PSU ECE 510 AER Class Project

Tony Muilenburg, Hussein AlAmeer, Ben Huntsman

Robot Theater Comedy Sketch

Spring 2015

Contents

[Project summary 3](#_Toc422160987)

[Previous work 3](#_Toc422160988)

[Hardware setup 3](#_Toc422160989)

[Software environment 3](#_Toc422160990)

[Overview of Robots 5](#_Toc422160991)

[NXT Firmware and Code 7](#_Toc422160992)

[Catbot, Ratbot Overview 7](#_Toc422160993)

[Bohr/Einstein Overview: 8](#_Toc422160994)

[Theatrical Play Scenes 9](#_Toc422160995)

[Prolog 10](#_Toc422160996)

[Bohr/Einstein Hardware: 11](#_Toc422160997)

[Right Arm Design 11](#_Toc422160998)

[Right Arm Assembly 12](#_Toc422160999)

[Left Arm Design 13](#_Toc422161000)

[The Left Arm Assembly 15](#_Toc422161001)

[Maintainability 17](#_Toc422161002)

[Base 18](#_Toc422161003)

[Bohr software control 18](#_Toc422161004)

[Controlling Servo motors using Raspberry Pi 18](#_Toc422161005)

[Controlling DC motors using Raspberry Pi 21](#_Toc422161006)

[Bluetooth Communication with Central Computer for Robot Theater 21](#_Toc422161007)

[References 22](#_Toc422161008)

# Project summary

Many intelligent robots have been developed in Dr. Marek Perkowski’s lab over the years at PSU, but one of the most interesting challenges is to integrate them to the degree that they can work together, communicate, avoid collision, etc.  For this project we put together a theatrical comedy sketch that utilizes this integration.  A framework was developed to provide control of five robots providing control, voice synthesis, voice recognition, and computer vision for robot localization, face detection, gesture detection, etc.

# Previous work

While the framework for motion control and the voice synthesis was new, this project focused on integrating robots that were previously developed at PSU.  The five robots include:

1. Bohr/Einstein
2. Marie Curie
3. Schrodinger cat
4. Baby Schrodinger cat, or catbot
5. Ratbot – same implementation as catbot, just a different synthesized voice, and Bluetooth ID

Motivation for the play came from the desire to enable the “[Quantum conscious](https://drive.google.com/open?id=0Bw13WqGlAMkAYl9TOHpybGxxX00&authuser=0)” play.  We decided to put together a simple play that would use multiple robots that could work together, as well as assign a voice to each robot and perform a self-test for each of them.  Ideas for integration and integration came from the document titled:  “[Towards robot theater](https://drive.google.com/open?id=0Bw13WqGlAMkAazlrNmVCS1JKcVU&authuser=0)”.  Stage setup including computer vision came from the document titled: “[Vision system for Marie Curie](https://drive.google.com/open?id=0Bw13WqGlAMkAMW1FMnNNZ21SUTA&authuser=0)“.

# Hardware setup

We came to the conclusion that one of the most difficult aspects of getting a stable working theatrical presentation put together is that students would use their own computers, different operating systems, coding languages, etc.  When students completed the class, the next students would spend a significant amount of time recreating what was done before, and in many cases, solving problems that were already solved.  To reduce this pain for the future, a laptop was donated to the lab, locked to a bench, and set up with remote access so that it can be used off campus.   The laptop has all of the code used for the theatrical play as well as all of the programming environments needed to develop, compile, etc.  Other hardware includes a Microsoft HD webcam, two Kinect cameras, a microphone, a Bluetooth adapter, and speaker with an amplifier.

# Software environment

The main controller for the play was written in c# using visual studio 2013 (this IDE is free).  GUIs for control are written in other languages, but all work together easily because text files are used to communicate between them.  This allows for easier debug as well.  Figure 1 shows a screenshot of the controller.  To begin the play, the button titled “execute from batch file” is pressed.

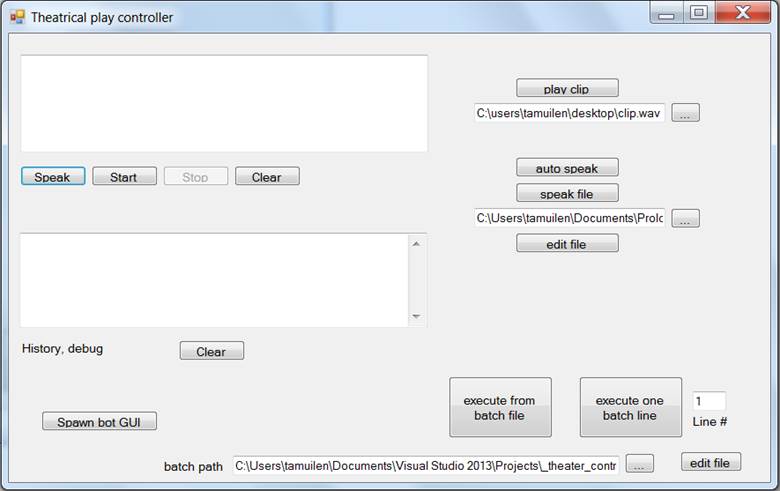


Figure 1: Theatrical plan controller GUI written in C# using visual studio

After the button is pressed, the controller will open the scene.txt file which has the commands the robots should perform.  Figure 2 shows an example of a file with some of the available commands.  The first line describes the expected format.  The first item should be the robot name, then the number of commands to execute, the type of command, the value, then the delay to wait in seconds after the commands is performed.  The line: Schrodinger; 1; say; “self test”; 2  will cause the Schrodinger cat voice to be loaded, say “self test”, then wait two seconds.  The move command below that with an argument of “f” causes the Schrodinger cat to drive forward for one second.  Farther down in the file, after the pause, is an example of some of the text from the play.  The three scenes will be described in another section below.

