ECE 478/578 PROJECT 2 REPORT

Fall 2018

Project:

Schrodinger Cat

Submitted by:

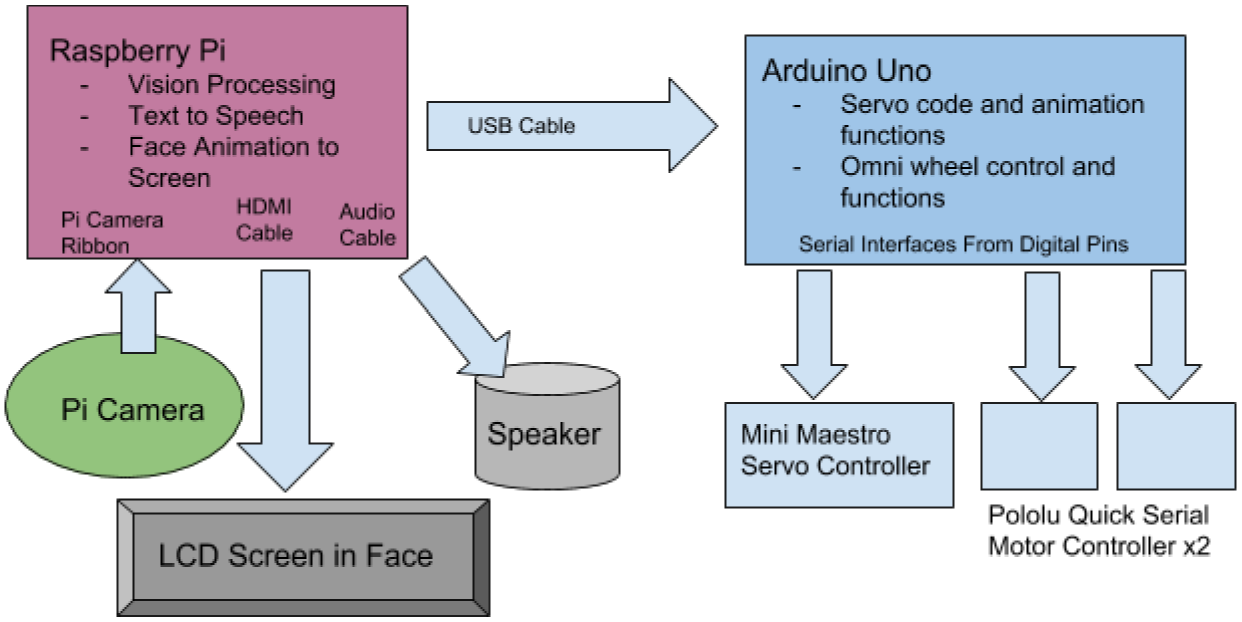
Rakhee Bhojakar
Chelsea Brooks
Erik Fox
Dakshayani Koppad
Jamie Williams

Table of Contents

Project Summary

Schrodinger Cat (what used to be the Einstein robot) is one of the oldest robots in the lab. Schrodinger Cat is made up of a full body (head, arms, and legs controlled by servos) suspended above the ground by a base (which moves via a set of omni-wheels). The servos and motors are controlled with an Arduino Uno microcontroller. We added a Raspberry Pi to the head of the robot to add vision, speech, and a screen to the robot.

The control scheme of the robot is as follows:



Raspberry Pi
- Vision Processing
- Text to Speech
- Face Animation to Screen
Pi Camera Ribbon
HDMI Cable
Audio Cable
USB Cable
Arduino Uno
- Servo code and animation functions
- Omni wheel control and functions
Serial Interfaces From Digital Pins
Pi Camera
Speaker
LCD Screen in Face
Mini Maestro Servo Controller
Pololu Quick Serial Motor Controller x2

For project two, we had to do even more interfacing in order to get all of the processing power needed for the robot, since it was required to run on ROS. We had the speech to text/text to speech running on a separate laptop in order to minimize the computational node on the on-board Raspberry Pi. We also offloaded the camera processing nodes to a third laptop. For ROS, we also had some issue with displaying the face to the LCD screen, which did not end up being part of our final demo.

Previous Work on Robot

When we received the robot, it came with all the hardware needed control the motors and servos. There were minor issues, however, which lead to redoing much of the wiring of the board, as well as replacing a few components which had died. We also removed the Bluetooth capability, as it was not needed for this Project.

The previous group used a laptop to provide most of the intelligence of the robot. We decided to change that to a Raspberry Pi to take advantage of the Pi camera and image processing abilities. We also removed the existing robot head, and replaced it with a 3D printed frame and an LCD screen showing a face animation.

