**Embedded Linux – Project Documentation**

Project Description

"Develop a mechanism that can take still photos looking in any possible direction, controlled by a web interface. You must be able to position the camera at the end of a boom, aim it in any direction relative to some fixed reference frame, take a picture and send it back to some connected location. Develop a web interface to display the images; you will have to think about how you will construct a frame of reference relative to which the camera angles can be specified."

Project Summary

The Camera and Web interface were the easy aspects of the project so far. The Camera was easy to work with and our group knew web programming well enough to program a web interface fairly quickly and efficiently. The camera, which is mounted on a boom, is accessed through a web interface. The web interface provides a reliable video stream, and uses two sliders representing the x & y axis, which control servo movement; the interface displays the current camera coordinates dynamically. Picture button allows users to snap screenshots of the current camera stream and saves it into a gallery database. Finally, when the user unloads the webpage, a notification will be sent to the server and the servos will reset to their origin point.

Project Components

**Motor Movements and Commands** – Previously received commands for the continuous servo from the interface and worked with the 16 channel servo driver to make the servo rotate. This was found to be too bulky, and inefficient as it had too many components and made the setup of the project too complicated. The decision made in the end was to simplify the setup, so instead of using the 16 channel servo driver, the servos were connected directly to a power source by soldering header pins to a spliced micro usb cable that connected directly to the servos and had 2 jumper cables connecting the servo I/O ports to the RPIO pins (pins 22 & 23) on the raspberry pi.

**Web Interface -** Website displays video stream of the current camera coordinates, as well as displaying these coordinates on the page. The coordinates are adjusted by sliders provided by the web interface, of which include an x & y axis; the x axis may move a full 360 degrees, and the y axis may move up to 180 degrees. Slider coordinates determine the servo direction, and the y axis is restricted to 180 degrees of movement so the camera never looks down at the boom. When a slider is adjusted (in increments of 20 degrees), the new & previous values of the servo coordinates are sent to the web server for processing. Both of the servos will be reset to the origin point when the user exits the website. Camera button takes a photo of the current stream and sends it off to the webserver to save into a gallery database. Users may press a link on the website to view a gallery of their pictures.

**Web Server (Flask) -** The web server is runs Flask, and handles all web interface commands. When the web interface sliders are adjusted, an AJAX call is made to update the position of the servo motors--in turn, adjusting the stream view. These commands are processed in the server by using the axis, direction, and degree of movement provided by the web-interface. When the user takes a picture, a post request is made to the server and the server will provide a standardized name (if none is provided by the user), finally storing it in a database of images.

**Camera Functions** - Handles streaming and takes images.

**Camera Boom** - Used Google SketchUp and a Ruby Plugin that allows SketchUp for export a STL file. The boom was printed by the MakerBot Innovation Center at New Paltz. The boom’s function was to hold the motors and give a base for the pi and wires. The overhang is used for a mounting for the motors and the pi is mounted below on the base.

Lessons Learned

Most of the information that was used to understand the working of the servo motor came from the servos data sheet. Here is the link to the servos data sheet: <https://www.adafruit.com/datasheets/154datasheet.pdf>. The initial method that was going to be used to connect and control the servos with the raspberry pi was by using Adafruit's 16-channel 12-bit PWM/Servo Driver. We found a tutorial on how to use the Adafruit 16 Channel Servo Driver with Raspberry Pi directly from the Adafruit website. Here is the link to the tutorial:

<https://learn.adafruit.com/adafruit-16-channel-servo-driver-with-raspberry-pi>.

For the boom, We learned that 3D printing has a lot more prerequisites that need to be followed. Since the printing is done with liquid plastic, overhangs and sideways outlets cannot be supported during the printing process. Also, Google SketchUp doesn't feature software that exports the STL file that is used by the Makerbot Innovation Center that is here on Campus. I downloaded a Ruby Plugin, found here <https://extensions.sketchup.com/en/content/sketchup-stl>

which allows the user to export a STL file of the model. If we were to remake the boom today, we would make separate pieces that fit together so they could be printed in a way so extra supports did not have to be added and in a way that would make it look and function better.

Originally, The servo motors had been programmed to move using the GPIO library, which used duty cycles percentages and hertz; this would have worked perfectly fine if the servers had not been continuous. However since the servos we chose *were* continuous, the direction that they move in is determined by the specific time between subcycles of each PWM pulse. A large subcycle time would move the servo clockwise, and a smaller subcycle time would move it counter-clockwise. Since the GPIO library worked with duty cycle percentages, it was very difficult to find a frequency & duty cycle that would provide the desired sweep speed & degree of precision. This was solved by switching to the RPIO.pwm python library. This library allowed the program to output subcycles at specific millisecond intervals, as the servo motors needed. After some programming, the servos were able to move much smoother & with very precise movement.