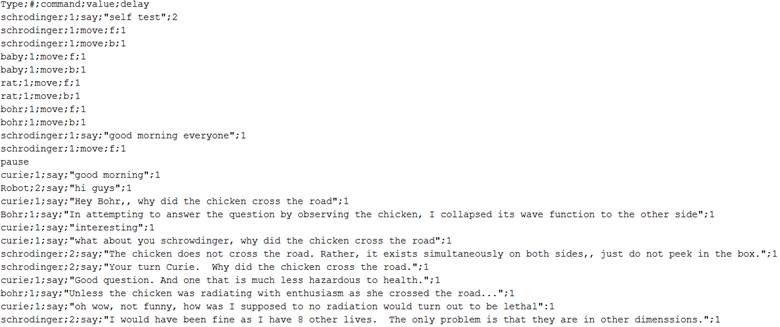


Figure 2: Example flow for the play

The theatrical play controller has buttons to test other features, and also a text box that prints out messages as the play progresses.  The start and stop buttons enable speech recognition.  The speak button uses speech synthesis so say whatever is in the textbox above the button.  Auto speak caused the program to synthesis and say any text that is saved in the text file (the location can be specified with the browse button).  Play clip demonstrates the code the can play audio files.  One example is a clip used in the play for a crowd laughing.

Tutorial videos for the development of each of the components were created and posted to:

<http://ece.pdx.edu/~muilenta/robotics>

Tutorials are also available for many of the other components used like OpenCV, Kinect computer vision, logging into the controller laptop, and an introduction to project goals.

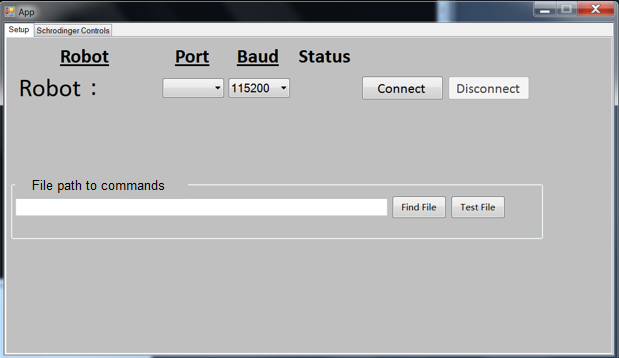
See the appendix for examples of code for each of the components.  The full code for the controller and each of the robots is posted to the project website at:

<http://ece.pdx.edu/~muilenta/theater>

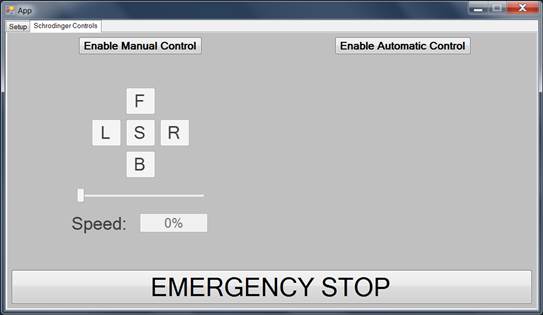
# Overview of Robots

Three of the robots use a GUI very similar to that developed for the original CatBot robot.  For the full CatBot documentation along with videos, see: <http://ece.pdx.edu/~muilenta/catbot>   This modified GUI can have multiple instances spawned, and is generic so it can talk with any of the three robots: CatBot, RatBot, and Bohr.

The user interface is split up into two tabs.  The first allows the user select the Bluetooth device that is to be used, and configure it.  While some laptops come with integrated Bluetooth, the laptop used for this did not, so a USB dongle was added which added it.  After selecting the Bluetooth port, the connect button will enable communication with the NXT equipped robot that has been flashed with the robotc firmware created for this project.  The theater controller creates a text file to pass location data to this robot control code.  The path to the file the computer vision code generates should be added to the textbox using the find file button which pops up a file browse dialog window.  The “Test File” button should then be used to ensure that the path is valid and file permissions are ok (permission to read the file is granted).



The second tab has buttons to allow for manual or automatic control. In manual mode, the robot can be turned, or orders can be given for it to go forward or backward.  A speed control is available, as well as an emergency stop (see below).

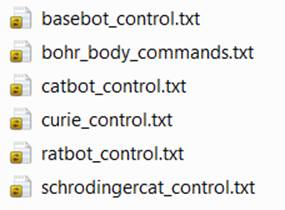


In manual mode, the selected robot will move forward, left, back, and right depending on buttons pressed.  In automatic control, the robot will respond to commands given to it by the theater controller by reading from the text file selected in the setup tab.

In order for all three robots to work, each should have a text file for commands, and the correct serial port should be selected for Bluetooth control of the appropriate Lego NXT controller.

An important distinction between this controller GUI and the original catbot controller, is that that the text file here is looking for simple characters to turn left, stop, go forward, etc.  whereas the original catbot GUI looked for coordinates from a computer vision program that ran at the same time and printed coordinates to a file.

Control of the Schrodinger cat and Marie Curie robots are controlled using text files as well.  The theater controller writes requested commands to the appropriate control files. Below is the list of text files.



Note that two text files are used for Bohr as there are separate controllers for the driving base (BaseBot) and the arm and head movement that is controlled with the bohr\_body\_commands file.

In addition to being able to drive and turn, the Bohr robot can perform many other movements which will be outlined below.

# NXT Firmware and Code

The commands sent via Bluetooth from the GUIs to the NXT firmware are intercepted by the NXT controller, and read using a robotc program called catbot.c   Visit the theater website for the full code.  Simple characters like f, b, l, r, and s are sent to the controller.  The ‘f’ character turns on both motors in the forward direction, and ‘s’ stops movement.  If r is sent, one wheel will move forward, and the other backward, for example.

A one year license of RobotC was purchased and installed on the theater control laptop, so changes can be made to the NXT code.