Team Contribution

Text to Speech(Amazon Polly) / Speech to Text(Dialog Flow) – Chelsea

Face and Robot Detection ROS node - Rakhee

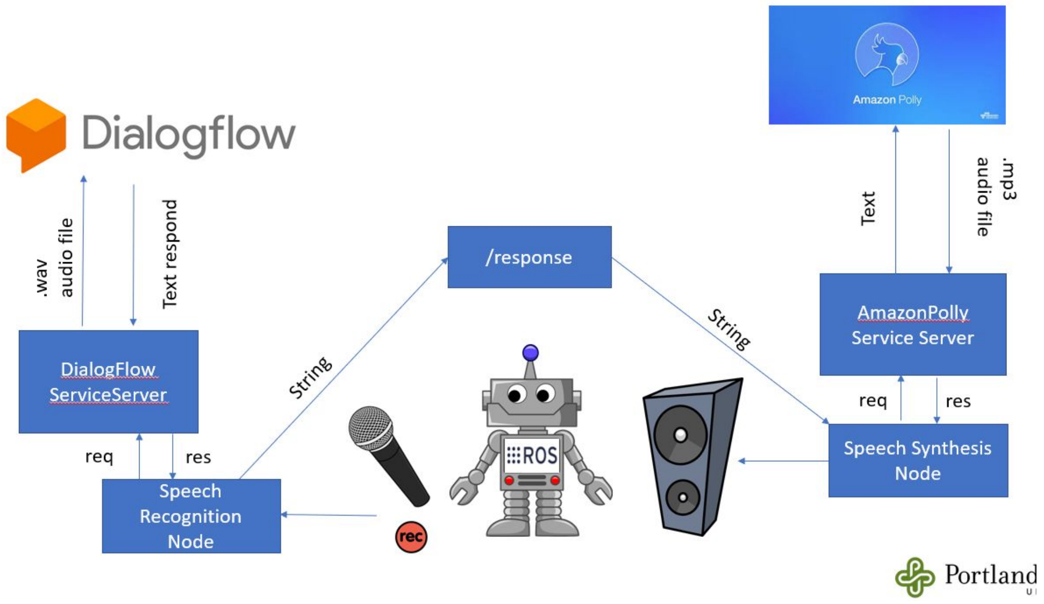
Servos, Arduino Code, Script Writing - Jamie

ROS node for Wheels, Servos, Arduino Code - Dakshayani Koppad

ROS System, 3D Printing/Screen Face - Erik

Text to Speech/Speech to Text

Text to Speech/Speech to Text is a great way to give life to your robot. Our original robot did not have text to speech capabilities, so we used ROS, Dialogflow, and AmazonPolly to give Schrodinger’s Cat a voice. With amazon polly, you can change the gender, speech speed, spaces between words, language, emphasis, rate, volume, etc. Dialogflow allowed us to train our cat what to say when asked questions so that it would make it seem like you were having a conversation with the cat.



Amazon polly is a text to speech synthesizer. Amazon polly allows for the robot to have lifelike speech. For our cat robot, we used the voice Ivy. Ivy is a little girl's voice. We thought it was the perfect voice for our cat. We used amazon polly to record the robot theatre script. In the script, we used SSML format to emphasize words, make the robot louder or quieter, raise the pitch of the voice when it was annoyed. The SSML made the speech more realistic. Prosody controlled the volume, rate, and pitch of the cat's voice.

```
Text = '<speak> <s><emphasis level="strong">Meow! Hi</emphasis> <prosody volume="x-loud"> everybody </prosody> </s><s> <prosody volume="x-soft">I am Professor Schrodinger</prosody> </s><s> <prosody volume="x-loud">  I mean, I am Professor Schrodingers cat </prosody> </s><s> Schrodingers cat, for short. </s> </speak>'
```

We also wrote an amazon polly node that talked with a dialogflow node so that the cat could respond to our questions. Dialogflow is a natural processing tool. Dialogflow allows for the user to train their robot what to say to different commands. The website takes in the speech and outputs a text line that is then sent to, in our case, amazon polly.

We used ROS to implement dialogflow and amazon polly nodes. The amazon polly node ran with the rosrun command and the dialogflow node ran using roslaunch. The amazon polly node was written as a service.

One thing we wanted to do for this project was put our ros speech node on our raspberry pi, but with our network issues, we were unable to have the raspberry pi run amazon polly and dialogflow. What we did implement was we placed amazon polly code inside our nodes so that the robot would talk when it detected a robot with the camera.

