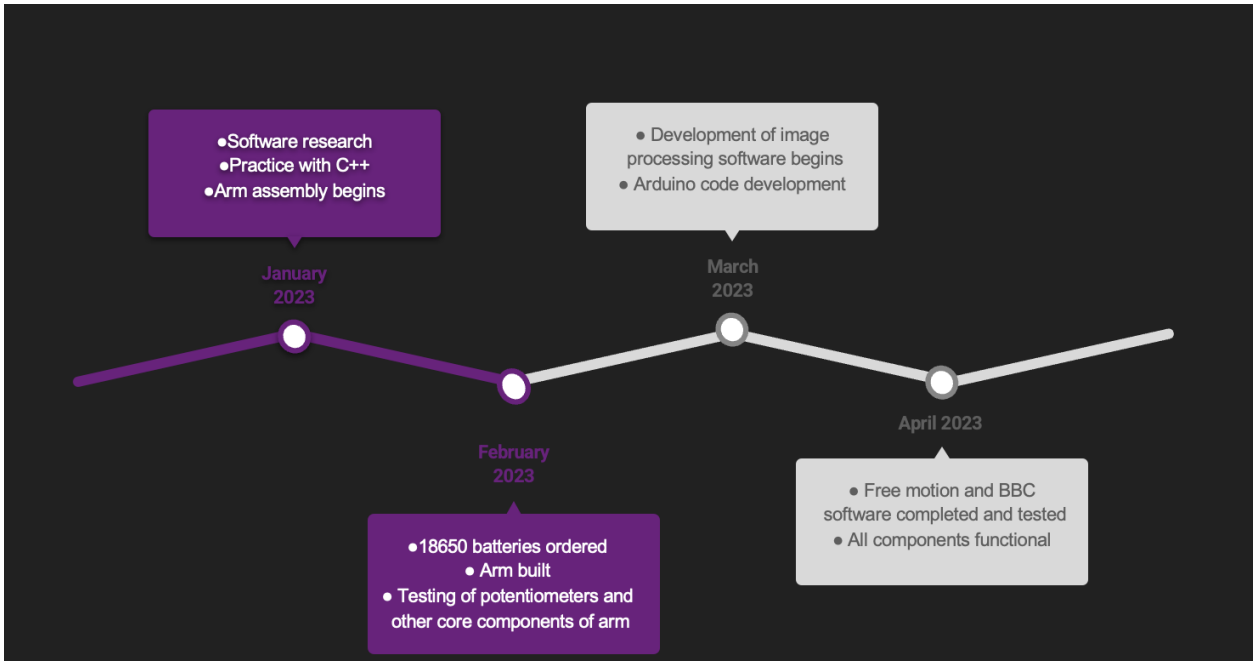
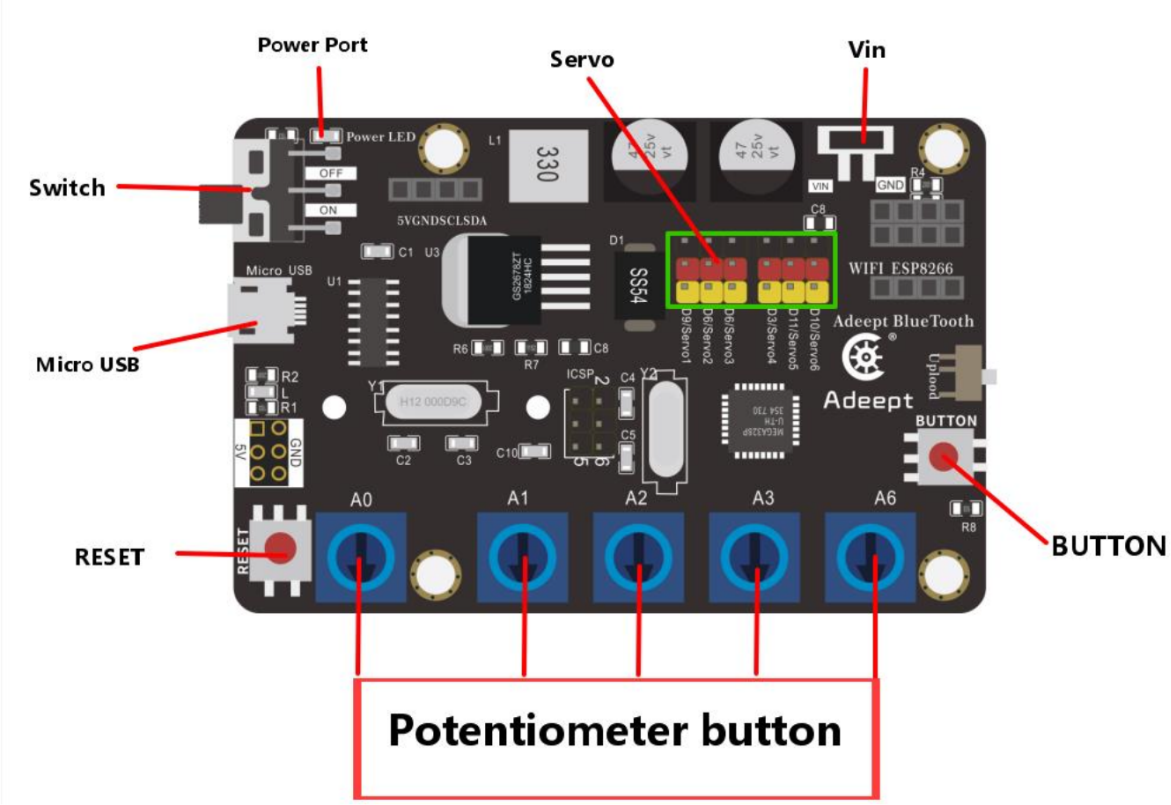
Robotic Arm Programming