# Catbot, Ratbot Overview

The original Catbot and Ratbot  robots were built using First Lego League kits.



This kit is described on Lego’s website at this link: [9797](https://education.lego.com/en-us/lego-education-product-database/mindstorms/9797-lego-mindstorms-education-base-set).  Instructions for assembling the robot can be found here: [robot base instructions](https://education.lego.com/en-us/lesi/support/product-support/mindstorms-education-nxt/nxt-base-set-9797/building-instructions).

To simplify the design, 3d printed brackets were designed and printed.  

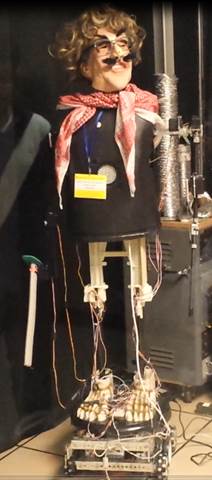
See thingiverse for the cad files:

<http://www.thingiverse.com/thing:823874>

See the [theater website](http://ece.pdx.edu/~muilenta/theater) for the full software

# Bohr/Einstein Overview:

The Bohr robot has two functional arms, a moving head, and is mounted to a driving base.  The base is controlled with the same interface the catbot and ratbot use via Bluetooth and an NXT microcontroller.  The head and arms are controlled using a raspberry pi via wifi.  The Bohr sections below have the full hardware and software description which can also be found on the theater website.



# Schrodinger Cat

Major updates were made to the Schrodinger cat in 2015. It now uses an Arduino controller and Bluetooth module rather than an NXT for control. The base can drive around, and the cat has movement of arms, legs, and the mouth using rubber bands.



# Marie Curie

Marie Curie has the ability to bend over at the waist, kick a drum with her legs, and move her arms. The left arm is a 3d printed inmoov robot arm which is documented at <http://inmoov.fr> Control for the robot is achieved with an Arduino board and Pololu for servos.



# Theatrical Play Scenes

The play is composed of a self -test and three scenes.

Scene 0: Self test

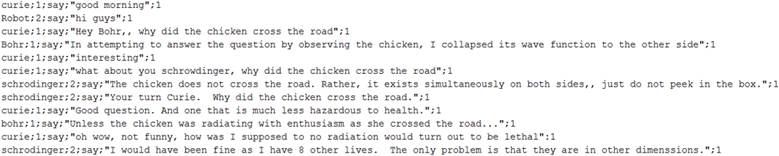
At the begging of the play, each movement of each robot is tested, as well as the voice for each of the robots.  This scene can be skipped if everything has been verified working in advance

Scene 1: Why did the chicken cross the road?

This scene was developed as pre-cursor to the [Quantum conscious play](https://drive.google.com/open?id=0Bw13WqGlAMkAYl9TOHpybGxxX00&authuser=0) developed by Dr. Perkowki.  In this play, we want to do some simple testing of speech, movement, location tracking, etc.  The five robots each go through the some of the jokes outlined online at this site:

<https://www.physics.harvard.edu/academics/undergrad/chickenroad>

The jokes include:



Scene 2: Exercise

In these scenes, the robots decide to do some exercises, but when some of the robots cannot complete some of the movements, they get competitive and start teasing each other. Curie is not able to stand on her feet, so when she is teased, she tells them that she is 3d printing another body. They then uncover the plot that she used machine learning to make money on the stock market, and she hired Elon Musk to build her a rocket to go to Mars. She does not like how warm and polluted our planet is becoming.

Scene 3: Prolog

In the third scene, we use the prolog code we developed to show the ability to use reasoning by analogy. For example, some robots will not like others because of the relationship and attitude of other robots. For example, Schrodinger cat should not like anyone that dislikes baby the baby cat.

The full flow is outlined in the scenes.txt file. This is the file the theater controller should use to run the production. See the [theater website](http://ece.pdx.edu/~muilenta/theater) for a video of the first portion of the play.

# Prolog

We have been studying prolog, a powerful high-level programming language. For the current robot theater consisting of 5 robots we wanted to model the robots’ behavior and provide reasoning on why the robots do specific actions. Prolog provides us this luxury.

The prolog program we have written is based on the 5 different robots in the theater. There are 2 humanoid robots and 3 animal robots. The humanoid robots are Bohr and Curie. The animal robots are Schrodinger’s cat, a baby cat, and a rat.

To illustrate the power to Prolog’s recursive searching of a solution our Prolog logic can be boiled down to who dislikes whom. Then we have who likes whom based on the logic of ‘the enemy of my enemy is my friend’.

First we define what kind the robots are. We then added an interesting relationship for the animal robots. First we decided that Schrodinger’s cat is the parent of the baby cat. We then classify predators and preys and then say predators hunt preys. Finally we add a hate relationship to base who dislikes whom.

Here is how we define the ‘dislike’ relationship:

dislikes(PersonA, PersonB) :- hates(PersonB, PersonA)

dislikes(PersonA, PersonB) :- hunts(PersonB, PersonA)

dislikes(PersonA, PersonB) :- dislikes(PersonC, PersonB), parent(PersonC, PersonA)

Next we define the ‘like’ relationship based on “the enemy of my enemy if my friend”:

likes(PersonA, PersonB) :- dislikes(PersonC, PersonB), dislikes(PersonA, PersonC), not(hates(PersonA, PersonB))

Note the use of the ‘not’. We need to make sure that Person A does not hate Person B as that might still yield Person A liking Person B while hating each other just because they both dislike a common individual.

