COUNTESS QUANTA

ARM DESIGN AND SOFTWARE CONTROL

Saroj Bardewa
ECE:479/579- Winter 2016
Dr. Marek Perkowski

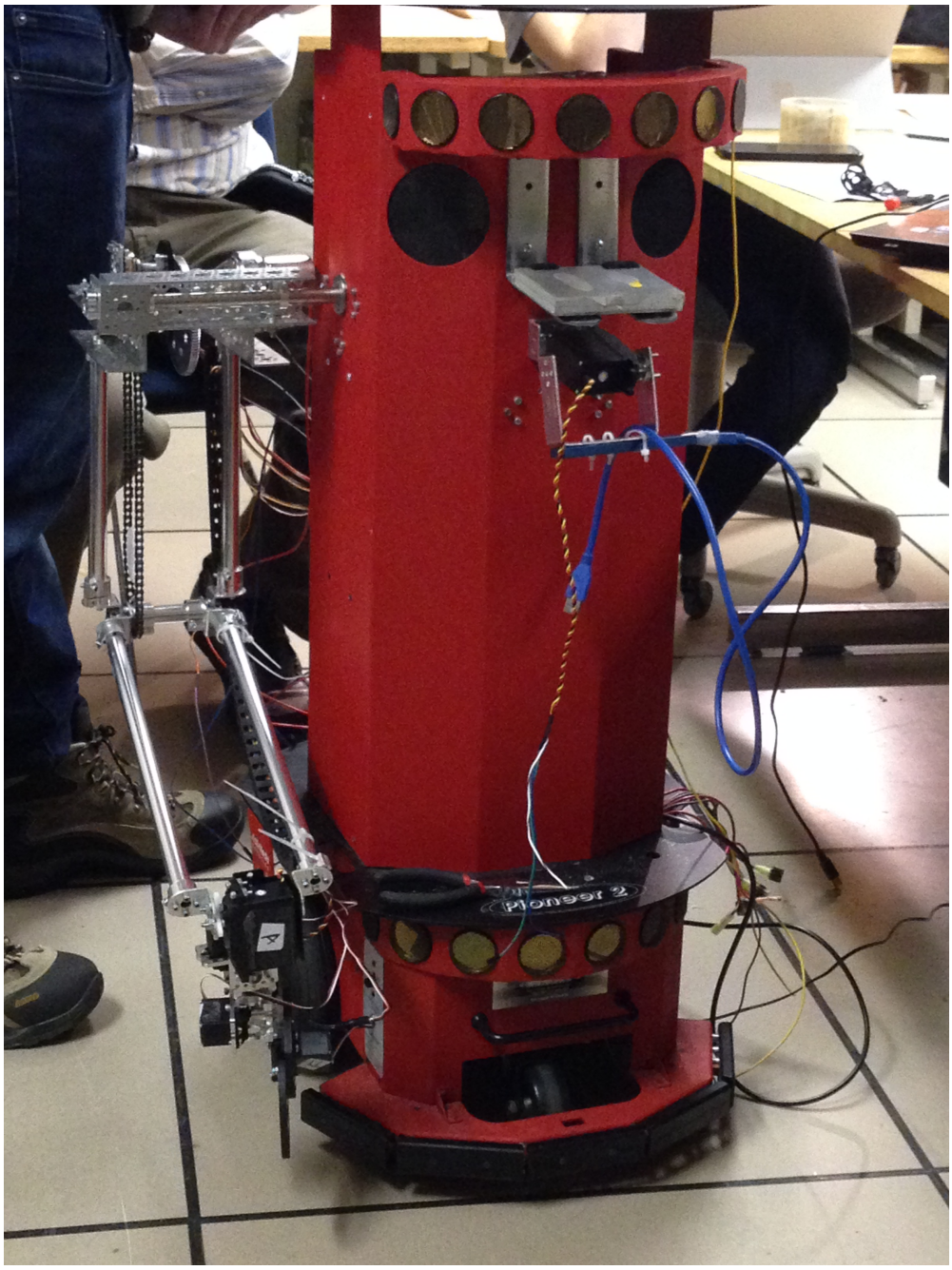


Table of Content

Introduction

The purpose of this document is to provide the detail of Countess Quanta arm design and control system performed in winter of 2016. This document mostly focuses on the software control system of arm design. In addition, the motivation of arm design, it's current state and the future modifications will be covered in detail.

Overview of Countess Quanta

Countess Quanta is a mobile robot. It is designed to give the tour of Maseeh College to people, and provide information about PSU campus. Its base consists of wheels which is a Pioneer robot, and can be controlled and navigated independently. The past functionality of the Quanta included speech recognition, human interaction, autonomous locomotion, playing an instrument and object tracking.

With the recent modification of its right arm, Countess Quanta is equipped to pick a small objects such as soda cans and medicine bottle from the floor. However, Quanta still needs full hardware and software integration to be able to achieve that goal.