The text to speech function of the robot was embedded in the vision detection python script. Using the robot's ability to talk, we could learn what the robot was seeing. This was helpful for testing and debugging the vision software. When the robot detected a yellow object we could have it say that it saw the object, which side it noticed the object on (left or right), and then it could say when it saw the object leave its frame of vision. It would also say a sentence of greeting or describe how it wanted to interact with an object (e.g. "Robot detected on right")

Robot Theatre

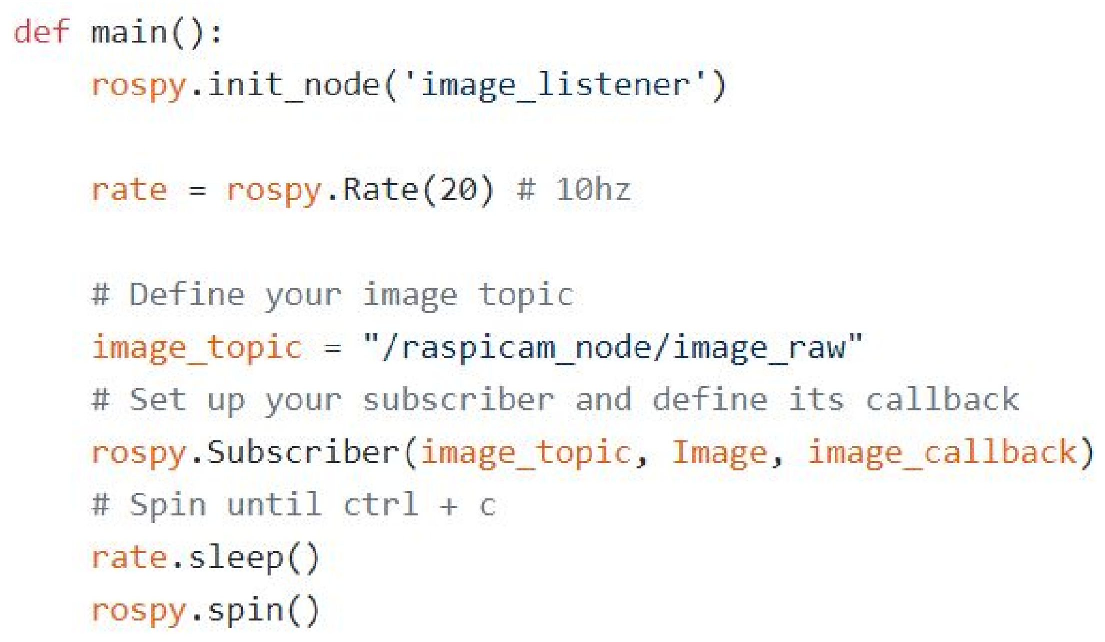
Project two required a short play segment as one of the requirements. For our robot theatre play, we were paired with a team working on a robot to represent Einstein. While a script for dialog between Schrodinger's Cat and Einstein already existed, we had to modify the script slightly to match the capabilities of the robots, and what we thought we could reasonably accomplish in the time we were given for the project.

First of all, we cut down the length of the script considerably. The original script was very long, and included several different scenes. We cut the script down to 40 lines, which took about 4 minutes for the dialog alone, without pauses. The script was also edited for clarity, as many of the original scene transitions were abrupt and made little sense as an overall play. To coordinate actions, the modified script also contained stage direction for each of the robots, different from the stage direction of the original script, that was written to match the limited capabilities of each robot. The speech for the play script was done via text to speech software (see above section for description), and both parts were played on the same laptop, to facilitate communication.

For the stage direction of the robots, we had to be very careful with how much action we could make them display. Schrodinger's Cat robot is very large, and has very powerful motors to control the wheels at its base. In comparison to the much smaller turtlebot (the form factor of Einstein), it could do significant damage if it started moving out of control. When testing the wheels of the robot, we had had issues with the wheels getting out of control, so we were very concerned about that happening for the play. Additionally, we were concerned that the stage would be too small for both robots, and then any movement could possibly send one robot off the edge. Because of this, we decided to only have the robot turn, instead of moving forward and potentially running over the other robot. Unfortunately, before we could perform the play, one of our wheels broke, and we were unable to demonstrate any wheel movement at all.