Project Description: Build a robotic arm that is capable of both free motion via user input and able to pick up user determined objects

Timeline:



Hardware: Adeept SmartHub Development Board



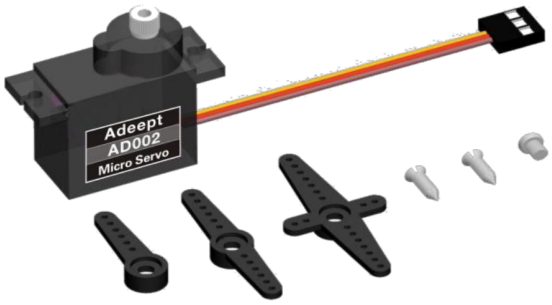
- Potentiometer Buttons: Control five servos manually
- Reset/Spare Button: Reset the board or serve other function
- Vin (6-24V): Power Supply
- Power Switch: Turn on power supply
- MicroUSB Port: Serial communication and power supply
- ATmega328P Microchip with 32KB of flash memory

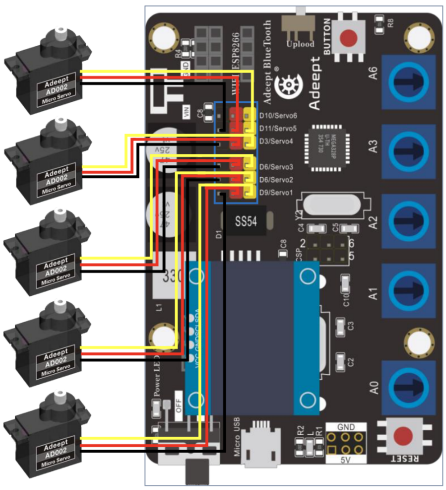
The potentiometer buttons are not used in the final product, but were heavily utilized during tested and assembly. The potentiometers allowed us to manually the angle of each servo individually to ensure proper function and that the arm had been assembled correctly. Once the arm was finished being assembled it was connected to a laptop via micro USB to provide power, and we found that the micro USB port provided both insufficient power and that the arm failed to function. Following this, 18650 sized batteries were ordered and installed after which the arm still did not function properly. It was then determined that the joints on the arm had been tightened far too much and in doing so servos two and three had burnt out and needed to be replaced. New servos with metal gears were then installed with the joints appropriately loosened.