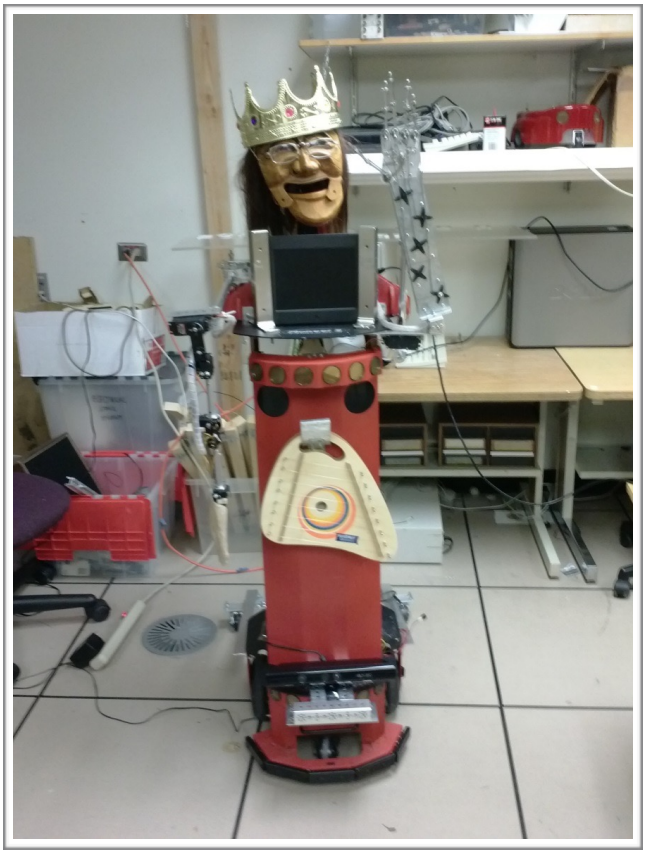


Figure 1: *Countess Quanta in the past consisted of two mechanical arms, mobile pioneer robot base, vision with Kinect, and speech recognition*

Motivation of Arm Design—Loki

The motivation of Countess Quanta arm modification sprang from a robot named Loki[1]. A hobby design of Intel engineer,Dave Shinsel, Loki is also a mobile robot which has more advanced features than Countess Quanta. Loki has more sophisticated speech recognition, object tracking, navigation and human-like interactive features.For example, with its well designed arms,Loki is able to pick an object from the floor. Interestingly, the arm design of Loki is fairly sophisticated. This robot is able to interact with humans, move independently, and handshake with an individual.

In fact, Countess Quanta's arm functionality is comparable to Loki's. The current arm design and control system of Quanta is the first step towards the goal of making Quanta as functional as Loki's. The end goal is to be able to incorporate the arm features and capabilities of Loki, and more. Thus, the first endeavor has been to be able to design an arm that can grab an object from the floor at a rather controlled environment.

