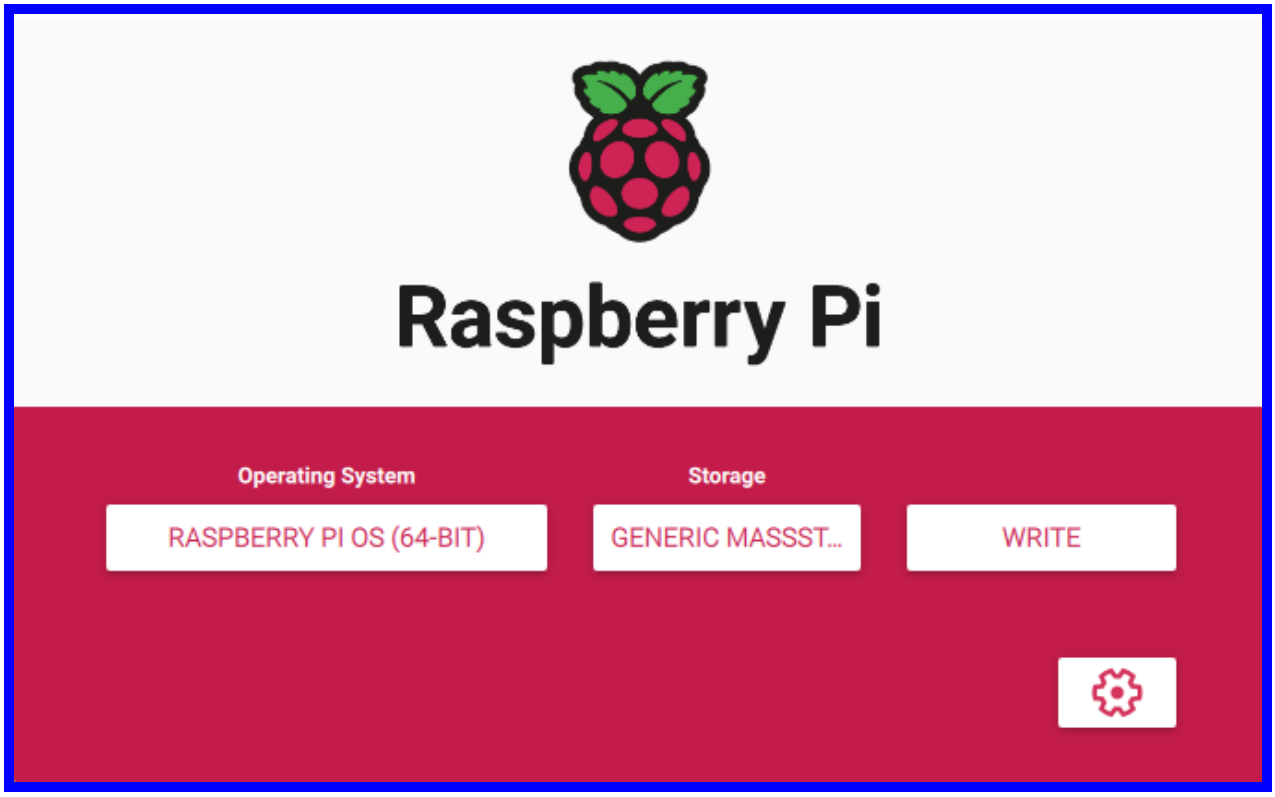
Ball Tracking Robot (v2.0.0)



Raspberry Pi Setup Summary

First we tried to set up the Raspberry Pi on the 64 bit OS. We tried to set up the camera with the raspberry Pi but it didn't work because there was new camera software installed. When we disabled the libcamera, the VNC would not work. Hence we re-setup the Raspberry Pi with the 32-bit os and this worked. Then we tried to reinstall opencv but it would not work. Hence we installed a new camera module named Picamera to get the output from the camera since the open-cv video capture does not work. After doing all of this I'm able to get the camera output working.