Robot Detection

For second part of the project we have implemented ROS node to implement detection of robot. Here ros node for raspberry pi camera is used instead of plain python code., which gives advantage of communication with other nodes in the network. First we have subscribed to
To image topic named /raspicam_node/image_raw as shown below.



```
def main():
    rospy.init_node('image_listener')

    rate = rospy.Rate(20) # 10hz

    # Define your image topic
    image_topic = "/raspicam_node/image_raw"
    # Set up your subscriber and define its callback
    rospy.Subscriber(image_topic, Image, image_callback)
    # Spin until ctrl + c
    rate.sleep()
    rospy.spin()
```

And using color detection algorithm we are detecting yellow colored robot. And when robot is detected using amazon poly , it says where robot is detected to the left, right or front.
And If robot is detected to the right side , amazon poly tells to move right. And conversation between two happens simultaneously. Once object is detected we have published the data that is integer number to other node to tell robot is detected. y _cordinate.py file taked data from image_listner.py file and decides to take action depending on where robot is detected. But due to issue with wifi communication over nodes, we were not able to publish data to other nodes running on other system to control the robots servos.So finally we kept only one node which detects robot.
Future work of this robot detection will be communication data obtained after detecting robot to another node through wifi and making robot take actions based on it that is either change direction or turn to left or right. And we could add more object detection and face recognition so that drobot can identify the person in front of it. Code files for this part is on github directory in ROS folder which comes under Software folder. Instruction to run this code is also given in readme file for pi camera node on github.

Robot Head (Screen and 3D Printing)

The original face for the Schrodinger Cat Robot was an ESRA- Expressive System for Robotic Animation. This device did not have a flexible design, nor did it look like a cat. Additionally, it did not provide an ideal frame to place a camera. After deciding to replace the ESRA with a design of our own, the first step we took was to determine the requirements for our design. We concluded that we needed our design to be expressive, adaptable, and be able to house our camera. A 3D printed cat face frame, with a display to show facial animations, checked all of these boxes.

Our final design consists of a 3D printed cat face frame, an LCD display, a camera, a Raspberry Pi, and a series of cat face animations. The frame is designed to have a abstract look of a cat, a hole to expose a LCD display, a frame to hold the LCD display, a hole to expose a camera, and a stand to mount the head to the robot. The cat face animations are sent to the display using OpenCV function calls.

When implementing our design, we ran into a few issues. The biggest challenge was the print of our cat face. At first the design was too large to fit on the only available printer. After resizing the design, to be as small as was possible with display we had available, it was still at the boundaries of the printer. This resulted in a few complications. The first was where to place the design on the printer software. It took multiple attempts to get the design placed correctly. The second issue was that printing surface had uneven heat. This resulted in failed prints because parts of the frame curved up and were hit by the printing head. Another challenge faced from printing resulted from the resolution of the printer. We anticipated this issue in our schematic by adding length to the dimensions, however it was not enough. To get around these issues for version 1.0, we used one of the printed faces where only the facade printed correctly. On this we created a support for the LCD with screws.

Aside from printing issues, there were some challenges faced with interfacing the Raspberry Pi with the LCD display. The first of these issues was that the Raspberry Pi wouldn’t fill the LCD monitor. This issue was fixed by making some changes to the /boot/config.txt file. The second is was that the OpenCV image output function wouldn’t take ownership of the display, which was necessary for the animations to look correct. We resolved this issue by including functions from the screen info library. Links to the solutions to these problems are included in the reference section.

The future changes to cat face should be to update the frame design, and to improve the animations. For the frame, each part should be printed separately, and the margin of error should increase so that the dimensions will work after printing. These changes would increase the likelihood that a print will be successful. Additionally, we should add an enclosure for the Raspberry Pi. As for the animations, we could improve them by utilizing digital animation software. This could allow for more realistic and professional looking cat faces.

Project 2: Use ROS to control cat face

For Project 2, the plan for improving the robot head was to utilize ROS. We developed the framework for this functionality, however, in the final implementation we experienced system failure. The face control node, which subscribed to a topic with data published from the image processing node and used this data to determine which animation to output on the LCD monitor, worked independently, however running it on the same Raspberry Pi as the camera it caused the Pi to freeze. For future improvements to the robot, we should implement the image processing and LCD display control on a different computer.

Robot Body (Servo control)

The body of the robot was made up of a full body (head, arms, and legs) suspended above the ground by the base. After removing the old head at the beginning of the project, we were left with 15 servos in the body. There were four in each arm: the wrist/forearm, elbow, and two in the shoulder. Each leg had 3: the knee, and two in the hip. There was also a servo for the neck, to turn it side to side, that was left over after removing the old head.

All of these servos were controlled from the same servo control board, a Pololu Mini Maestro. This board had been added previously, and supports 24 channels of servos, of which only 15 are being currently used. The Mini Maestro is controlled via serial connection to the Arduino, using two digital pins capable of serial communication (we used pins 10 and 11). Pololu maintains an Arduino library for the board (referenced at the end of this document) that allows for easy communication and control of the servos.

There were several issues with the wiring of the servo setup. First of all, the connections between the Arduino and the Mini Maestro were not done correctly, leading to the code we received not appearing to work. We managed to get the servos to work, but we found that the poor connections of the wires between the Maestro and the servos lead to multiple servos moving when only one was activated. We re-soldered those connections to make them cleaner and to provide larger wires for the power. Even after this, we found that there were several dead servos on the torso.