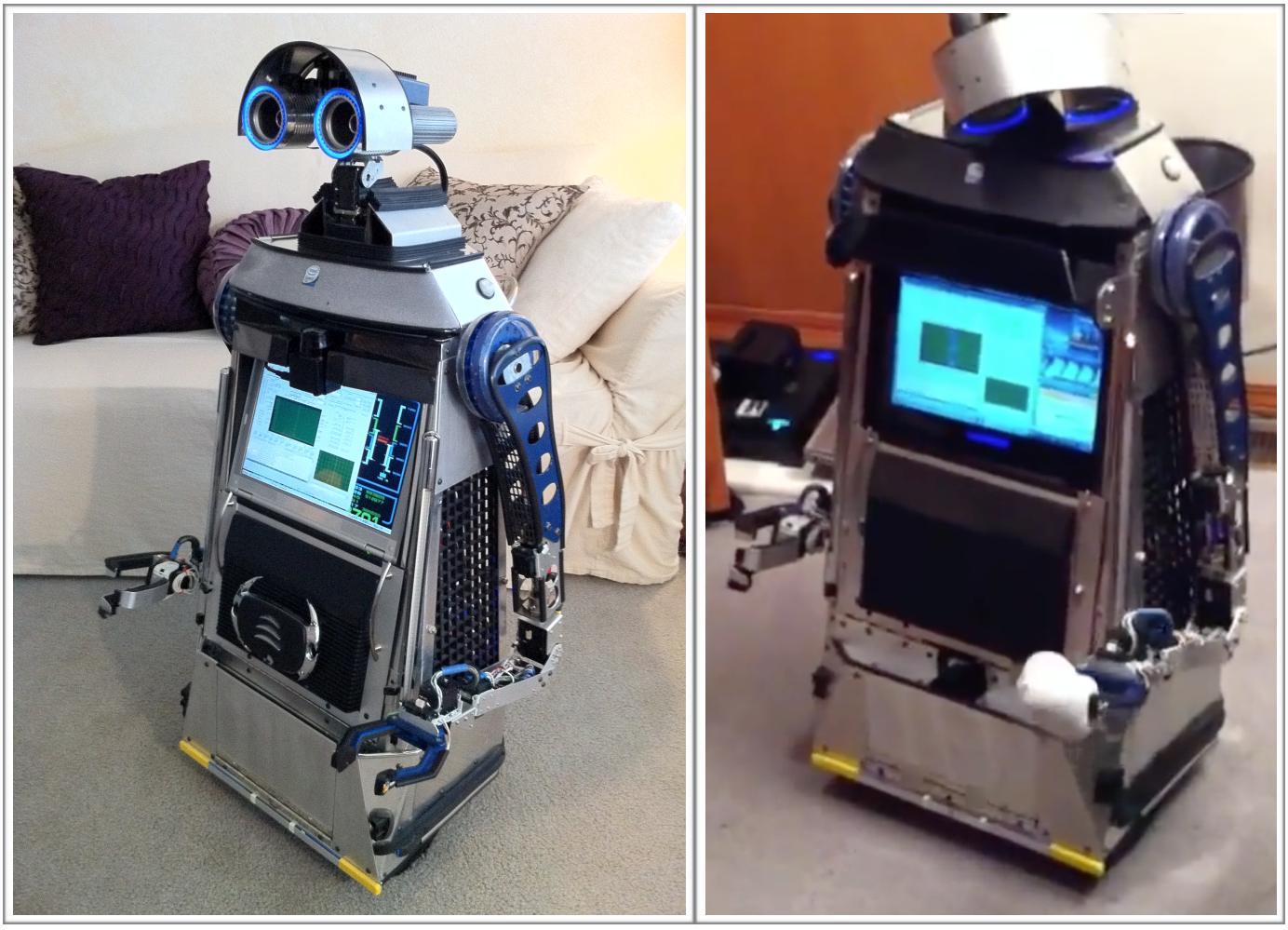


Figure 2: *The complete design of Loki robot (Left) and Loki picking an object (Right)*

Countess Quanta Arm Design

There has been couple of modifications made on Countess Quanta arm. The current arm is designed of light-hollow steel framework with a plastic gripper as fingers . This document very briefly introduces the arm design, just enough to aid how the software is able to control the arm motion. The more detail of the arm design is covered in supplementary documents.

The mechanical right arm design consists of three parts: shoulder, upper arm and lower arm. The short construction and functionality of the arm are as follows;

Shoulder

The shoulder joint consists of a motor, a gears and a potentiometer attached to two other gears. The detail of the shoulder motion will be covered later. The motor and gear is connected to a metal rod, which is protruded outside the body of Quanta. As the motor rotates, the metal rod attached to the gear moves as well. Furthermore, the rotational motion of the rod rotates the arm connected to it that in turns controls the position of shoulder.