Specifics for setting up the Pi

1. Use raspberry Pi imager to write os and settings onto a sd card. In the settings, turn on SSH, set the hostname, set a username and password and allow it to connect to the correct Wi-Fi.
2. Then insert the sd card into the Raspberry Pi and boot up the Pi.
3. Once the Pi is booted up try to ssh into the Pi using terminal:
   Command: ssh pi@raspberrypi.local
4. Run sudo raspi-config and go into interface options. Then enable VNC and change the resolution to a relatively large number.

5. Open the vnc viewer and connect to raspberry Pi through the search bar.
6. Then connect VS Code to ssh through the ssh extension.
7. Then connect the terminals through the button in the bottom left corner.
8. Cd into the correct directory.

Camera

We then connected the ArduCam to the Raspberry Pi through a slot on the Pi. The blue tape on the camera cable must face the black clip on the connector.

Robot Assembly

Assembling the robot kit was pretty challenging as there were a lot of parts and the instructions on the website weren't easy to follow. It took approximately eight hours to assemble the entire robot. Once we had the robot assembled we moved onto the wiring. The wiring was not too complicated as the kit came with a Raspberry Pi hat which made all of the wiring relatively straightforward. The website also provided diagrams of what each port on the hat was used for, which sped up the process even more.

Software Creation Process

Image Detection / Processing

First we tried to get the contours in an image and then tried to filter out the ball. However it didn't work because we were unable to filter out the ball. So we tried a different way of generating the binary image that was being used to generate the contours, but we were still unable to filter out the ball. For the third and final iteration of the software, we decided to use image masking to create the binary image. To do this, we first changed the camera configuration from stream_configuration to still_configuration. This allowed us to take images in RGB instead of YUV420 as it was in the stream_configuration. RGB images are just easier to deal with and manipulate. Once we got the image we converted it to BGR which we used to convert the image to HSV. Once the image was in HSV we set up lower and upper HSV color bounds which allowed us to filter a specific color out of the image. This is called image masking. This generated a binary image that replaced all the colors we wanted to detect with white and the rest with black. Once we had that image we then proceeded to create a kernel for it. We used this kernel to erode and dilate the image which helped smoothen out the image. Initially, we used the average location of the white pixels to find the center of the object. However we realized that we could use this new binary image to find all of the contours in the image. We then used the areas of the contours to filter out the object. We then drew a rectangle around the edges of the object. The center of this rectangle was the center of our object, and we used this center to move the robot towards the object. We found that this method of finding the center was a lot more accurate than the average location of all of the white pixels.

Distance Measurement

We also wrote software to accurately measure how far the object was from the robot. Our first idea was to perform linear regression and find a linear pattern between the number of white pixels and the distance from the ball. This failed because the area of the object gets quartered while the distance only gets halved. Hence we decided to use another concept called focal distance. The focal distance(F) can be calculated by taking an object with a known width(W) and then placing it a known distance(D) away from the camera. We can then take a picture with the camera and then calculate the width of the object in the picture taken by the camera (Perceived Width)(P). The focal distance can be calculated using this simple formula: $F = (P*D)/W$. Once we have our focal distance we can then use it to calculate how far the object is from anywhere using this formula: $D = (W*F)/(P)$. We used this to stop the robot as soon as it was 12cm away from the object.

Robot Movement

The robot moved by rotating the two motors in appropriate ways and turning the front wheels to point in the direction of movement. The motors were programmed to accept PWM (Pulse Width Modulation) which allowed for the car to move at different speeds. This made the robot move smoother and less shakily. This also allowed me to position the robot more accurately. The front wheels change direction using a servo motor that is controlled through software depending on where the object is located. The direction that the front wheels point in is crucial to the robot's movement. The rotation of the front wheels is crucial because if the front wheels don't rotate, then the robot won't be able to turn in either direction.

