Camera Boom Project

By Jonathan Greenberg and David Furfaro

David and I were assigned the Camera Boom Project. The overall goal of the project was to create a mechanism that takes still pictures and displays them on a web interface. The mechanism also had to be able to turn on a single plane to the left and right using servos. We broke the project into two main components: Hardware implementation and Software implementation. After some initial discussions about the project, we decided that David would do the software and I would do the hardware. Neither of us had much experience in web programming or hardware, so the project proved to be a big challenge for us.

Jonathan broke the hardware portion of the project into two parts: Physically building the hardware, and implementing code for the hardware. In addition to the Raspberry Pi and the Camera Module we already had, Jonathan purchased some additional hardware in order to build the rotating mechanism. Jonathan bought a Pan/Tilt kit from *mindsensors.com* which included two micro servos, a servo controller board, plastic mounts to build the Pan/Tilt bracket, and some screws. Using the Pan/Tilt kit proved to be a smart purchase as it prevented us from having to tediously wire everything together by hand, saving us countless hours. After completing the design, we had to figure out how to write code to get the servos to rotate the bracket. After doing some research

about servos and how they work, Jonathan found a driver called Servoblaster. Servoblaster, created by Richard Hirst, allows a user to rotate a servo through any GPIO pin on the Pi. To give a brief summary of servos, servos are controlled through a series of electrical pulses that require precise timing in order to work. Each pulse has a specific width that determines how far a servo should rotate. Since each servo is different, servos have different ranges of rotation limits. Without using the Servoblaster driver, users would have to figure out the precise timing and the width of the pulses in order to make the servo rotate. Servoblaster lets users rotate a servo by specifying the PIN # the servo wire is connected to and the position in degrees they want the servo to rotate, without the need for timing or pulse widths. For example, if you want to move a servo that's connected to PIN #5 200 degrees, you would say: *echo 5 = 200 > /dev/servoblaster.*

The project was definitely a learning experience, but also a tedious experience. As mentioned before, neither of us had much hardware or web programming experience so countless hours of research was required. No aspect of the project was "easy", but no aspects were "impossible" either given enough research and practice. Though we were successful in our project, we would have liked to add/edit a couple of things, the main thing being the web interface. Unfortunately, we had to make several buttons that rotated the servos to specific positions instead of just having one button for left and one for right that rotated the servo a specific interval. This was most likely due to

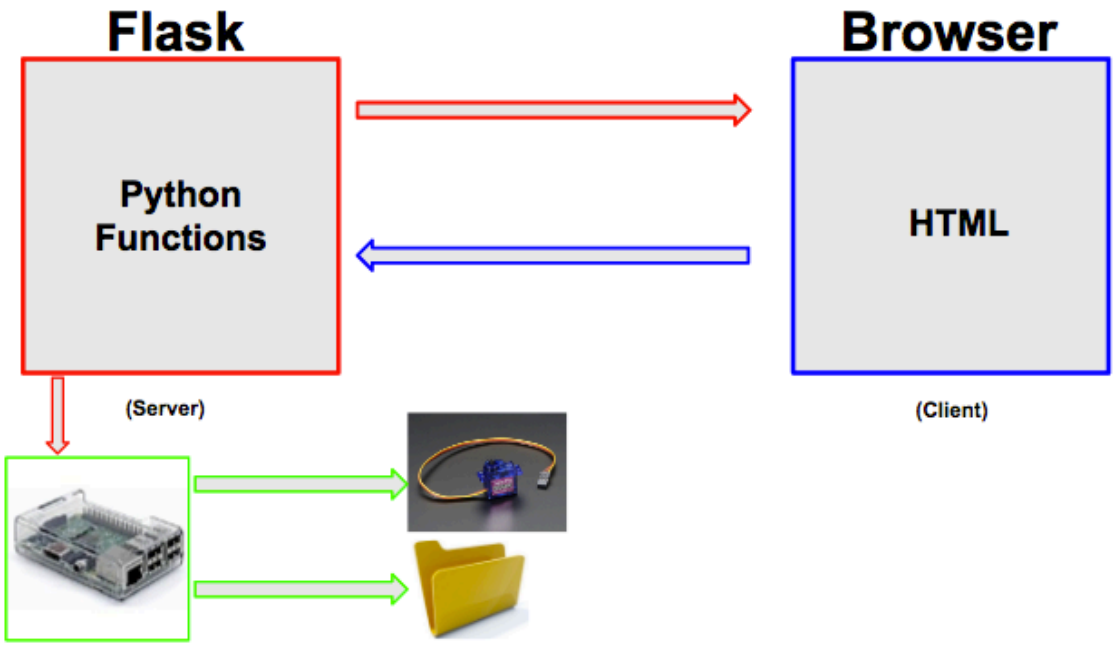
inexperience on our parts. Neither of us could figure out how to store a variable inside the Flask application. This capability would have allowed us to apply logic to the moving the camera that only needed two buttons, “Left” and “Right”. The obstacle of no stored variables also compromised our “Take Picture” feature. When implemented it takes a picture and stores it in the directory provided in the code. However, because we were not able to save variables we could not alter that directory/file-name and as a result every time a picture is taken, it overwrites the previously stored image. Our last issue with our software is the inability to reboot the stream after taking a picture. We chose to share what the camera was looking at by using Mjpg-Streamer. The camera however can only host one client at a time. As a result, Mjpg must be shutdown before Raspistill can take the picture. We planned to counteract this problem by providing two buttons “Prep Stream” and “Restart Stream” to reestablish the Mjpg stream after taking a picture. Unfortunately we could not fix an issue in which the server continuously loads after reestablishing the stream. We would have also liked to include a tilt function along with the pan function. This would have allowed us to move the servo up and down along with the left and right pan function. We didn’t include tilting because we were afraid of running into the same issue with buttons that we had with panning. If we included tilting, there would have been around 20 buttons on the web interface which we felt was just too overwhelming not only for us, but for potential users as well. In the end, we managed to achieve what was assigned to us, nothing more, nothing less, which was still a major success to us given our experience.