After fixing that issue, we found that the mapping between the servos on the body and the Mini Maestro channels was wrong, which led to the wrong servos responding to commands and moving in ways that could damage them. We had to spend time remapping each servo on the body to the appropriate channel on the control board. And, even then, the maximums and minimums defined by the previous group were not working as they should of. So, each of those parameters had to be redefined as well. The previous groups made use of only 3 servos per arm and only 2 per leg, so there were 4 servos that we needed to map and define from scratch. All of this had to happen before we could start working on animations for the robot servos.

Originally, for project one, the Arduino was supposed to receive serial commands from the Raspberry Pi about what the robot was seeing. The Arduino has code to initiate certain gestures when the robot sees a “mouse” (tracked object) or recognizes a human face.

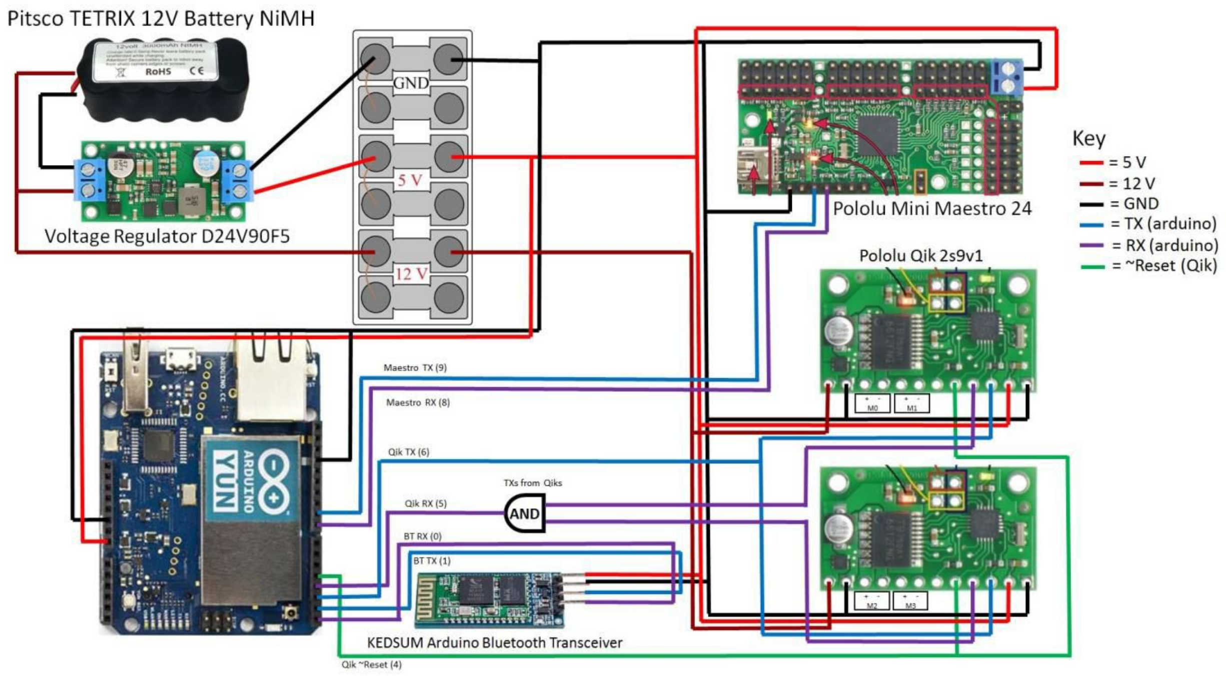
Project two added some complexity with the introduction of ROS into the project. While it might have been possible to move the servo control from Arduino to the Raspberry Pi (keeping the maestro servo control board), we decided to continue to use the Arduino. There is a ROS node to control the servos, but it used serial commands across a USB cable to communicate with the Arduino, which would actually execute calls to the servo board. We chose this, first of all, because the arduino allows for a form of parallel computing, where it can process older commands, and complete the actions associated with them, while the rest of the ROS code continued to work on other things. This allowed for pauses, sleeps, and other delays without halting the ROS node, as well as letting the ROS node continue to listen for newly published topics (The wheel control works in exactly the same way).

We did have some issues communicating between the Raspberry Pi running ROS and the Arduino. We initially were planning to use the GPIO (general purpose input/output) on the Raspberry Pi to communicate with the GPIO of the Arduino. However, we were having some trouble with the python GPIO library, and so were switched over to communicating serially between the two devices via a USB cable. This works well, as we send a single character over the cable to indicate a certain action (either with a servo or with the wheels), and that character can be interpreted on the Arduino sign. This is a fast and relatively error-resistant way to communicate across the two devices.