Once the arm was once again powered on the potentiometers were used to manually test each servo functioned properly. The second servo still had minor issues with bringing the arm back up, and it was determined that this was because the servo itself was not held in place tight

enough meaning that the servo was having to utilize much more force than intended to bring the arm upright. The servo was then replaced, tightened, and passed all tests. Another function that the potentiometers in conjunction with the OLED Screen was to display the current angles of each servo so that the grabbing motion could be performed with the proper angles being recorded to be used in the Ball By Color function.
The micro USB port was no longer used as a power supply after the batteries were installed and instead only utilized for serial communication.
The microprocessor found on this board is the ATmega328P which is the same microprocessor found on an Arduino Uno board. The microprocessor is equipped with 32KB of flash memory to store the program on, 2KB of SRAM, 1 KB of EEPROM (only utilized during testing), and a 16 MHz clock speed.

Servo/Potentiometers





The servos in order were:

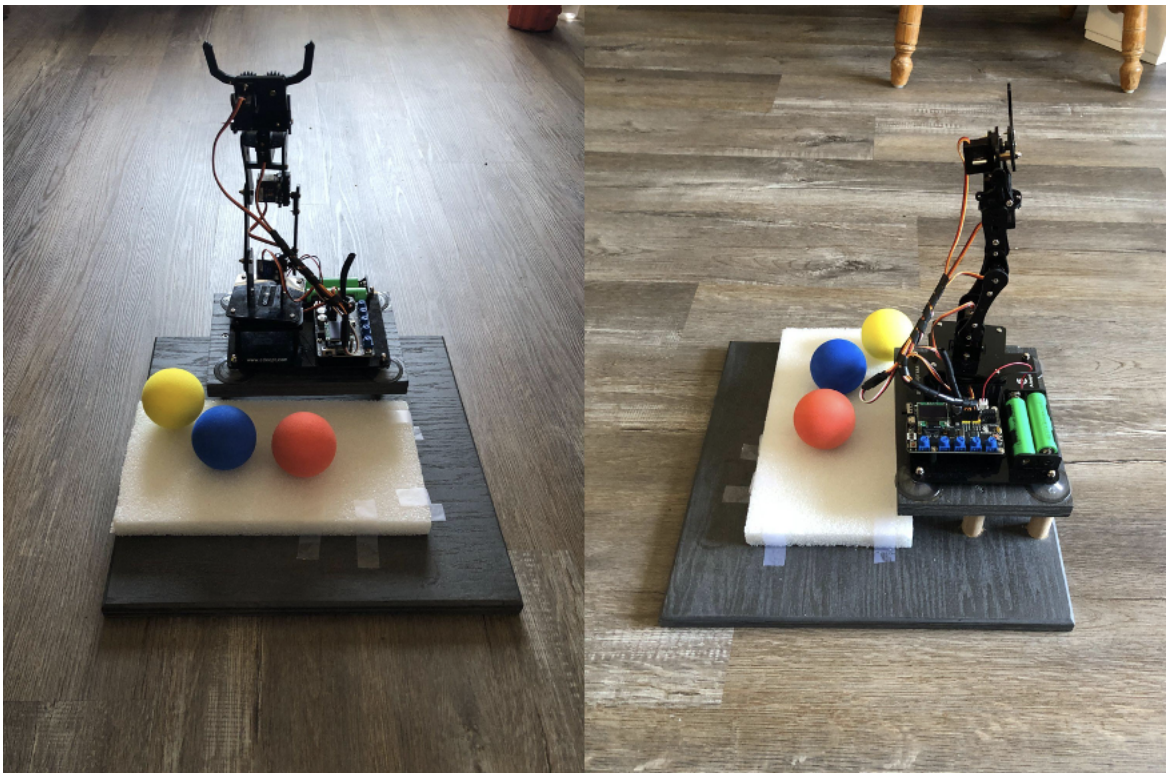
- Servo 1 - **Base**
- Servo 2- **Shoulder**
- Servo 3- **Elbow**
- Servo 4- **Wrist**
- Servo 5 - **Grip**

Each servo could be manually controller using the potentiometer buttons found on the right side of the board with A0 through A3 and A6 controlling their respective servo in order.

Capabilities

Ball By Color Mode: Allows a user to place three balls colored red, blue and yellow into three predefined positions. The arm will then grab the correct ball that the user prompts for. The user can place the balls in any arrangement and the Arm will recognize where it is.