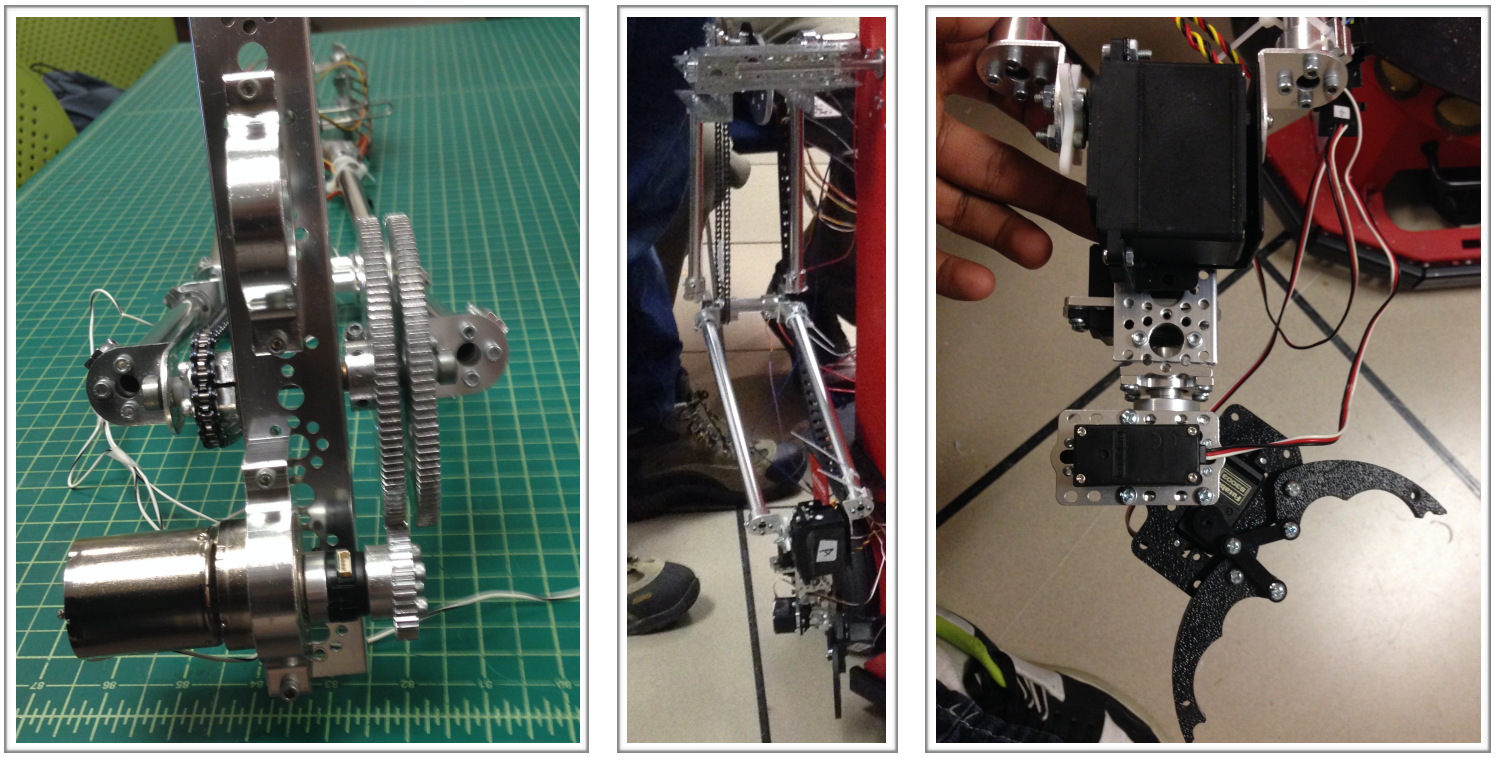


Figure 3: *Modified shoulder (left), upper arm (middle) and lower arm(right) of Countess Quanta*

Upper Arm

The upper arm of Quanta consists of a motor, gears and potentiometer. Also, there is a chain that contains the upper arm with the lower arm to be able to control the motion of the arm. The potentiometer provides feedback to the motor such that the arm can be moved to the user-requested position.

Lower arm

The lower arm consists of four independent servos that commonly provide five degrees-of-motions. These servos are placed in such a way that their motions provide human wrist-like behavior: rotation, up and down, side to side, and grabbing functionalities. The shape of the gripper at the tip of the arm is such that it allows the robot to grab a small object , for instance, bottles and cans.

Arm Control with Arduino

The two motors on the shoulders and upper arm, and the servos on the lower arms are programmed to be able to control the motion of the arm. The program is written in Arduino IDE on a Linux OS (see Appendix 1). The purpose of programming on a Linux environment is solely to be able to integrate Arduino with ROS.

There are two specific parts in programming the arm control:

A. Programming the servos
B. Programming the gear motors.

Programming the servos

There are four servos that can be programmed to control the motion of the lower arms. A specific set of sequences can be created to be able to further animate the motion of the servos. However, for the purpose of giving options to control individual servos, the code has been written to allow user to enter the specific position to move a servo independent of other servos. The value of each servo location can be entered on the "Serial Monitor" of Arduino.

"**A** "—> Servo to control up-down motion	"**C**"—> Servo to control side-to-side motion
"**F**"—> Servo to control gripper motion	"**W**"—> Servo to control rotation motion