Although this may not be a very important lesson learned, while creating the web interface & server it would have saved a lot time if I had a second monitor. Switching between the pi display and my desktop always took a solid 4 or 5 seconds, which added up over the many of times I had done this while setting up the site. Aside from this, there was definitely a lot learned in respect to how jQuery interacts with AngularJS. More specifically, I learned how to properly use AngularJS’s ‘scope’ variable, and how to access AngularJS functions from jQuery using this variable.

Group Members and contributions

**Ferrao, Lionel** -

· Worked on servo motor, getting it to run with GPIO.

-The first objective was to get movement from the servos and multiple tests were done in order to continue with the project's goals.

· Hardwire and solder the spliced micro USB cable to power the servos.

-Initially we tried using Adafruit's 16-channel 12-bit PWM/Servo Driver to control the servos, but we faces an issue. The first issue was not being able to power up the servos. The servos were not powering up and that made testing of the servo movement to be quite a challenge. So, instead of using the servo controller and a breadboard, we decided to splice a micro USB cable and solder header pins to the red and black wire for a Vcc and Ground connection respectively. The header pins could connect directly to the servos from a power source. The micro USB cable used for this project was provided by Prof. Easwaran.

· Used jumper cables for the header pins in order to connect the I/O port of servo to the raspberry pi.

-We utilized header pins that connected to the I/O ports of the servos and then connected the other end to pins 22 and 23 of the raspberry pi using jumper cables.

· Utilized heat shrink tubes in order to cover up the solder and make the connections look neat.

-By using the spliced micro USB cable, we were able to reduce the amount of clutter that came from using many wires for the project. By having a direct connection from power supply to servo and servo to raspberry pi we were able to simplify the look of the project and made it look more neat and tidy. I used heat shrink tubes to cover up the soldering done on the spliced cable, insulate it and provide abrasion resistance and environmental protection to it.

**McCabe, Michael** -

· Configured camera and wrote python script for streaming and saving pictures with directional commands.

Wrote and Designed Midpoint Presentation

· Added scripts to the Flask framework that handled servo movement

-This was completed by reading in positional data sent by the web interface and adjusting accordingly. The web interface sent new and old positional coordinates to the web server, which would then be retrieved inside my python script. The script would subtract the new & old values, making the degree of rotation this value, and the sign of this value the direction of movement. The script would then move down through a sequence of movements, until the remaining degree of movement left is 0--this was handled by moving through a sequence of while loops, which provided movement degrees of 180, 100, 80, 60, 40, and 20.

**Putterman, Daniel** -

· Researched code to enable servo motor movement.

* I found some basic code that I modified and made my own in order to get the servos moving. The code could move the servos in 360 degrees, the user could tell the servo where to stop, so that we could take the picture, and for how long to hold that position, I did this so that the camera could take a properly focused picture, the camera on the pi is very slow so I set it to hold for five seconds before it moved again.

· Communicated between raspberry pi and motors

* I used the general purpose input/output (GPIO) pins from the raspberry pi to send the data to the motors, there were a few data sending techniques that the raspberry pi could handle that I learned from the Embedded Systems class with professor Otis that I took in the previous semester, such as i2c and SPI, but using simple GPIO commands was by far the fastest and easiest way to send data.

· Powered the motors

* Once I plugged the servo motors in and ran the program I created nothing happened, I discussed it with the group to try and figure out why it didn’t do anything until I had the idea to go the vendor's website and found out that the motor needs more power than the raspberry pi can provide by itself. We then asked the professor for a battery pack or something similar to what the guide on adafruit had. We received a 5 volt wall to usb device that would be compatible with our pi after a little tinkering.

· Spliced,and wired equipment

* We spliced the micro USB cable that we received from the professor, used the red wire as VCC and black as ground for the motor, we hard wired and soldered it to power the servos. After this was done I tested the program that I initially came up with and saw that the servos were rotating, moving, stopping, and behaving as expected. At this point I began collaborating with the people that were tasked getting the website running and pushing commands to the servos via the website to rotate our servos which in turn rotated the camera. This helped the group better understand how the servos work so that we could use the basic code to start to develop over-the- web servo commands over the next few weeks.

· Soldered to keep things in place

* Once we were ready to start putting all the pieces of the project together Lionel and I soldered the header pins in order to connect the I/O port of the servo to the raspberry pi so that it would stay in place so when the servos were moving nothing would accidentally disconnect.