Software Implementation

David used two languages In order to implement the Web Interface and allow it to control the Raspberry Pi. The server was implemented using Python, while the client side of the Web Interface was designed using HTML.

Below is an illustration of the software implementation:

Implementation Diagram

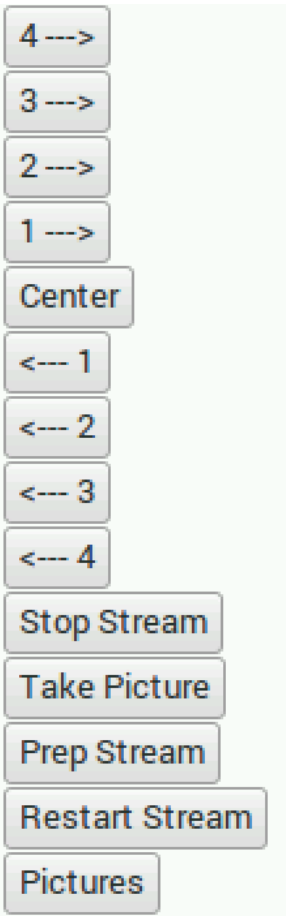


As shown above, the client (which is your browser) runs HTML code and makes an HTTP request to the server. The blue arrow illustrates this. The Flask

framework allows the server to accept the request and perform its respective function in the python script “testcam.py”. The script function then sends instruction to the Raspberry Pi, illustrated by the green arrow. Once those instructions are executed and the python function is complete, the Flask framework allows the server to send a HTTP response back to the client, the red arrow.

Examples of Interactions Between Client and Server:

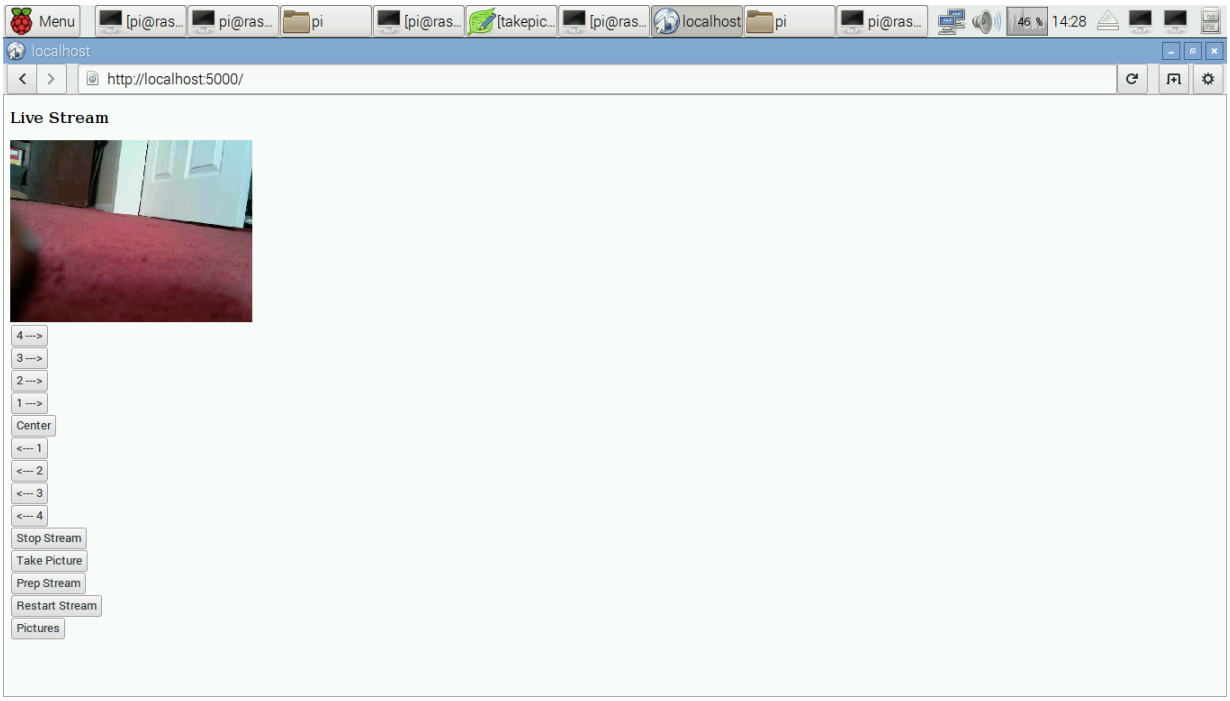
Web Interface Buttons



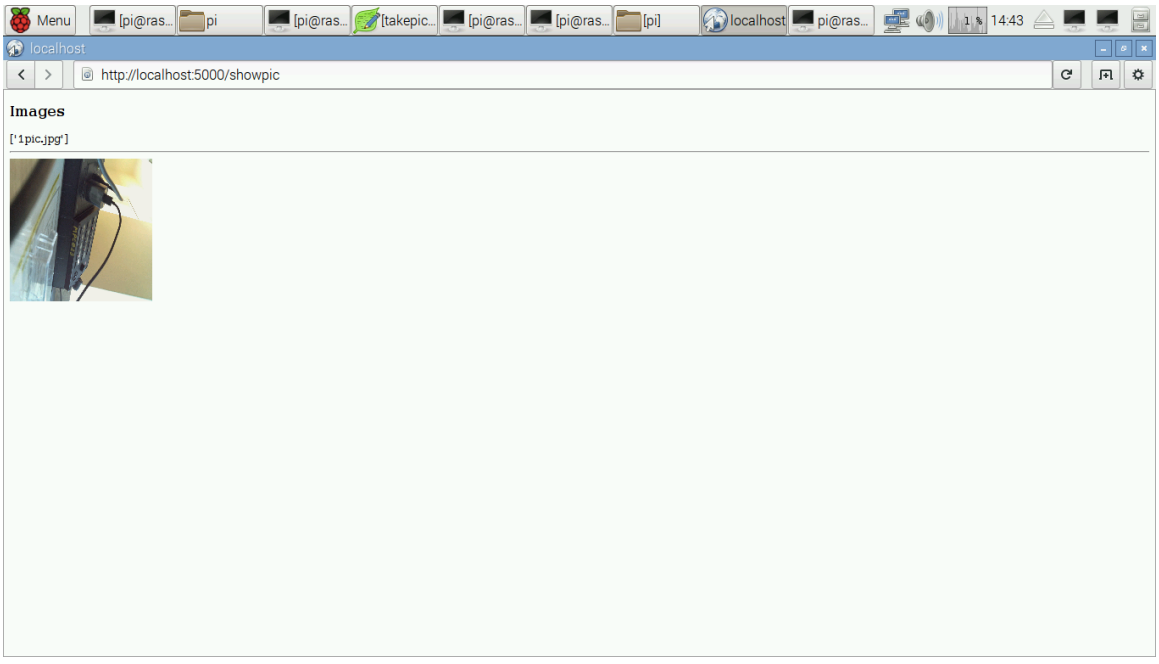
When the Web Interface first loads you are presented with an HTML template containing these buttons. Each button allows for a request from the client to the