✦ *Flow Chart of Controlling servos*

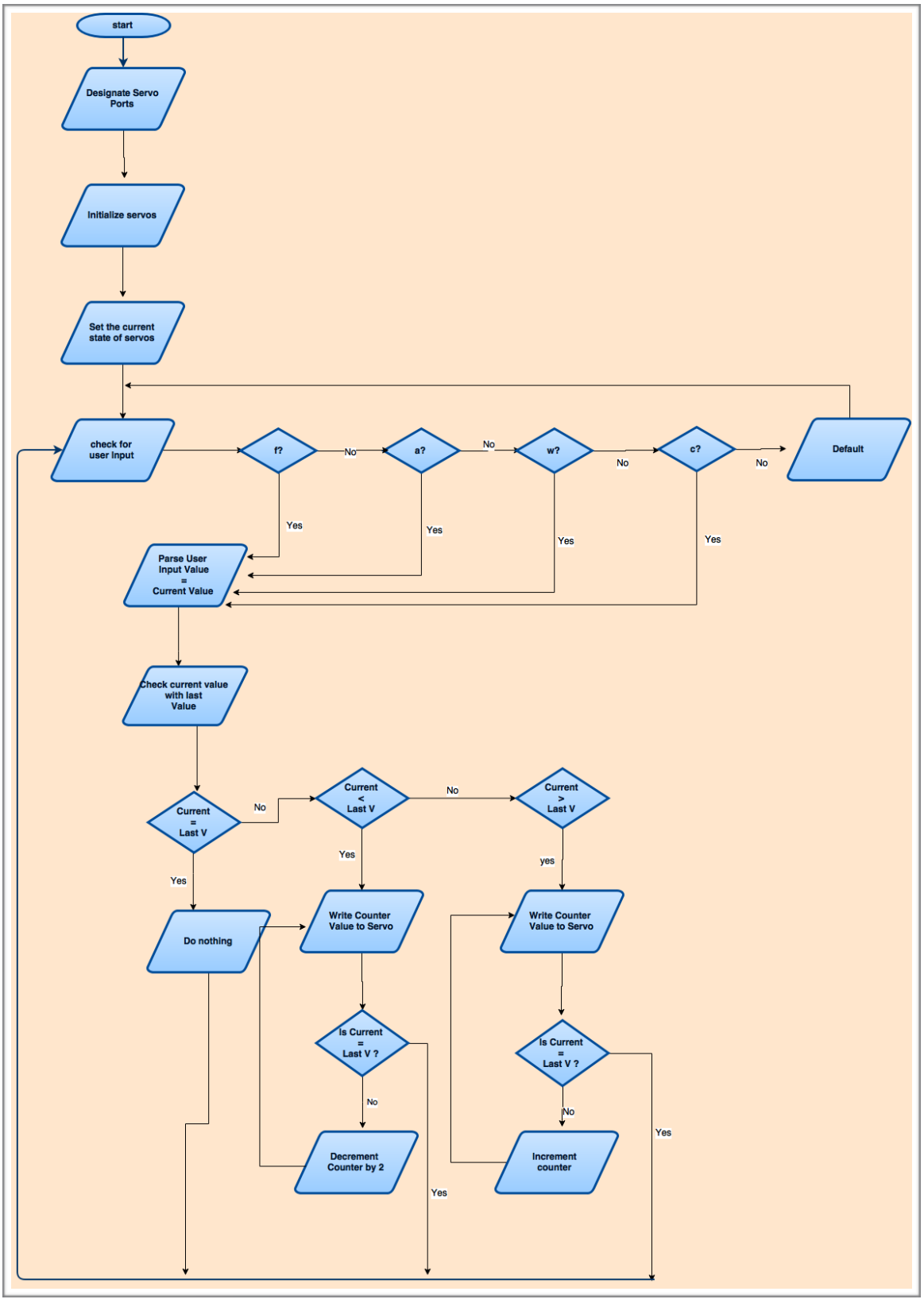


Figure 4: *Flow Chart showing control of servos on the lower arm*

Figure 4 shows the flowchart of how servos are controlled on the lower arm of Countess Quanta. As shown in the diagram, the user has the ability to command the position of a particular servo. If the user input value, aka. current value, is same as the last servo position, no action is taken. Whereas, if the current value and the last servo value don't match, the servo value is gradually increased or decreased and written to servo register until the final position of the servo and the user input values match. The actual code is placed under Appendix 1.

Caution: Since each of the servo can safely handle 5V only, caution is taken to connect the servos at 5V power supply. Also, Arduino and the servo's power source are connected to the common ground by connecting the ground of servos to the ground of Arduino.