Our prolog program prints to a file of the results with its reasoning. Below is the output of the file based on the prolog program we wrote. This can then be read by our main theater’s program to adjust each robot’s behavior toward another robot based on its feeling.

schrodi dislikes curie because curie hates schrodi

bohr dislikes schrodi because schrodi hates bohr

curie dislikes bohr because bohr hates curie

rat dislikes babycat because babycat hunts rat

rat dislikes schrodi because schrodi hunts rat

schrodi dislikes curie because curie hates schrodi

babycat dislikes curie because schrodi is the parent of babycat and schrodi dislikes curie

schrodi dislikes curie because curie hates schrodi

bohr dislikes schrodi because schrodi hates bohr

rat dislikes schrodi because schrodi hunts rat

rat likes curie because curie is the enemy of schrodi and rat is the enemy of schrodi

# 

# Bohr/Einstein Hardware:

## Right Arm Design

The arm has been designed using Lynxmotion aluminum channels and servo brackets. Lynxmotion channels were used as they are fairly light material. Plus they are come pre-cut and drilled. Using pre-cut and drilled materials makes assembly easy using the available lab equipment.

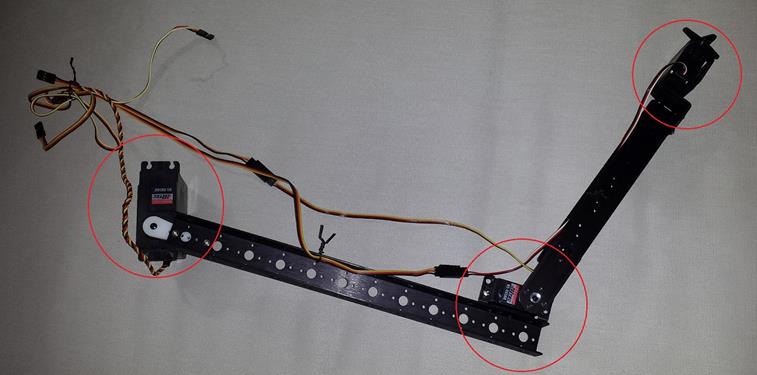
Figure 1: Various Lynxmotion Brackets used for designThe arm has 3 degrees of freedom, shoulder up and down, arm up and down and wrist rotation. According to our calculation, the total length of arm, which approximately matches a human body’s arm proportions, will be:

In addition to that, the total arm weight should be:

Figure 2: Hitech HS-805BB mega servoThe servo motor used to hold the arm and do the shoulder up and down motion is a Hitec HS-805BB mega servo, which has stall torque of at 5V. This means it can hold 2000g for 10 cm or 400g for 50cm, and for a basic arm design it will suffice to move the arm forward and backward. It also provides makes upgrading the arms fairly easy if ones desires that.

## Right Arm Assembly

The arm was assembly is fairly simple, because the Lynxmotion channels are already cut and drilled, and they provided all the needed screw, nuts and washers. The shorter aluminum channel was used for the forearm because only a small servo is used to connect the arm to the forearm, and this is going to reduce the load on the small joint servo. A tiny servo at the end of the forearm to connect any item, or in the future can be used to connect an actual hand. Currently it holds a sword from the previous project. A large servo (HS-805BB) is used to attach the arm to the robot’s body. Below is a picture that shows how each servo is connected

Figure 3: Right Arm Assembly with Servo’s directly attached

## Left Arm Design

Compared to the right arm, the left arm was designed from the beginning to be a much more superior arm whilst making as light as possible. Many of the components seen below can be found from the website servocity.com. They specialize in servos and robot related items. They even design and sell custom made brackets for all different kinds of servos.

Figure 4: Servo with a servo block from servocity.com

One of the problems in the right arm is overheating of the shoulder’s servo after a short period of time due to the weight of the entire arm it holding. Fortunately, a solution for this problem was found at servocity.com when designing the left arm. Servocity.com designs a servo block that is used for increasing servo’s load-bearing capabilities by helping to isolate the lateral load from the servo spline and case. The extreme versatility of the servo blocks allows users to create complex, extremely rigid structures with ease using standard Hitec servos available at the lab.

Figure 5: Carbon Fiber tubes used in designUsing the servo blocks allows the use tubes instead of aluminum channels, as can be seen in the right arm. Tubes are much lighter and stronger than aluminum channels, and are much easier to work with. Since the aim was to make it very light carbon fiber tubes was used. They only weight 25 grams per tube. They should be lighter than other tubes.