Free Motion Mode: Polls a game controller for the current state of held buttons and translates that data into servo movement. This allows individuals to manipulate the arm freely via said controller.

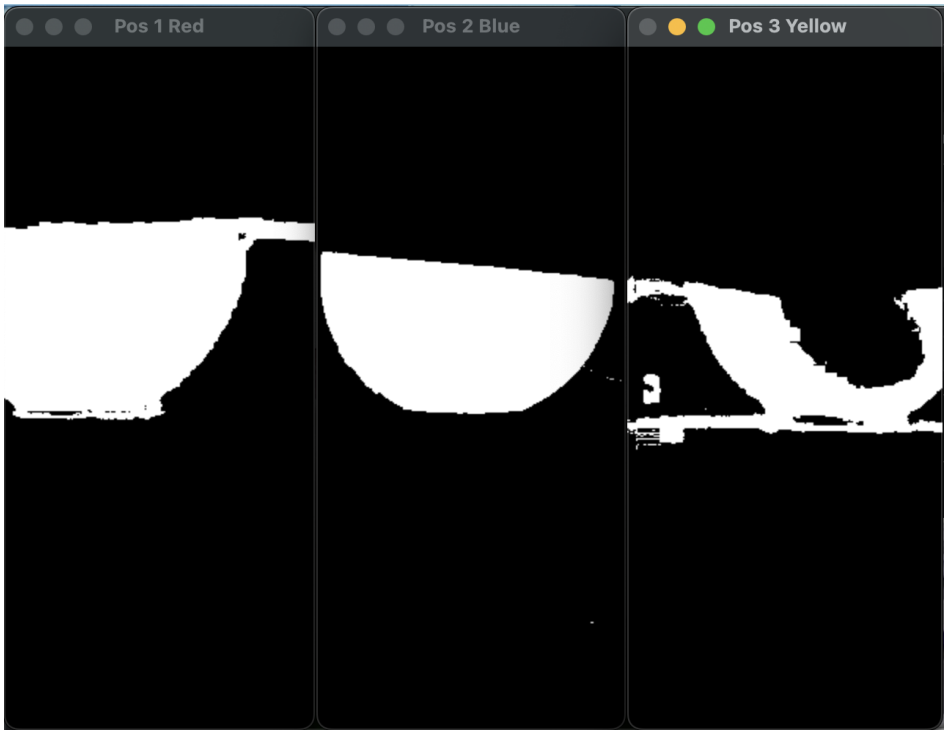


Ball By Color: Image Processing
Libraries used: Open CV2, NumPy, pySerial

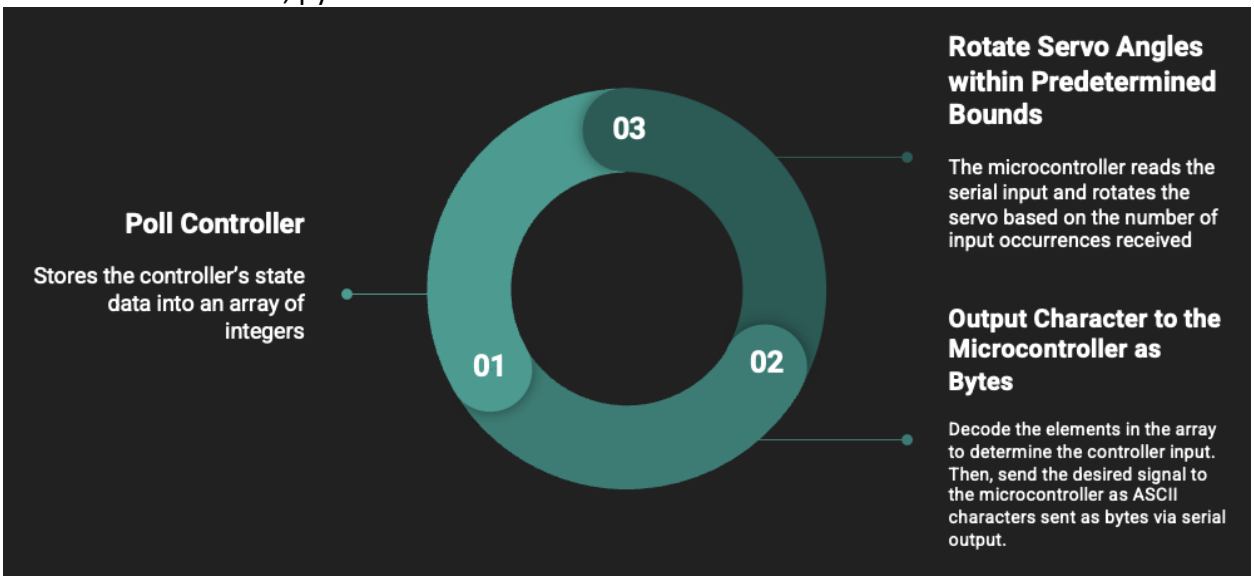
- Video is a collection of frames. Frames are merely collections of pixels.
- Pixels = data. Pixel values can be stored in array and manipulated.
- Pixels stored in a matrix with three dimensions, split into three positions.
- Masks set up to evaluate presence of color in each position
- Masks are established using upper and lower bounds of pixel values
- xLogic: If the color is in the position && user picked that color -> send ASCII characters as bytes via serial output to indicate which position to grab from

