State University of New York at New Paltz

“PiCam”

Raymond Borriello, Tyler Nevone & Anthony Rivera

CPS 342 - Embedded Linux

Spring 2017

1. Project Description

We were tasked with developing a mechanism, using the Raspberry Pi 3, that can take photos in any direction on the x and y axis (i.e. up, down, left, & right). We also had the responsibility of allowing the user to control this movement via a web interface. While on the web interface the user is able to enter a desired motion for the camera (i.e. up, down, left, right) and with that command our camera will adjust its position accordingly. If the user clicks a button to take a picture, the camera takes a picture and stores it in our application directory where it can be viewed on the home page. To start this project we were given two continuous servo motors for camera motion, a 16-channel servo driver to regulate power sent to the servo motors, and a Raspberry Pi camera. Additionally, we required a small breadboard in order to interface the Pi with the servo driver, a 4xAA battery pack to provide power to the servo motors, a soldering iron for the driver heads, and some various craft supplies (cardboard, glue, etc.) for the base.

2. Project Goals

We had several mini goals in this project, all leading up to the overall goal which was a fully functional, movable camera. Our first goal was to get the camera to take pictures via commands from the Pi. The next step was to figure out how the servos worked (first individually, then in unison) and how to coordinate them through the servo controller. We also needed to set up the web interface with buttons so that they can control both our servos and the camera's capture feature. And lastly we were implicitly tasked to think of a clever way to put everything together so that our mechanism can function properly (i.e. what to use as the base for the servos, how to connect the servos to each other and how to connect the camera to that).

3. Project Implementation (Hardware & Software)

3.1: Camera + Boom

Our base was constructed using household craft supplies, namely cardboard, glue, and a shaved-down pencil. The lateral (left-right) motor is glued to the cardboard base, oriented so that anything attached to the motor moves right when the motor rotates clockwise and left when the motor rotates counterclockwise. The longitudinal (up-down) motor is then glued to the lateral motor, so that when the lateral motor rotates the longitudinal motor rotates left or right with it. The longitudinal motor is oriented such that a) the motor head to which the camera is attached overhangs the lateral motor below it, allowing the camera to move freely and b) anything attached to the motor will move down when the motor rotates clockwise and up when the motor rotates counterclockwise. Attached to the longitudinal motor is a pencil, shaved down so that the motor can rotate without it hitting the base. Using a rubber band, we were able to securely (enough) fasten the camera to the pencil to allow for camera movement and clear pictures.

3.2: Web Interface (PiCam.html)

The web interface that we implemented was fairly simple, using just HTML. At the top of the page is displayed the most recent photo taken with our PiCam (if an image exists). If one does not exist, the area is left blank. Directly below the image is a button labeled “Take Picture,” which takes a picture and stores it in our application's “Pictures” directory. Below the image are four buttons labeled with their corresponding directions: “Up,” “Down,” “Left,” and “Right.” By making each button a separate HTML <form> element, when clicked, each button submits an HTTP GET request to our web server to move the camera boom in the chosen direction. The GET request consists of a string indicating which directional button was clicked (“up,” “down,” “left,” “right”). As the actual moving of the servo motors and taking of pictures is handled in our web server, you can find a description of their implementation in 3.3: Web Server.

3.3: Web Server (app.py)

Python already has a GPIO module, called RPIO, that allows for a Raspberry Pi to send commands to peripheral devices through its GPIO pins (see 3.1: Camera + Boom). Specifically, RPIO provides us with RPIO.PWM, which allows for pulse width modulation – a necessity for making the servo motors move. For this reason, we chose to use Python's Flask in our web server; however, theoretically there's no reason why this project could not be replicated using something other than Flask. Our server is hosted on the SUNY New Paltz Wyvern server as provided by the school.

Upon startup, our application initializes both servo motors and sets their pulse width frequencies to the recommended values (lines 9-12). It has its home route (‘/’), which renders a Flask template to display the homepage (lines 17-21). The application also has a route for each direction of movement: ‘/up,’ ‘/down,’, ‘/left,’, ‘/right’ (lines 23-55). As mentioned in 3.1, we oriented the motors so that clockwise rotation of each corresponds to down (longitudinal motor) and right (lateral motor), while counterclockwise rotation corresponds to up (longitudinal) and left (lateral).