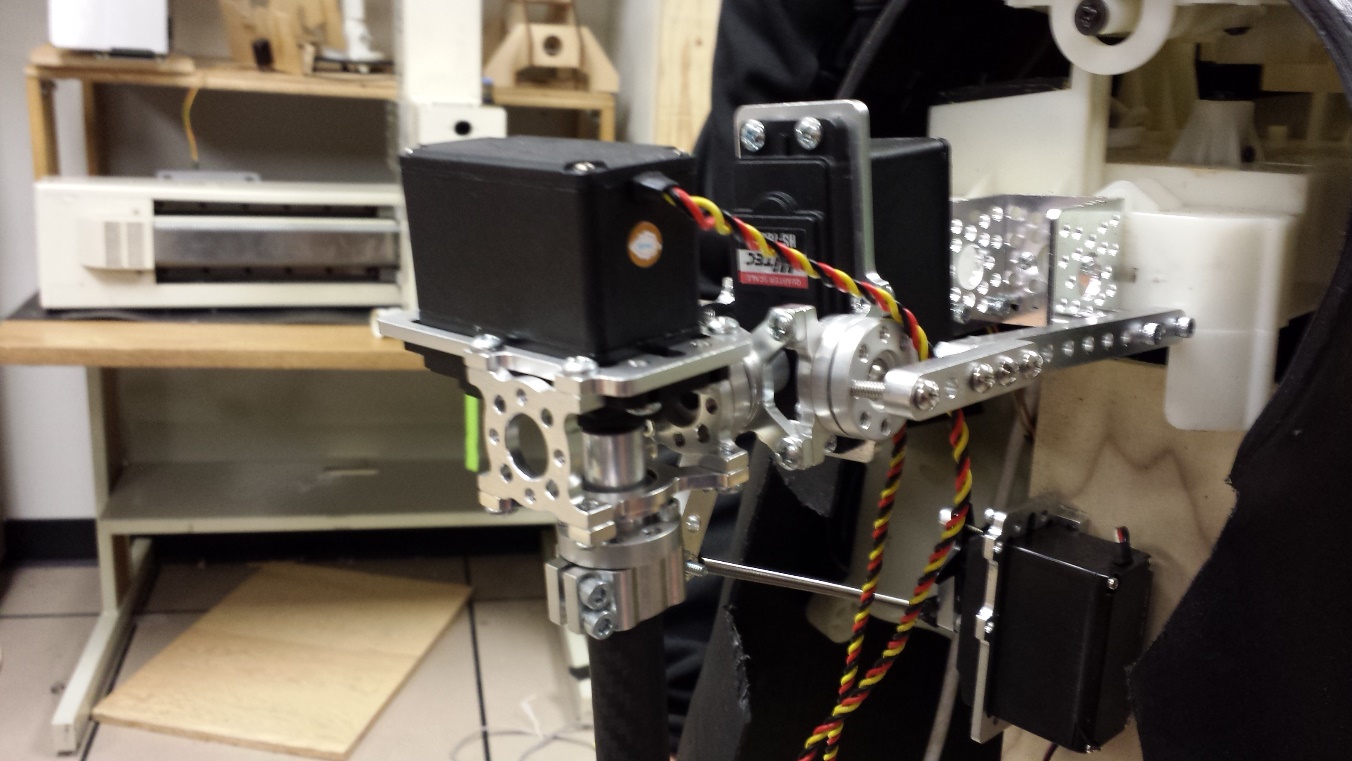
As we are using servo blocks to reduce the load on the servos and carbon fiber tubes to reduce weight, it is possible now to design an arm with more degrees of freedom. After contacting servocity.com’s technical support department to ask for arm designs suggestions, they pointed to YouTube video that shows an arm design with four degrees of freedom using their servo blocks. That design was adopted for the left arm and it was modified it for the robot as can be seen later.

Compared to the right arm, the left arm’s weight load is almost negligible because of the servo blocks used to reduce the weight load on the shoulder servo which is holding the entire arm’s weight. The weight was calculated for the entire arm just for the record.

(Hitec HS 755HB) + 50 (2 x carbon fiber tubes) + 102 (TS-75) + 88(2 x Servo Blocks) + 8.5 (Servo Bracket) + 11.5 (Joint Swivel) = 370 g

Although the Hitec HS 755HB servo stall torque is rated at 360g for 30 cm, the servo should be able to move the entire arm despite the fact that the arm’s length is twice what it could handle because the servo block is going to assist in supporting the weight stress on it.

## The Left Arm Assembly

Figure 6: Shoulder assembly with 2 servo blockThe left arm’s assembly is a little bit harder than the right arm as there are more components to assemble. First a support board inside the robot’s body had to be built to hold the servo that is going to move the entire arm inside and outside. Two servo blocks were attached together to form the should, and this shoulder is going to mover the arm up and down, left and right, just like a human’s shoulder.