The Ball By Color Mode was established by using the OpenCV library available for Python. Additionally, for the purposes of array manipulation, we used the NumPy library. Using a simple, low-resolution webcam, we took a “snapshot” of the positions and established masks. There is a mask for each color of the three positions (a total of nine masks). To establish the masks, we used the OpenCV “inrange” function, which takes three parameters- the video frame to be manipulated, a lower bound color value, and an upper bound color value. The image would then be thresholded and only the pixels within the assigned bounds would appear as white. These white pixels were then tallied using the numpy sum and numpy nonzeroes method. The combination of these two functions essentially allowed us to count the number of pixels that registered within the bounds of the color we were searching for and assign those to a variable that would be used as a weight. If said weight exceeds a certain count of pixels, we would output “X color is present in Position X” to the command line. This is how we isolated the

colors and determined the presence of the colors. This also gave us free reign to place the balls at any position and be able to be detected. If we wanted the positions further away from the arm, we would adjust our conditional statements to look for a smaller weight value. If we wanted the positions further away, we could increase the necessary weight value.



Free Motion:
Libraries used: hidAPI, pySerial

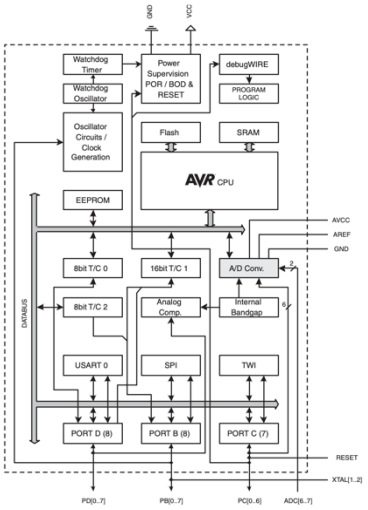


The free motion mode was tricky when it came to transforming controller input to a servo response, and it is done using the hidAPI library for Python to collect data stored in the

controller's registers. To do this, the vendor ID and product ID must be provided so that the program knows which device to connect to. A read() function is then used to collect the controller's current state of held buttons and the data is stored into an array. Each element in the array is a different byte, and specific bytes change based on which button is being pressed or held down. For example, when the start button is pressed the seventh byte is 100000 or 32. A while loop is run in both the ballByColor() and freeMotion() functions in the python script that continuously poll the controller for data and store it in the snes[] array. Once the data is stored, several if/else statements are run through to determine if any buttons are pressed or held down. If an element in the array indicates that a button is being pressed the program then outputs an ASCII character as bytes (using b"char" format) via serial output to the microcontroller. Within the Arduino code, a do loop is ran while a serial connection is available that waits for serial input from python. Once a character from python is received it is then passed through numerous conditional statements to determine which servo is meant to be moved in which direction. The correct servo angle is then adjusted one degree in the correct direction and both loops on the board and Python continue.

