberry Pi's more recognizable ports, namely USB for power and the 3.5mm jack for audio transmission. The Raspberry Pi had to be configured inside the Raspbian operating system to use the speakers as the audio device. Once I had completed that, it was just a matter of using mpg123 to play mp3s.

Evaluation and Testing

Software Manual

See the Software section of the System Design section for a detailed overview of each script and software piece. For more in depth understanding, the code is commented thoroughly explaining each step and is attached along with this report.

Test Environment

This project was tested almost entirely in production. As soon as my roommates found out some part of the system had started working, they were eager to try it out and find all the ways it broke. As mentioned earlier, I programmed a special function that would mimic the primary function of counting pullups without actually modifying any of the data in the database. This helped with testing new code or different calibrations without skewing any data. Other than that, all testing was done by my roommates. In the first two weeks of having the system somewhat functional, the three of us combined performed over two thousand pullups. Along the way, many bugs were found and fixed. And many new ideas were spawned, considered, and sometimes implemented. Formal testing was done using the untracked function after any change to the algorithm, and testing continued informally constantly as my roommates brute tested everything by using it multiple times every day.

Test Cases

I used use case testing to test the system. Thus, most of the test cases are written in the form of use cases. If the use case is achievable, the test is considered passed.

- User enters their code. System responds by welcoming the user and waiting for pullups to be performed.

- User performs a pullup. The system responds with an audio indication and increases the pullup counter.
- User finishes a set of pullups. The system responds by displaying progress information to the user and saving information about the set in the database.
- Different user codes are inputted. The system responds by counting sets for the users associated with those codes.
- User enters 11 on the keypad. System responds by fetching the recent stats from the database and displaying them on the LCD.
- 11:55 pm passes. System responds by announcing via the speakers the winners and losers, updating the web app with all the new stats, and saving the daily totals to the database.
- 9:00 am on the first of a month passes. The system responds by storing the monthly totals and announcing via the speakers the winner and loser for the month.
- User navigates to http://pullups.mdowd.me. System responds by serving a page of statistics regarding the competition.

Conclusion

This project was a marked success. I integrated hardware and software components to create a system that provides a useful addition to the lives of myself and my roommates. I have never done any hardware projects before, so I learned a lot along the way. If I could expand the project further, I would like to host the web server on a second Raspberry Pi. This proved to be too difficult in the timeframe allowed because of the way our home network is set up. The eight test cases listed above all work flawlessly though, so the project reached a point I am happy with. My aim at the beginning was to create a system that is easy to use and tracks pullups consistently. I hoped to also include a convenient way to access all the information recorded. I achieved both of these goals. My one regret is letting Jarrod win March. It's hard getting called out by your own creation as a loser.