· Utilized heat shrink tubes in order to cover up the solder

* This was another step that we took in order to keep all of the connections in place and to make a clean, neat look and presentation that would be aesthetically pleasing.

**Inglese, Christopher** -

* Designed web interface - The aspects of the web interface I had done included: layout, image serving, image taking/saving, gallery display, and searching/sorting the gallery.
  + Layout - I used bootstrap to help make the site mobile friendly; although it may not work on some smaller phones, it should adjust properly for tablets and larger phones. This was a relatively easy task, and implemented bootstrap’s column and row methods for making the site adjust properly with window size. The site also used bootstrap for creating a more visually appealing look.
  + Image serving - For this I used some AJAX calls. When the user clicks on a gallery image javascript will detect these events and send SQL search data to the server, the server will then create a search query, and sort through the image database looking for an image with the same details. When one is found, it will send back to the web interface image details, such as the description, title, and the image source. A modal is updated with these details, and when the image is finished loading, the modal will show itself to the user.
  + Image taking/saving - I used the Pygame library to help implement these features. The library allows the script to easily access a webcam or recording device, and take/save pictures. The image is served by sending an AJAX request to the server, which alerts the server to use the Pygame library to take & save a picture; this data is sent back to the web page. When the user clicks the save button it will send SQL query data to insert the image info into the SQL database. Otherwise the image will still be saved, but not stored in the database--so it will not be publicly available or displayed on the webpage.
  + Gallery - The gallery display uses a bit of the AngularJS javascript framework. Whenever the page is loaded, or an image is added to the database, the gallary will be reloaded. This is completed by using the AngularJS ‘for each’ function, which will create a new segment of HTML code for each element in a list. In the context of this code, for each image returned from the server, it will create a div displaying the image. This way the gallery will always be able to load every new image from the database.
  + Image searching/sorting - This is done by providing the user with a search box, and the ability to order by title, or date--both of which may be sorted by ascending or descending order. This works by sending the server query data whenever the user selects a new sorting order. It will stringify the value of the search box, detect what to sort by & in what order, and send this data off to the server--which will create an SQL search, and return the results. Afterwards, the gallery is updated with the new images.
* Finalized Flask framework - Created a flask framework that handled all web page and AJAX requests, including: webpage serving, and SQLite3 database interfacing. Learning flask was a bit of a pain, but only the first couple pages of documentation were necessary for implementing all the important aspects of this project.
  + Web page serving & request handling - This part was relatively simple, in Flask each file is represented by a portion of Python script--titled as the directory it should be found at. In the case of serving the index page, the web site would send a request server, asking for the page at the directory “/”. The python script with the title directory of “/” would be executed, returning the data for the index page.
  + All SQLite3 interfacing - This part required some research into how to do some slightly more complex database queries. Image serving was completed by searching for an image that matched the requested image source, and returning that row. The gallery SQL query simply selected all image rows & returned them to the website. The search/sort sql appended wildcards to each string searched for, and looked for any image rows containing these strings. It would then sort the results by the requested method, and order them in descending or ascending order--all of this being done as a single SQL query.

**Calonge, Jhonathan** -

· Investigating MJPEG streaming and helping Christopher with an online GUI to control servo motor.

* initialized Flask framework
  + [[details]]

**MacLarion**, **Sean (GitHub Username: N02788929)** -

· Wrote Midpoint documentation

· Wrote Parts List

· Wrote and Designed Slides about Camera Boom Design and Implementation

* Wrote a Lessons Learned and Implementation page for the design and printing of the Camera Boom.
* Added Images on Component Pages and Boom Design Screenshot for Final Presentation

· Wrote Boom Step-by-Step Documentation

* Included in this Documentation is a step by step procedure of how to make a 3D printed model using Google SketchUp and the 3D printing lab here at New Paltz

· Researched Methods for 3D printing

* I researched popular 3D modeling software and started using Google SketchUp. It was a simple program to use and made it simple to design a model that would work as a boom for our project.

· Drew designs and measured parts to use during 3D printing design.

* I had to measure the servo motors and wires we were using to determine the dimensions to create the model. The dimension of the base to the stand had to be less than 1 meter for the camera wire to reach and the gap between the bottom base the the overhand had to be greater than the height of the Pi.

· Designing Boom with Google SketchUp

* Using Google SketchUp, I designed a base for the pi with a overhang to mount the motors on. The boom was designed to have an outlet for all the wires to feed into. However, the process of printing did not allow any unsupported structure and it could not be implemented. Once, the design was made, I used the ruby plugin(mentioned above) to export the STL file to be printed.

· Modifications made to the boom after printing

* After the boom was printed, I had to make some modifications using a Utility Knife to cut out some supports and pieces that were used to make the boom functional.