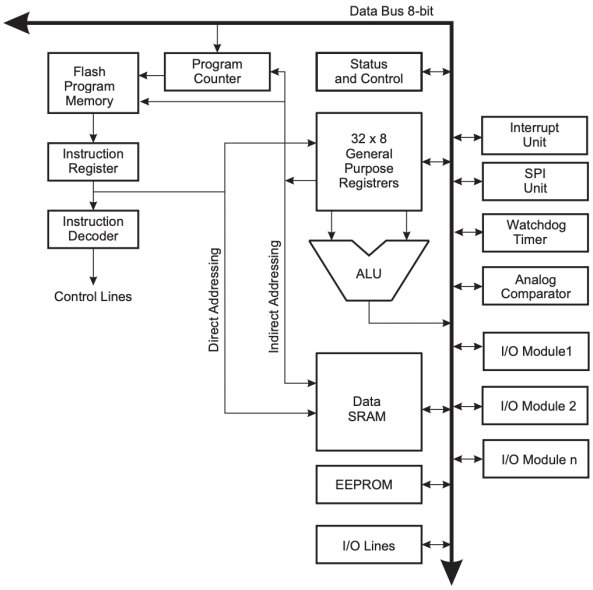
ATmega328P Microcontroller

Block Diagram



and

AVR CPU Core



Data Memory Map

32 Registers	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F
160 Ext I/O Registers	0x0060 - 0x00FF
Internal SRAM (2KB)	0x08FF

As previously mentioned, the ATmega328P microprocessor is the same as the one found in the Arduino Uno. Because of this, all of the programming within the Arduino IDE was written, compiled, and uploaded to the board as though it was an Arduino Uno. Overall, the microcontroller had plenty of processing power to carry out the instructions/functions needed within the program. When attempting to implement variable rotation speed based on the analog stick input from the game controller however the processor and bandwidth did show its limitations. The servos moved very slowly and occasionally jerk in the direction held. Another drawback of the processor was the libraries recommended for the OLED screen caused the program to run very slowly as well as the issues that were present for the initial presentation (which will be touched on).

Arduino Pseudocode (the actual code will be provided with this report):

setup()
- Attach servo objects to the correct pins and set all five to 90 degrees
- Initialize OLED screen

slowWrite(servo, currentAngle, targetAngle)
- move servo angle to the desired target angle with 18ms delay after each degree adjustment

loop()
- Picture loop to print text to the OLED screen
- Read serial input
- Select other mode if input specifies

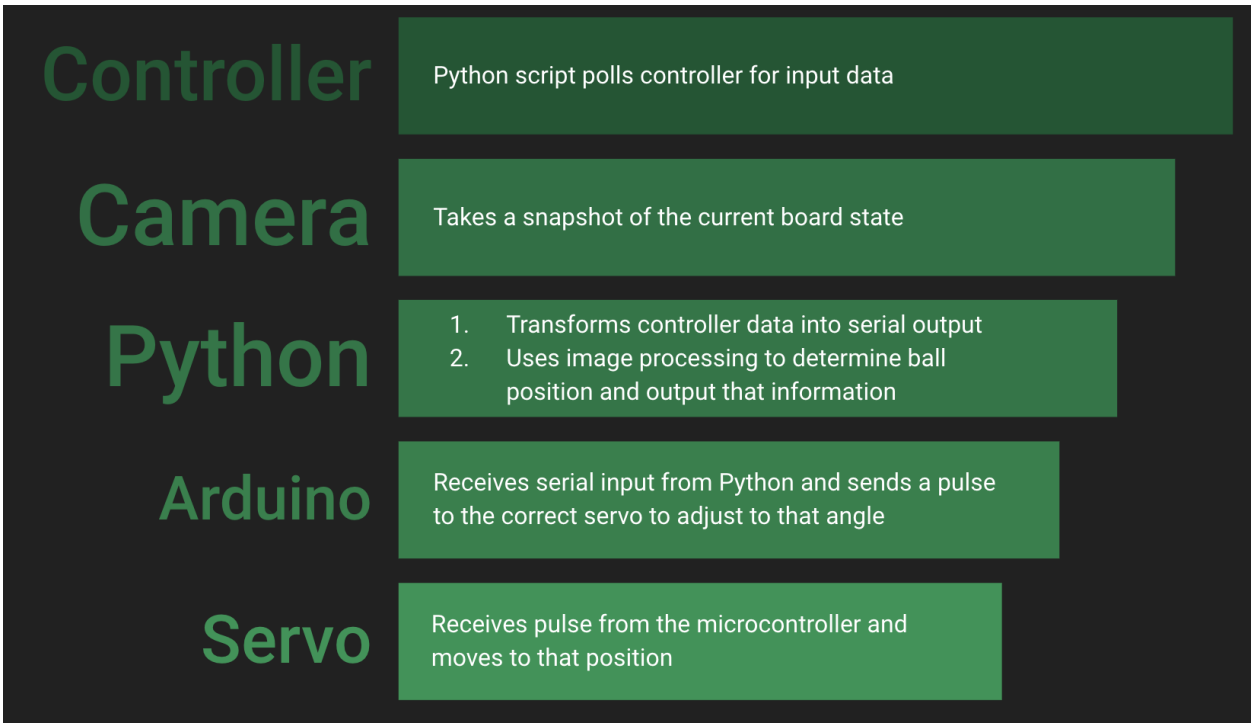
(if mode == 0) //free motion
- Adjust servo angle by one degree in the desired direction