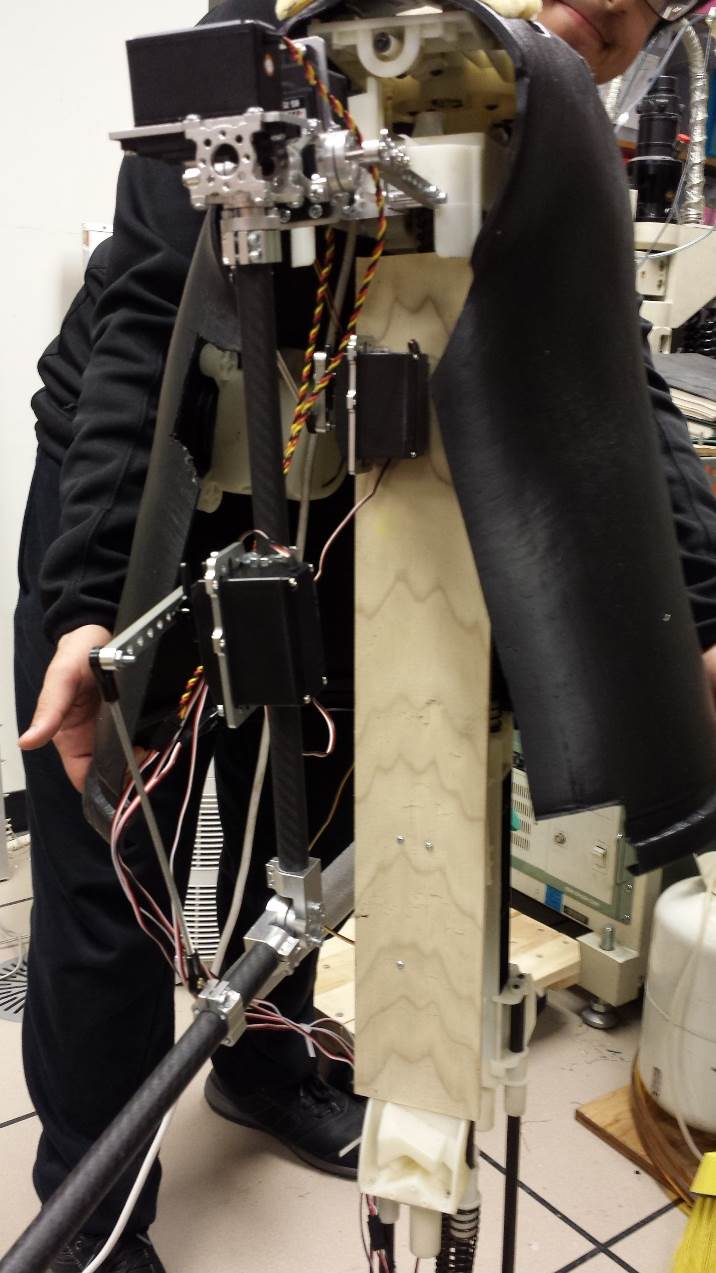
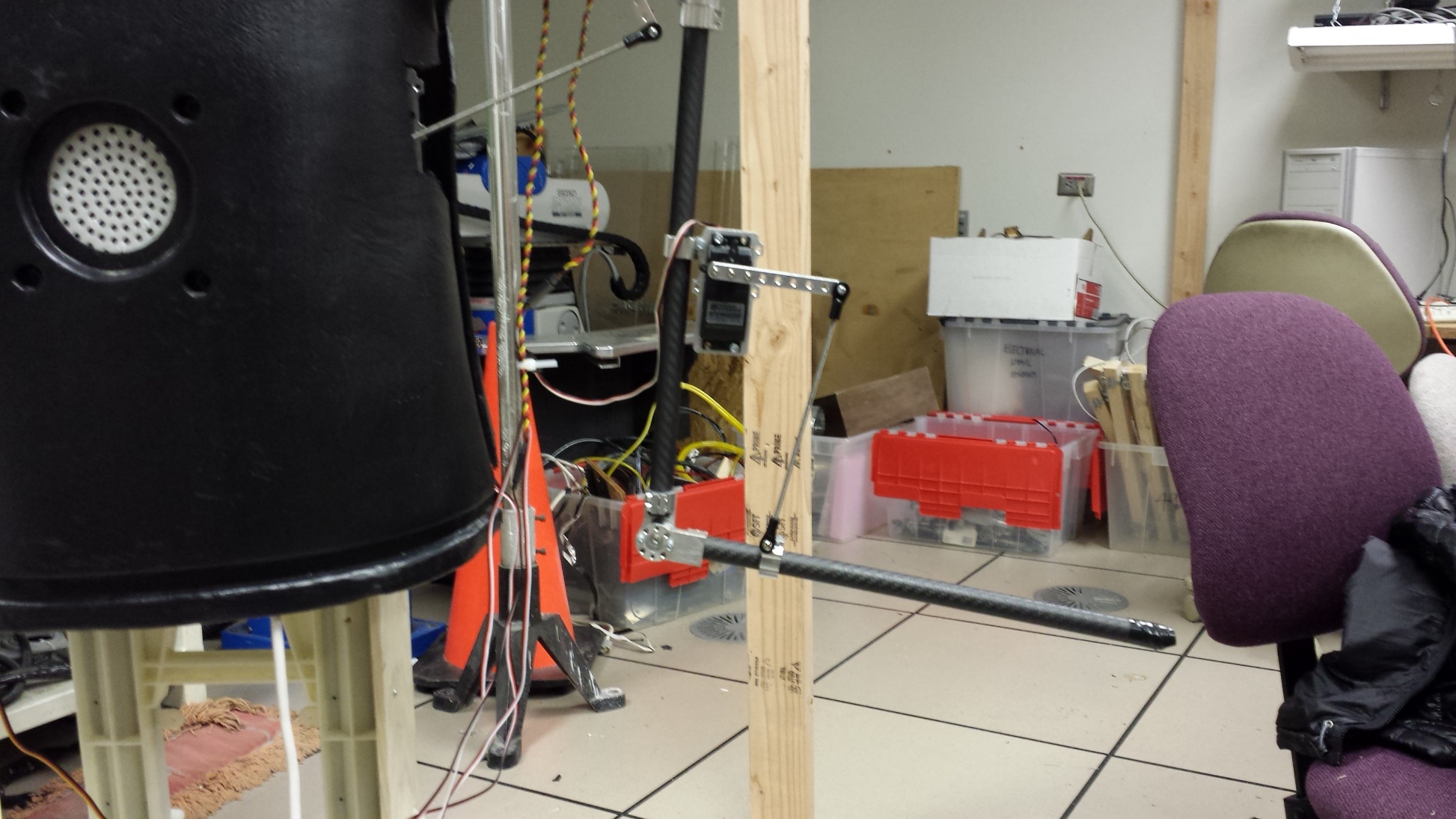
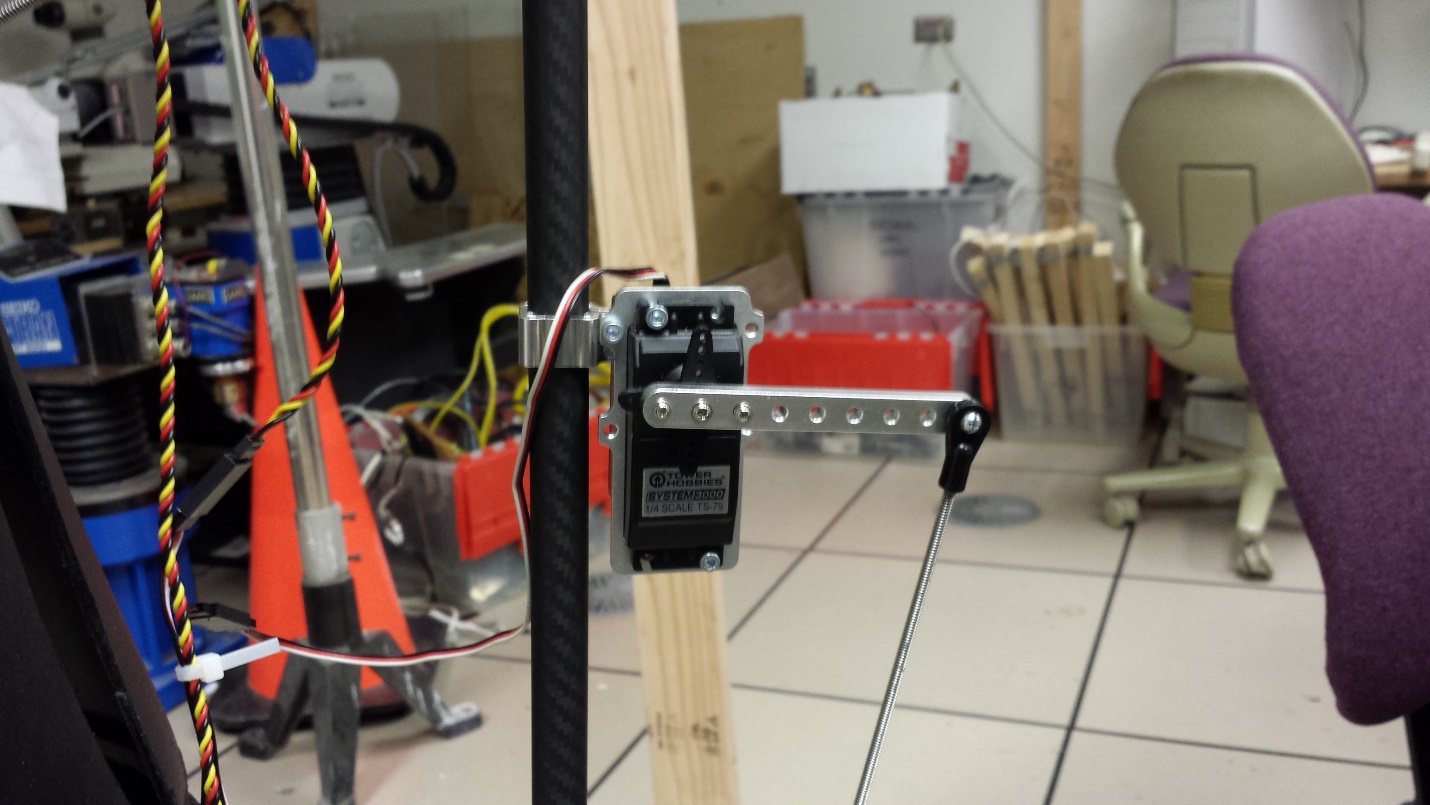