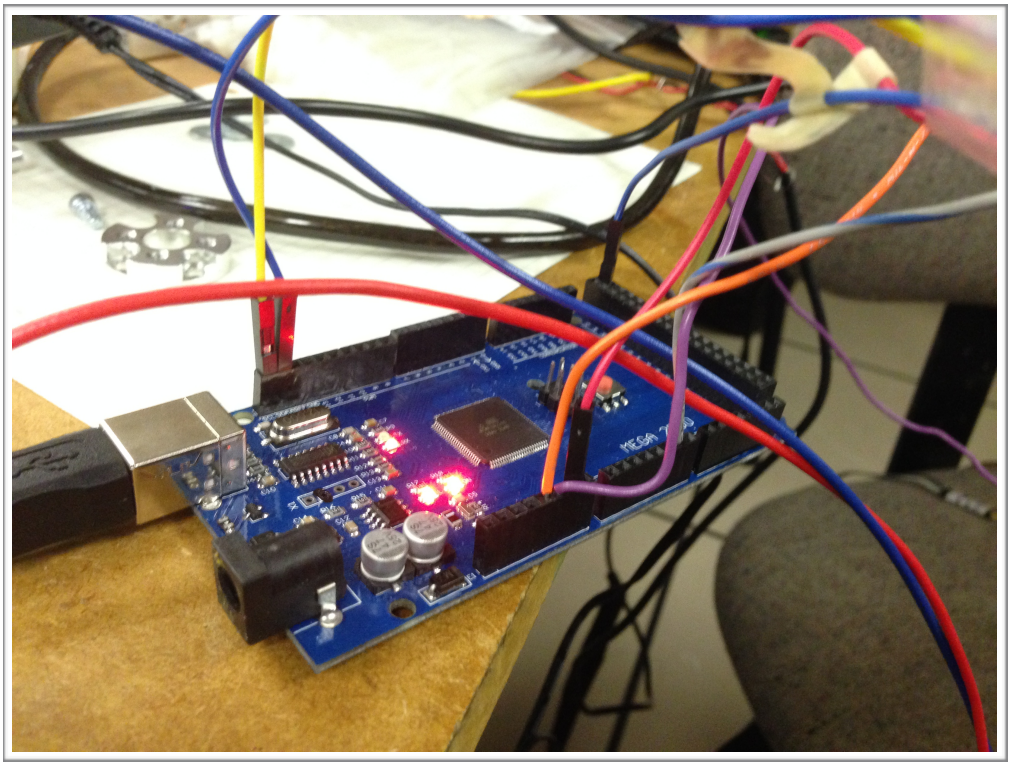


Figure 5: *Connections on Arduino Mega*

Programming Gear Motors

Since the gear motors require separate 12V power supply, the servos are controlled using an external motor controller: HiTechnic DC Motor Controller for Tetrix. This motor controller has registers for two motors (see Appendix2). Also, the modes of the motor controller can be changed according to the need. For example, user can set whether to run the motor in power controlled mode or constant speed mode

The cable of the motor controller has also been modified on one side to be able to connect it to Arduino board. The wires are labelled as shown in figure 6. The motor controller and Arduino are connected as shown in Figure 7.

Similarly, a potentiometer was used to control the position of motor. The motor rotates until the current angular position of the motor and the angle of the potentiometer perfectly match to each other. The direction of the rotation, i.e. counterclockwise or clockwise, of the motor depends on the last position of the potentiometer.