For the current servos, if the control board supports it, changing the speed of the servo movements might create more fluidity in the gestures. This will need to be experimented with later. Additionally, new animations will need to be created to match the new behaviors of the robot. If we can get the current servos working well, it would be nice to add addition servos to the robot to give it new features. At the moment, the ears are just a part of the face plate. If we had time, it would be nice to have more realistic cat ears that could move or rotate to show emotion. Also, another servo (or multiple servos) could be added to create a tail for the robot, which could also bring more emotion and expression to the robot.

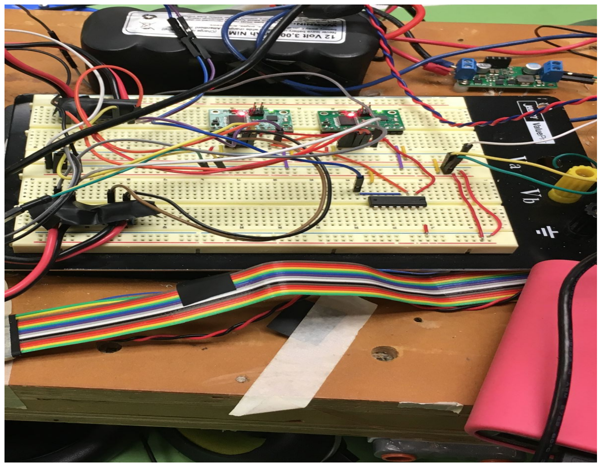
Robot Base (Electronics Board and Omni-Wheels)

Robot Electronics: The Robot base houses the Electronics of the board. The Schematics Blocks are shown below (although the Bluetooth capability was removed).



We have a 12V Battery that powers all the electronics and the motors. The wheels of the robot use 12V to run. We have a voltage regulator to step down the DC voltage to 5V that is used to power the rest of the components on the board.

Challenges: The NAND gate IC was damaged. Had to rewire the entire breadboard of connections between the Arduino, Pololus and the motors and replace the IC.



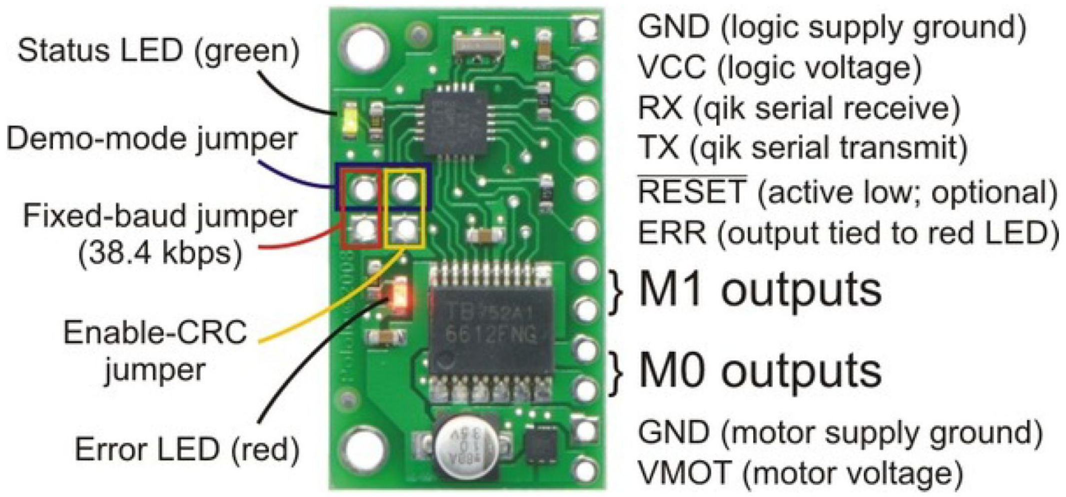
Next project: Transfer all the electronics on the breadboard to more reliable solderable general purpose PCBs.

Robot Wheels: The Robot uses 4 Ohmi Wheels. Omni wheels or poly wheels, similar to Mecanum wheels, are wheels with small discs around the circumference which are perpendicular to the turning direction. The effect is that the wheel can be driven with full force, but will also slide laterally with great ease.