server. One example of this would be if the user clicked on the “Center” button. When clicked on, the <form> element in the HTML file “takepic.html” , sends a request to the Flask server. Flask then interprets the request and executes the correct python function within the script. In this case the “CenterCam” function. This python function sends instructions to the Raspberry Pi. For this example it executes os.system("echo 5=120 > /dev/servoblaster"). The “os” function takes in the give string and runs it as a command directly on the Raspberry Pi. This specific command accesses ServoBlaster, which controls the Micro Servo in which the Camera Module is mounted. The command is telling ServoBlaster to position the camera at 120° on it’s plane. Since the micro servo can be positioned from 80° to 160°, David considered this degree to be as centered for the camera. Once the python function is complete, using “return render_template” the Flask server sends a response back to the client to load the template “takepic.html”.

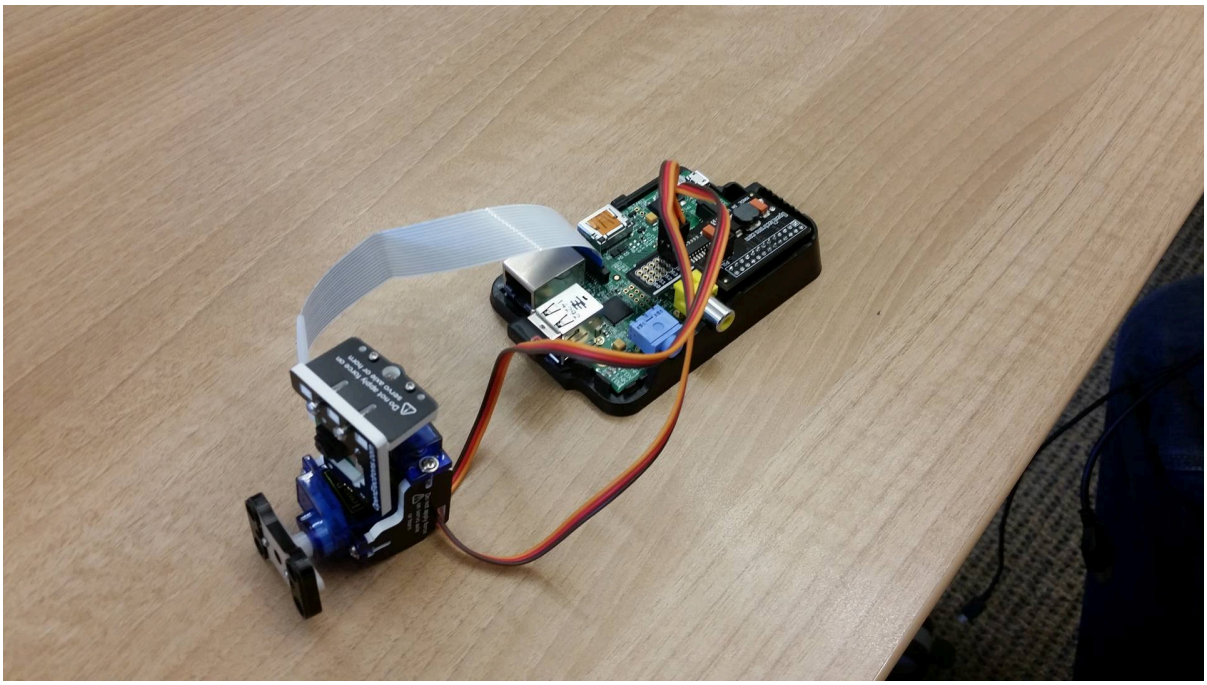
Final Web Interface



(Main page that provides stream and action buttons)



(“Pictures” button displays photos inside directory specified in TakePic function)



(Picture of Raspberry with Camera Module and Micro Servos)

Group Members and Roles:

Jonathan Greenberg - Jonathan researched and constructed all the hardware needed for this project. He also researched and implemented the library (ServoBlaster) needed to control the micro servos

David Furfaro – David researched, designed and implemented all the software needed for this project. He wrote the python script as well as all the HTML templates.

List of Reference and Resources:

Install Mjpg-Streamer:

http://blog.cudmore.io/post/2015/03/15/Installing-mjpg-streamer-on-a-raspberry-pi/

Install ServoBlaster:

https://www.raspberrypi.org/forums/viewtopic.php?f=44&t=54067

Creating HTML Forms:

http://www.w3schools.com/html/html_forms.asp

Learning Flask: **http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world**