(if mode == 1) //bbc
- Read position input and grab the ball in that position

The Arduino code (written by Michael) was one of the most difficult areas of this project given that I had no knowledge of robotics or writing C++ code. I spent most of my time working on this aspect of the project studying sample Arduino programs and understanding how the specific board used in the project worked. I also had to spend an extensive amount of time learning more about servos, how to display text to the screen, and serial communication. Once I had a good understanding of how the code was to be written, I found that to be the easiest part. The most difficult part in fact was the extensive bug testing and problem solving that I had

to do to get the code to work. A major issue that the arm had was that if the voltage was too low or the program crashed the servos would rotate rapidly at random angles until the power was disconnected. Additionally, I wrote a slowWrite() method to slow down the speed of the servos with a while loop containing a delay since the write() method had the servos moving far too fast. Additionally, within the do loop there multiple nested if/else statements so that the processor does not have to trace through the entire loop each iteration. The if statements start with one large if/else dividing the two modes so that only one is traced through to decode the serial input received into servo movement.

Datapath:



The datapath follows the figure above, with 1 in the Python block representing the free motion mode and 2. representing the ball by color mode.

Challenges and Limitations

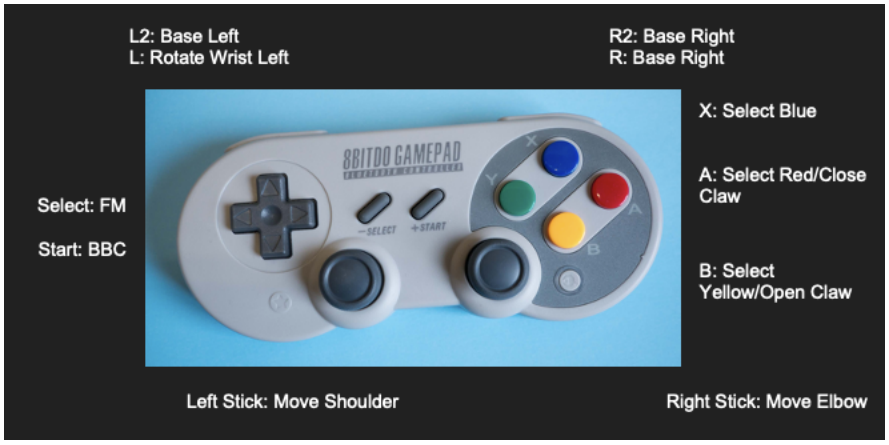
- Waiting for parts and replacements to arrive
- Multiple servos burnt out and needed to replaced with those with metal parts to avoid burning out again: This causes the motion to occasionally be shaky
- Allocating enough time out of our schedules to efficiently build, test, and debug physical and software components of the arm
- Robotic Arm cannot submit 2023 tax documents to the IRS
- Integration of Python script with Arduino board code
- BBC Mode image processing relies on minimal background noise and good lighting

Many of these challenges and limitations were previously mentioned, though I (Michael) would like to touch on them once again as the BBC Mode's image processing code has been improved to be able to function in multiple light settings and background noise is no longer an issue. This was accomplished by implementing additional conditions for determining if a color is found in a position, most notably the condition checking that the chosen position must have more pixels of the desired color than the other two. The arm still is not able to submit 2023 tax documents to the IRS unfortunately.