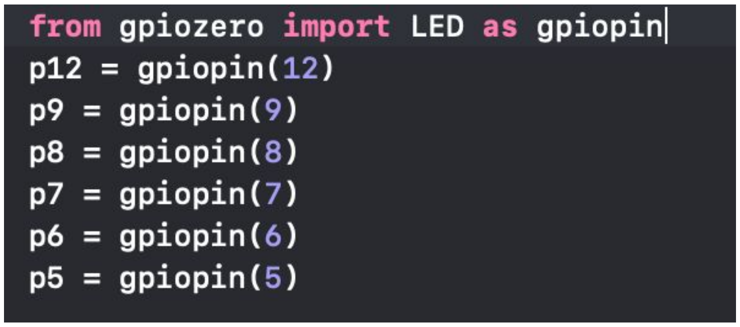
The robot wheels are controlled using two qik 2s9v1 to control the motors. It has the following features:

1. High-frequency PWM to eliminate switching-induced motor shaft hum or whine
2. A robust, high-speed communication protocol with user-configurable error condition response
3. Visible LEDs and a demo mode to help troubleshoot problematic installations
4. Reverse power protection on the motor supply (not on the logic supply).

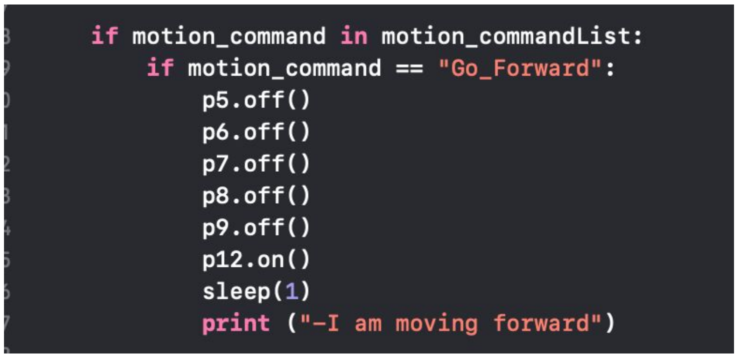
Connecting the Qik:



For project 2 we decided to use the raspberry pi in addition to the Arduino to run ROS node. At first we decided to use GPIO on raspberry pi to output a high or a low, which would be read as input on Arduino.
We used the gpiozero package to control the pins on our pi. Below is the example of the code snippet:



```
from gpiozero import LED as gpiopin
p12 = gpiopin(12)
p9 = gpiopin(9)
p8 = gpiopin(8)
p7 = gpiopin(7)
p6 = gpiopin(6)
p5 = gpiopin(5)
```



```
    if motion_command in motion_commandList:
        if motion_command == "Go_Forward":
            p5.off()
            p6.off()
            p7.off()
            p8.off()
            p9.off()
            p12.on()
            sleep(1)
            print ("-I am moving forward")
```

Unfortunately the above approach did not work.

So we decided the to use the serial communication for communicating between the Arduino and pi. We used the sample SerialEvent example code in Arduino. The pi will send a command string based on which the Arduino will take action and send commands to qik for wheel control. This approach took us some time to accomplish on as we had trouble receiving the string being sent from pi as we did not know that we had to send a /n character at the end of our output string in the ROS python code.

Below is the example of the code snippet for python ROS node:

```python
def callback0(data):
    global motion
    motion = data.data
    print('callback')
    navigation()
    #print(motion)

def navigation():
    global motion_cmd, motion
    motion_cmd = motion
    print("insde navigation" ,motion_cmd)
    if (motion_cmd == "Go_Forward"):
        print("FFFF")
        ser.write('f'+'\n')
        #rospy.sleep(1)
        print ("-I am moving forward")


    elif (motion_cmd == "Go_Backward"):

        print("BBBB")
        ser.write('b'+'\n')
        #rospy.sleep(1)
        print ("-I am moving backward")

    elif (motion_cmd == "Turn_Right"):
        print ("RRR")
        ser.write('r'+'\n')
        #rospy.sleep(1)
        print ("-I am Turning Right")
```

This approach enabled us to send a character to Arduino and control the wheels of the base. Below is the example of the code snippet for Arduino:

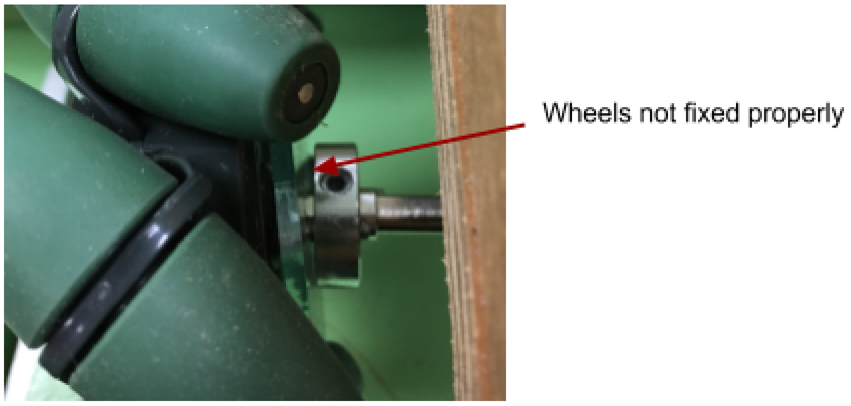
```
String inputString = "";          // a String to hold incoming data
bool stringComplete = false;  // whether the string is complete

void setup() {
  // initialize serial:
  Serial.begin(9600);
  // reserve 200 bytes for the inputString:
  inputString.reserve(200);
}

void loop() {
  // print the string when a newline arrives:
  if (stringComplete) {
    Serial.println(inputString);
    // clear the string:
    inputString = "";
    stringComplete = false;
  }
}

/*
  SerialEvent occurs whenever a new data comes in the hardware serial RX. This
  routine is run between each time loop() runs, so using delay inside loop can
  delay response. Multiple bytes of data may be available.
*/
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    inputString += inChar;
    // if the incoming character is a newline, set a flag so the main loop can
    // do something about it:
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

Challenges: The acrylic sheet holding the motor and the wheel was damaged. Just before the demo the acrylic sheet holding the wheels broke and the wheel came off.



ROS Networking

To properly integrate the ROS framework with the Schrodinger's Cat Robot, we decided that we needed to utilize Raspberry Pi's for all of our peripheral's nodes, and a PC to control computation and timing for the play. During the process of implementing ROS networking we encountered a few issues. The first was getting ROS working on enough Raspberry Pis. Eventually we got two working. One of the Pis used Ubuntu Mate instead of Raspbian and downloaded ROS using the Ubuntu steps we had experience using from Homework 3. The other was able to successfully combine the zip files, provided by Melih, into a single image, and upload onto the Pi. Once we had working Raspberry Pi's, the second issue we encountered was getting the PC and Raspberry Pi to sync properly. When we followed the instructions for testing the connection, we were able to ping the pi from the PC and read the list of topics from the PC, however when we tried to control the program running on the Raspberry Pi from our PC we experienced errors. Due to time constraints, we had to scrap our efforts to implement networking and simply run the program from a single Raspberry Pi. For the future, implementing the robot using ROS networking will allow the robot to be more powerful and functional.

Conclusion

While the robot we were working with has more degrees of freedom than many of the robots in the lab, Schrodinger's Cat is an old robot, and it shows in how difficult it is to work with the hardware. To start with, the omni wheels give it complete freedom to move in any direction, but also are badly attached to the base (leading to one breaking just before the final demo), and having so much torque that the robot is in danger of tipping over. The multitude of servos in the boy of the robot give it plenty of expressiveness, except that many of them are broken or produce only jittery movement. Not to mention, the idea of a full humanoid figure standing on a wheeled base does not line up with any of the usual ideas of a cat.

While we managed to get the robot working for this project, we spent much more time than we would have liked just fighting the hardware, and the awkward shape and weight distribution of the robot. In the future, this robot should be rebuilt before being used again by students, so that they do not have to fight these same issues in addition to whatever project requirements they might have.

Media Links

Videos of the robot can be found on the project GitHub page at:
https://github.com/cbrooks1/ECE578_Schrodinger_Cat/tree/master/Videos

Code Reference

All of the code for this project (for both the Raspberry Pi and the Arduino) can be found on the projects GitHub:
https://github.com/cbrooks1/ECE578_Schrodinger_Cat/tree/master

Index of Materials for Project 1

Raspberry Pi 3 - Used for text to speech, screen, and face/object detection - $35
Raspberry Pi Camera
XPT2046 - LCD Touch Screen : $35

References

Text to Speech
https://www.dexterindustries.com/howto/make-your-raspberry-pi-speak/
Melih's slides on Dialogflow and Amazon Polly

Robot Detection
http://www.theconstructsim.com/publish-image-stream-ros-kinetic-raspberry-pi/
https://github.com/UbiquityRobotics/raspicam_node
http://projectsfromtech.blogspot.com/2017/10/visual-object-recognition-in-ros-using.html

LCD Display/Face
https://gist.github.com/ronekko/dc3747211543165108b11073f929b85e
https://raspberrypi.stackexchange.com/questions/53931/why-is-my-raspberry-pi-3-display-not-fill
ing-the-screen/54074

Servo Control
Details on the Pololu Mini Maestro: https://www.pololu.com/product/1356/resources
Arduino Library for the Mini Maestro: https://github.com/pololu/maestro-arduino

Board Hardware
Details on the Pololu Qiks - https://www.pololu.com/product/1110
Arduino Library for the Pololu Qiks - https://github.com/pololu/qik-arduino