Figure 7: Support board to lift the arm

Figure 9: Left’s arm’s joint designFigure 8: Tube clamps and swivel jointsTube clamps and swivel joints from ServoCity’s website were used to connect the carbon fiber tubes to form a human-like arm and forearm with joints.  Another tube clamp and servo bracket was used for the upper arm to rise and lower the forearm. All the parts ordered from ServoCity come disassembled, so they had to be assembled to build the arm.

## Maintainability

The current design for both arms allow easy access to the servos. The right arm has three servos, a shoulder, an elbow, and a wrist. Each servo can be removed and replaced but since they are attached directly to the beams make sure to keep track of the location of the screws. Take photos or use current photos in this document to assist in disassembling and reassembling of the arms.

Figure 10: Remember screw locations and the shaft’s positionFor the left arm it should be easier to replace the servos since they are housed in their own brackets with bearings attached. Most importantly remember not to lose the bearings themselves during disassembly. Make sure where the screws are attached and also note the shafts position of the old servo and make sure it is the same when attaching new servos.

Note that the joint servo on the right arm has been stuttering sometimes while moving so it may need to be replaced soon.

## Base

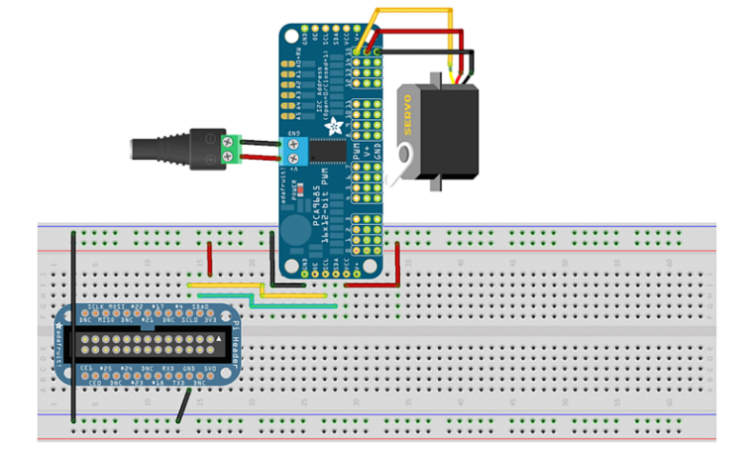
A base was added to Bohr that was a standalone robot previously representing Schrödinger’s cat. It uses two servos with PID control attached to OmniWheels as can be seen below:

The servos are attached to a Hitechnic controller that is then attached to an NXT controller. This made it easy to integrate previous work done to control robots with an NXT using RobotC. New firmware was flashed onto the NXT board and in RobotC the parameters were set to recognize the Hitechnic controller with PID to be able to use the pre-existing code to control the robot. See the other bots documents used in Winter 2015.

## Bohr software control

## Controlling Servo motors using Raspberry Pi

A Raspberry Pi is used as the controller for the body of the robot. The raspberry pi allows you to control servo and DC motors while using different programming languages and allows you attach a plethora of accessories to enhance the robot or have it be internet connected.

Figure 11: Attaching the Servo Driver to the Pi cobblerTo control the various pwm servo motors on the robot a servo driver controller is needed because Raspberry Pi has only one PWM output and cannot handle the power load those servos need. Adafruit  has a 16-Channel 12-bit PWM/Servo Driver that allows control of up to 16 servos using the Raspberry Pi with a cobbler breakout**, an** external power supply and a breadboard. Hooking up the Adafruit 16-Channel 12-bit PWM/Servo Driver to the Pi is very simple as shown in picture, and full tutorial explaining how to use the servo driver including a python code can be found here <http://learn.adafruit.com/adafruit-16-channel-servo-driver-with-raspberry-pi/overview>.