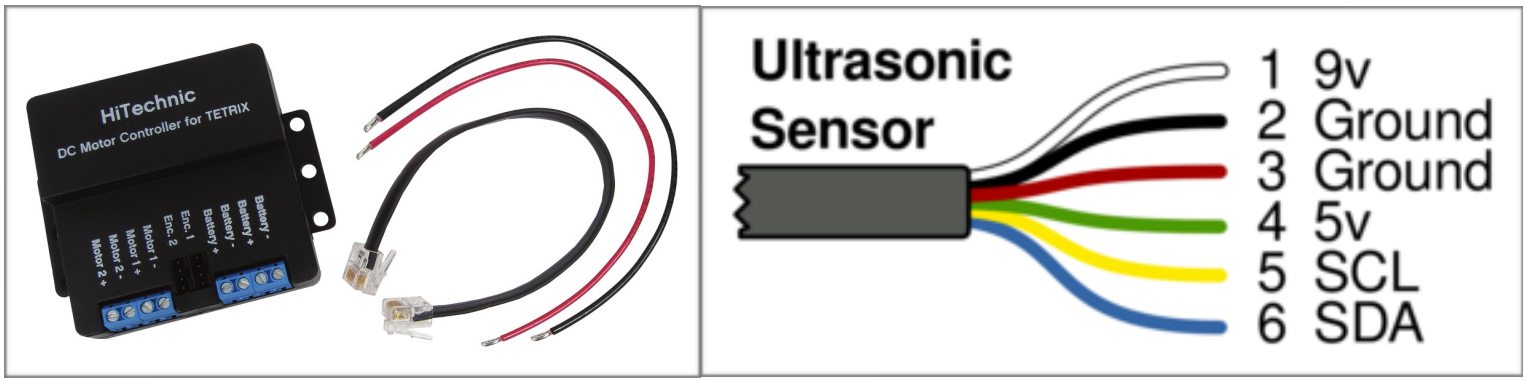


Figure 6: *HiTechnic Motor Controller (left) and connector wiring (right)*

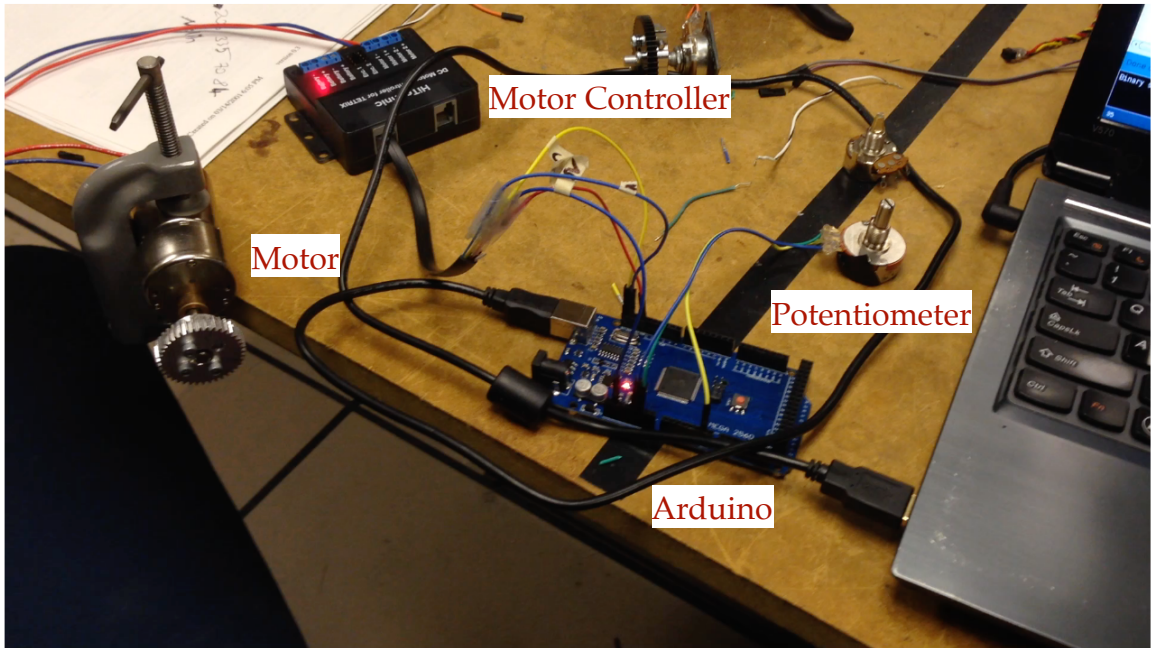


Figure 7: *Showing the connection among Arduino, potentiometer, motor, and motor controller*