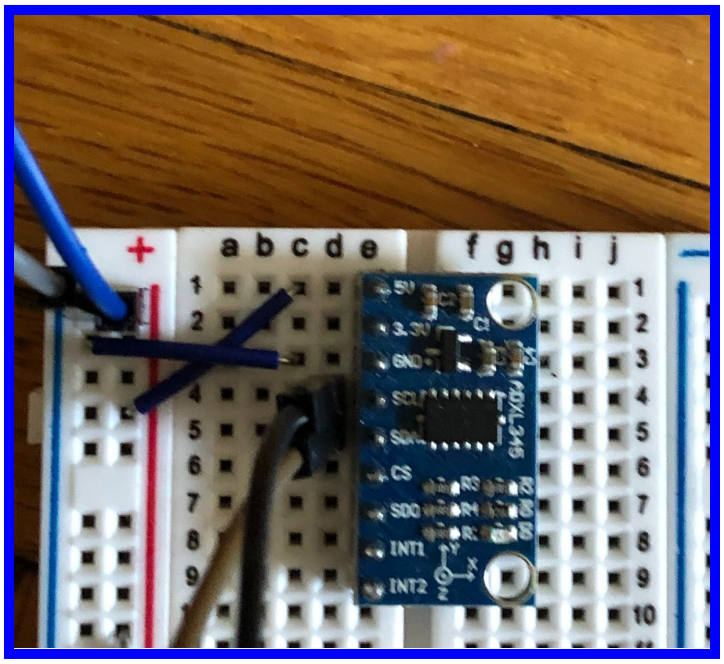
Robotic Arm Movement

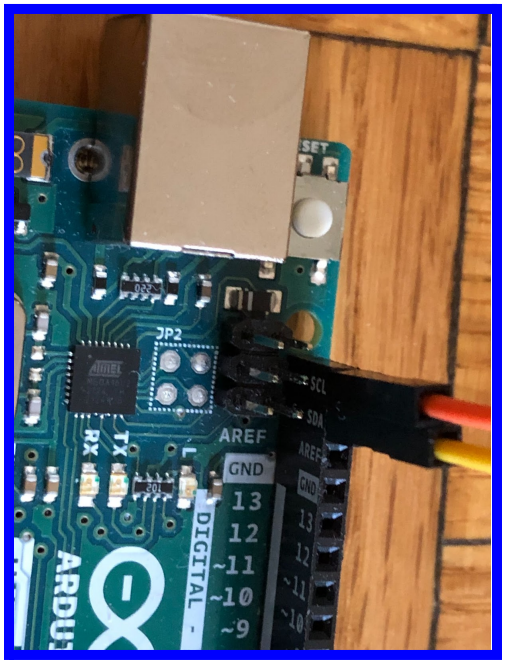
Once the robot stopped we had to figure out how we could make the arm move to the appropriate position to pick up the object. To accomplish this, we decided to use a technique called inverse kinematics. We implemented this by creating a triangle whose side lengths included the "forearm" of the robotic arm, the "palm of the robotic arm", and the distance between the robot and the arm. With these lengths we were able to calculate the angles of the triangle (the angles that the servo motors would need to rotate to move the arm in the correct position to pick up the object.) We ended up using the law of cosines to calculate these angles and then we also added and subtracted some offsets to make the angles of the triangle line up with the actual axis of rotation. Once we had the angles, we made the servos rotate to that particular angle using some additional software. The final modification we made to make the system more accurate was to rotate the arm in a particular direction depending on where the center was in the final image after the robot had stopped. This guaranteed that the robot would successfully pick up the object every time.

Link to the github repository for the code.

Setting up accelerometer and Arduino

We decided to set up my accelerometer with the Arduino. First we installed the Arduino IDE so that we could write and upload code onto the Arduino. Then we had to install the Adafruit ADXL345 library. We then installed the example code for bluetooth modules from the Arduino IDE. Once we had that properly set that up we calibrated the accelerometer offsets so that the accelerometer would accurately collect data. Finally, we calibrated the motions that the accelerometer would detect and then what it would do for each respective motion.





Setting up the bluetooth module

To set up the bluetooth module, we started with getting the bluetooth module on the breadboard with the accelerometer and then properly wiring it to the Arduino. Once we had that set up, we had to configure the bluetooth module in software. Once we had it configured we then enabled bluetooth on the Raspberry Pi and then connected the bluetooth module to the Raspberry Pi. We then used a library called pySerial to collect the information that the Arduino was sending. Once we had that data we used it to make the robot move according to the gestures made by the human. Then, every time we need to connect the 2 devices we had to run these commands:

1. Bluetoothctl
2. Open new terminal
3. rfcomm bind rfcomm0 00:14:03:05:0A:F0