The sample code that is provided in Adafruit’s tutorial does not show how to control the servo speed, so another code was obtained from Adafruit’s forum explaining how to control the servo speed in python using Adafruit’s own library.

Figure 12: Sample code to control the Servo’s using Pi#!/usr/bin/python

from Adafruit_PWM_Servo_Driver import PWM
import time

# ===========================================================================
# Example Code
# ===========================================================================

# Initialise the PWM device using the default address
pwm = PWM(0x40, debug=True)

servoMin  = 150  # Min pulse length out of 4096
servoMax  = 600  # Max pulse length out of 4096
sweepTime = 30   # We want the sweep to last 30 seconds
stepSize  = 10   # We'll change the pulse by a small amount each time

steps     = (servoMax - servoMin) / stepSize    # Calculate the number of steps we'll need
stepTime  = sweepTime / float( steps )          # Calculate the delay between changes

#  The calculation for stepTime looks a bit strange because of the way Python
#  does math.  If you divide an integer by another integer, it will round the
#  result down to the nearest integer.  If you want a floating point value, you
#  need to include a floating point number in the calculation.  The `float()`
#  function forces the intepreter to treat `steps` as a floating point value.

pwm.setPWMFreq(60)                                # Set frequency to 60 Hz

for t in range( 0, 2*steps ):                     # Make the cycle long enough for a sweep up and back
    if t < steps:
        n = t                                     # For the first half of the cycle, count up
    else:
        n = (2 * steps) - t                       # For the second half of the cycle, count down

    pwm.setPWM( 0, 0, servoMin + (n * stepSize) ) # Make the pulse width depend on the count
    time.sleep( stepTime )                        # Wait before taking the next step


## Controlling DC motors using Raspberry Pi

The robot has a couple of DC motor’s to control the head, jaw, and chest/stomach. To control those DC motors using Raspberry Pi a L293D Quadruple Half-H Drivers IC is needed along with the existing cobbler breakout and breadboard. The L293D chip is capable of controlling two DC motors in different directions, and hooking up the L293D is simple. A comprehensive tutorial is provided from Adafruit website here <http://learn.adafruit.com/adafruit-raspberry-pi-lesson-9-controlling-a-dc-motor/overview>.

For the whole’s robot’s body one cobbler cable is used to run the Adafruit servo driver and two L293D DC motors, because each chip can control only two DC motors, and the robot has three DC motors. I The Adafruit servo driver and L293D chips have been combined on one breadboard through one cobbler to the Raspberry Pi obtaining power from a 5V 10A power supply, which is not sufficient for the project. As there are three DC motors, they need more than 10V to work properly. We suggest using a separate power supply for the servo driver and the L293D chip to reduce the load on the circuit, in our case a 9V battery was used.

## Bluetooth Communication with Central Computer for Robot Theater

Bluetooth is the main communication tool currently used for all the various robots so it was also added to Bohr. To make interfacing easy a UART to Bluetooth module was used on the Raspberry Pi. Bluetooth serial communication was also used for the rest for the rest of the robots.

To send or receive information on the raspberry pi, an existing PySerial library was used. It automatically sets up everything after providing the serial parameters of the UART device. Below is a code snippet of the current configuration on the raspberry pi:

# configure the serial connections (the parameters differs on the device you are connecting to)

ser **=** serial**.**Serial**(**

      port**=**'/dev/ttyAMA0'**,**

      baudrate**=**115200**,**

      bytesize**=**serial**.**EIGHTBITS**,**

      parity**=**serial**.**PARITY\_NONE**,**

      stopbits**=**serial**.**STOPBITS\_ONE**)**

The pi currently is designed to only receive data. To receive the data from the UART the following function call is used ‘out **=** ser**.**readline**()’**. The received data should be a command. The current commands that can be issues to Bohr’s body is as follows:

point      <Time in s, accepting Floating Point>  
walk       <# of steps, accepts Integers>  
dance  
wave  
faceleft  
faceright  
faceno (face shakes back and forth signifying no)  
exit

To send commands to Bohr, the central computer was paired with the UART Bluetooth module. The pairing code is 1234. The device should now show up under one of the COM ports in Windows. A python script was written for the central computer that read from a file.

# References

[Towards robot theater](https://drive.google.com/open?id=0Bw13WqGlAMkAazlrNmVCS1JKcVU&authuser=0)

[Vision system for Marie Curie](https://drive.google.com/open?id=0Bw13WqGlAMkAMW1FMnNNZ21SUTA&authuser=0)

[Quantum conscious play](https://drive.google.com/open?id=0Bw13WqGlAMkAYl9TOHpybGxxX00&authuser=0)

<https://www.physics.harvard.edu/academics/undergrad/chickenroad>

Catbot site: <http://ece.pdx.edu/~muilenta/catbot>

Theater site: <http://ece.pdx.edu/~muilenta/theater>

Code tutorials: <http://ece.pdx.edu/~muilenta/robotics>

Dr. Perkowki’s site: <http://ece.pdx.edu/~mperwoks> (includes more plays)

[Inmoov 3d printed head progress report](https://drive.google.com/open?id=0Bw13WqGlAMkAcm5hYVBGbGFaVVk&authuser=0)

Full inmoov robot: <http://inmoov.fr>

[Google drive repository](https://drive.google.com/open?id=0Bw13WqGlAMkAflltb2dPNW9scjdPaTE1ZzJTaXliY2Q0aElFU1VpVElVc0Z5ZFV6NkZMbWc&authuser=0) used for the project

[Schrodinger cat documentation (2010)](http://web.cecs.pdx.edu/~mperkows/CLASS_479/PROJECTS_FOR_GUIDEBOT/Chris_Forsstrom_ECE578_Fall2010_Schroedinger%20Cat.pdf) (the report from this year will be posted to the theater site when I receive it from Sean).