From our research and a little experimentation, we determined that a pulse width of 120 milliseconds rotates the motor counterclockwise (i.e., up or left). The pulse is allowed to travel to the motor for 0.02 seconds, enough time for the motor to rotate a sufficient enough distance for the project's purpose. Once 0.02 seconds has elapsed, the pulse is stopped and the motor ceases rotating. To rotate the motors clockwise (i.e., down or right) we used similar reasoning and experimentation to find the correct values. We determined that a pulse width of 475 milliseconds (ms) rotates the servo motors clockwise (i.e., down or right). However, the motor strength is noticeably weaker in the clockwise direction. To compensate, we allow the pulse to travel to the motor for a longer duration, 0.1 seconds, so that the magnitude of one unit (in this case, the amount of rotation resulting from one button click) of counterclockwise rotation is as close to equal as that of one unit of clockwise rotation.

Lastly, we have our '/takePicture' route (lines 57-68). When this route is called, the application first uses Python's 'os' library to check if the 'static' directory exists. If the directory does not exist, then using Python's shutil library the 'static' directory is created. Then the application simply sends a command to the Pi to take a picture and stores the pictures in the application's 'static' directory.

4. Project Timeline

For the project we divided into different tasks what we needed to get done to help organize and prepare us before we started working on the project itself. Our group spent a week learning the commands for each of the different components (servos, camera, web server). Once we began to code, having all this information ready helped us immensely. We first had to ensure that the hardware worked before we went any further. We soldered the servo controller to the breadboard (referring to the Adafruit guide for pin positions) and wrote some test programs to move first one, then both servos, as well as take pictures with the camera.

Once we achieved an understanding of how our hardware worked and were familiar with it we started to work on the software side. At this point where to proceed first was divided among the group. We ultimately decided on tackling the web server and template first. After we had everything running we started working on the second half of the software side, the program that essentially links to servos and camera to the web server. We faced some issues with linking our app.py program to the web server, such as the web server not sending commands to app.py. We found the right call commands and some typos in our code once implemented we got it starting taking pictures and moving servos.

We had two separate problems with the camera component. First, during our initial testing we found that the picture was being taken and stored in our desired location, but would not display on our web interface. It turns out that a Flask application serves images from a specific directory within the application, the ‘static’ directory. So we solved this problem by simply changing the location where our application was saving the image to a location within the ‘static’ directory. Second, we had a browser problem that we simply ran out of time to solve within our application. When our Flask application is started and the user navigates to the home screen, the browser caches the image displayed and continues to show that image, even after a new picture is taken and the page is refreshed. This can be solved by performing a ‘hard refresh’ (CTRL-SHIFT-R) on the application home page, which clears the browser cache and reloads the stored image. Once that was sorted out we ran it through some final tests, then glued our servos together and created our base.

5. Project Difficulties

While working on the project we faced many situations that were at first difficult to overcome, but we were able to find a solution. Due to everyone in the group having a different viewpoint on how to approach the project we were able to find the easiest methods to get the web server, servo control and the hardware itself to work. We faced challenges due to our group’s lack of familiarity with Flask, which ultimately costed us much of our work time when we tried to upload the image onto the web server. While we were able to achieve what we were assigned our group had some goals that we were unable to get into our final code. We were unable to get a coordinate system into the program for when the camera goes up beyond 90 degrees, counterclockwise rotation becomes down instead of up; and a button to reset the camera to a neutral position. The first could’ve been done if we had been able to spend less time debugging our code. Bringing the servos back into a starting position is an issue due to the servos not being able to move the same distance for every turn. The coordinates would not be able to return to an exact, natural zero position.

6. Group Member Contributions

Raymond Borriello: Flask application config/initial setup - motor initialization, defined routes (but did not write code the actual functions in the body of each route); set up hosting on Wyvern; constructed web interface template.

Tyler Nevone: Servo and camera control wiring; soldering; hardware construction.

Anthony Rivera: Flask application commands to control motors and take pictures (i.e., functions within the routes)

All: LOTS of Flask troubleshooting; early project research (see 4. Project Timeline); base design; experimentation to determine necessary pulse widths for servo rotation.

7. References

Reed, F. (Date unknown). How Do Servo Motors Work. *How It Works*. Retrieved from http://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html

Townsend, K. (2012). Adafruit 16 Channel Servo Driver with Raspberry Pi. Retrieved from https://learn.adafruit.com/adafruit-16-channel-servo-driver-with-raspberry-pi/overview

Flask documentation: http://flask.pocoo.org/docs/0.12/

Python RPIO documentation: https://pythonhosted.org/RPIO/