Possible Improvements

- More possible color or placement options
- Variable movement speed based on analog stick input
- Able to submit 2023 tax documents to the IRS
- OLED Screen could indicate which ball is being picked in which position
- The balls may overlap positions if the camera is not oriented correctly

Having the variable movement speed based on the analog stick input was a goal that possible in theory, though it would require sending an input that could be plugged directly into the delay method without an arithmetic or memory access by the microprocessor. It is important to have the camera oriented correctly, and to accomplish that ArmCapture.py is written to test both the positions as well as the lighting for the camera feed. Following the presentation I (Michael) did take time to implement code to print which ball is being picked in which position to the OLED screen, which I will touch on. In the coming months having the arm be able to submit 2023 tax documents to the IRS would be a great stretch goal, however it may not be possible as the submission deadlines has unfortunately passed.



Pictured above is the controller and button layout used for the program. Currently this is the only controller that may be used as the code uses this specific vendor and product ID as well as the byte values for each button press (this varies between different controllers).

Conclusion

The presentation as a whole went very well up until the demo. During the setup for the presentation the camera was knocked out of place and this was not noticed until the demonstration when the program was not functioning correctly. Once the camera was reoriented the board then lost power once connected to both the battery power and micro USB again (due to the new batteries installed the day of). Eventually the arm was able to receive power but the display could not initialize due to the Adafruit library used for the screen functionality causing the program to hang. The arm was eventually able to function as intended, but it was unfortunately too late. Following the shaky demonstration, I (Michael) took the weekend to figure out what happened as it was not an issue that had previously occurred during development. I discovered it was both a voltage and display related issue after I had switched back to the old batteries and removed the display code from the program. I then rewrote all the display code using a u8g library that utilized a picture loop rather than print statements scattered throughout the code. The picture loop was tricky to figure out, but I have it to where it only loops once and terminates after that first loop. After that it only loops once again once a Boolean variable is set to true. I also made changes to the masks that made the ball grabbing far more consistent so that the small demo on Monday would be as smooth as possible. The arm now has a very high consistency rate and it is also now very responsive, especially when switching between modes (which before would take several seconds and now is instantaneous; this is due to using the new library that loads the display much faster). Overall, the project was a joy to work on and gave the three of us a much improved feeling of self-efficacy when it came to complex Computer Science projects. It was also very enlightening in terms of the benefits and possible difficulties that come with team projects. The project also provided each of us with a much better practical understanding of CSCI 310 concepts such as memory bandwidth, I/O, CPU architecture, data hazards, and more.

Reference Links:

https://pythonforundergradengineers.com/python-arduino-LED.html#use-the-python-repl-to-turn-the-arduino-led-on-and-off
https://pythonforundergradengineers.com/python-arduino-LED.html
https://github.com/arduino/EduIntro/blob/master/examples/by_topic/PhysicalPixel/PhysicalPixel.ino
https://www.tutorialspoint.com/how-to-find-the-hsv-values-of-a-color-using-opencv-python#:~:text=To%20find%20the%20HSV%20values%20of%20a%20color%2C%20we%20can,%2C%20255%2C%20255%5D%20respectively.
https://blog.thea.codes/talking-to-gamepads-without-pygame/
https://github.com/trezor/cython-hidapi#documentation
https://howtomechatronics.com/how-it-works/how-servo-motors-work-how-to-control-servos-using-arduino/
https://code.google.com/archive/p/u8glib/wikis/userreference.wiki

Division of Project Work:

Josh: Assembly of the arm and help with testing servo functionality
Jacob: Assisted assembly of the arm, image processing code, and construction of the wood base
Liam: Controller input and Arduino code as well as the integration of the image processing code

All three: Functionality testing, minor physical fixes, supply trips, and presentation preparation.