✦ *Flow Chart of Controlling a motor*

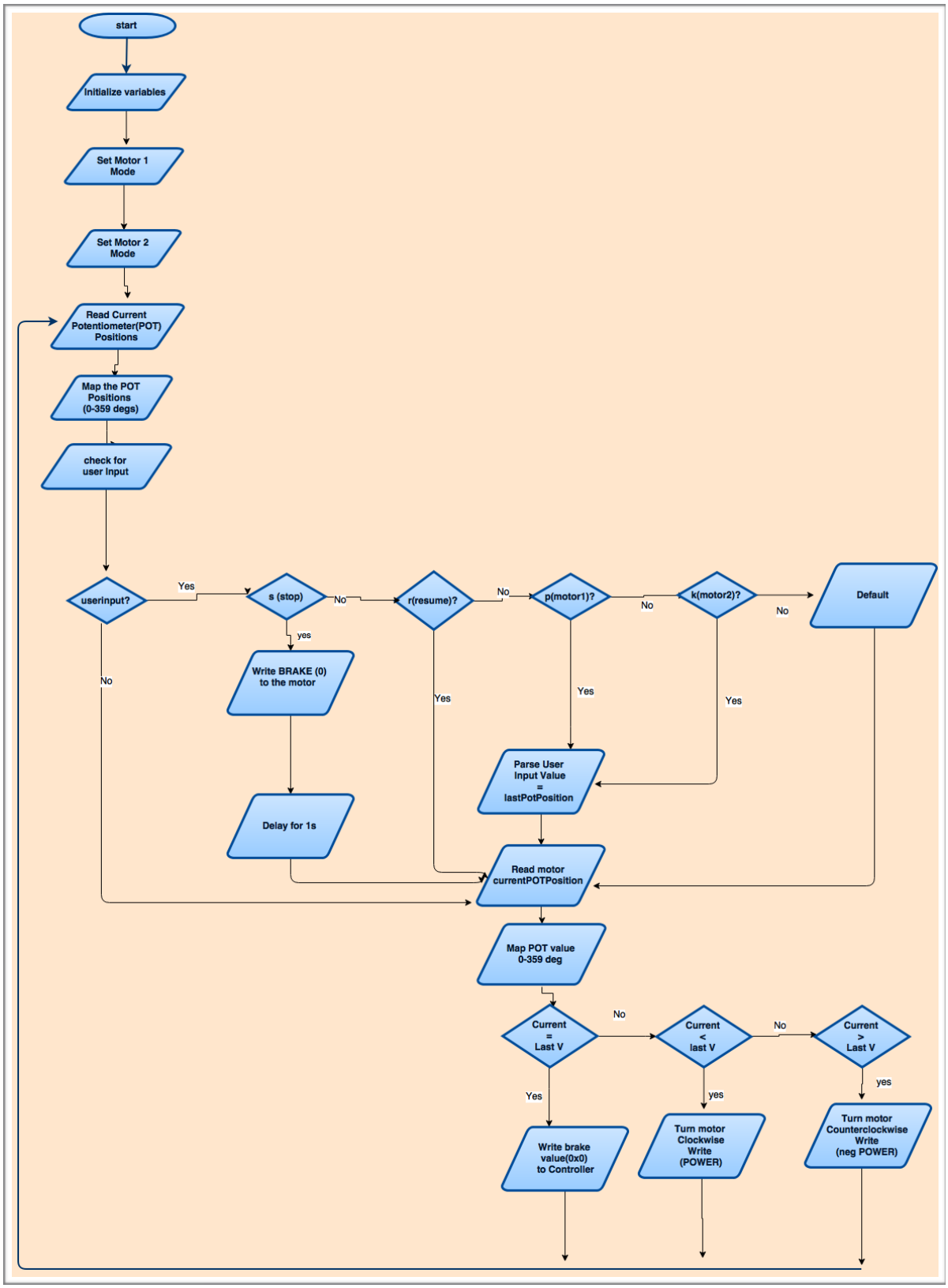


Figure 8: *Flow Chart showing control of Motors*

Figure 8 shows the general flow chart of motor control system. As show in the diagram, there are two motors that are being controlled. The first step is to initialize the motors to specific modes. There are four possible modes that can be set for HiTechnic Motor Controller (See Appendix 2). For the purpose of this program, we set the mode to power control mode writing "0x00" on the mode register of both motors 1 and 2. Then, the program periodically checks whether there is an input from a user. The design is made such that user can input a value in order to drive the position of arm to a specific position. This also allows a platform for an effective debugging. A particular sequence of action can be easily derived merely by writing values to the registers at some interval of time.

There could be three possible states of an arm when a user inputs a value. The given value could be the same, lower, or higher than the requested value. If the current arm position is lower than the input requested value, there is a counterclockwise motion of motor, such that the arm is raised higher up. In contrast, if the current position of arm is higher than the requested arm position, the clockwise rotation of motor is performed to lower the arm. It is also important to note that the given directions are with respect to the motion of motors.

Similarly, the potentiometer connected on the shoulder and upper arm gives necessary feedback to the program about the current arm location. In other words, the current position of the arm is known through the readings of the potentiometer. For that reason, the motor moves in a direction such that the angle of potentiometer and that of motor are equal. If the positions do not match, the motor will keep turning and re-adjust until the match is found.

The states such as "s"—stop and "r"—resume allow user to stop and resume the operation of motor. Similarly, "p" sets value for motor 1 and "k" sets value for motor 2. In order to give a command, a user should write something like "p30" for moving motor 1 to position 30.

Caution: Since the potentiometer have certain range, it is absolutely crucial to make sure that user does not enter values that will force the motor to go out of limit. In that case, there is a higher chance of breaking the potentiometer. The range of lower arm are found to be around p15-p60.

The codes for the motor controller are included in Appendix 1.

Appendix 1

Servo Controller Code

```
 * This code controlls the servos connected to the lower arm of
 * Countess Quanta.
 *
 * Saroj Bardewa
 * March-21, 2016
 * Robotics-2
 * Dr. Marek Perkowski
 */

#include <Servo.h>                          // arduino servo library
#define inputReadValue_delay                 10      // delay for inputReadValue stepsc
#define  FINGER_SERVO_POS           22      //'F'
#define  WRISTS2S_SERVO_POS         24      //'C'
#define  WRISTUPDOWN_SERVO_POS      26      //'A'
#define  WRISTROTATION_SERVO_POS    28      //'W'

#define INIT_FINGERS 0
#define INIT_WRIST_UD 60
#define INIT_WRIST_SIDE2SIDE 60
#define INIT_WRIST_ROTATE 60

// instatiate a servos
Servo fingers;
Servo wristRotation;
Servo wristSide2Side;
Servo wristUpDown;

// Define the present state variables
int presentFState;
int presentAState;
int presentCState;
int presentWState;

// Set up the servos to their initial values
void setup() {
   // initialize serial communication at 9600 baudrate
    Serial.begin(9600);
   // initialize servos and attaces to pins
    fingers.attach(FINGER_SERVO_POS);
    wristRotation.attach(WRISTROTATION_SERVO_POS);
    wristSide2Side.attach(WRISTS2S_SERVO_POS);
    wristUpDown.attach(WRISTUPDOWN_SERVO_POS);


  delay (200); // waits to be assign

  //Initialization
  wristUpDown.write(INIT_WRIST_UD);             // Wrist Up and Down range : 'A' 0 (down), and 180(up)
  delay(1000);
  fingers.write(INIT_FINGERS);                  // Finger Motor range 'F': 90 (close) to 180 (open)
  delay(1000);
  wristRotation.write(INIT_WRIST_ROTATE);       // Wrist Motor range 'W': 15 (out) to 180 (in)   and 145 is neutral
   delay(1000);
  wristSide2Side.write(INIT_WRIST_SIDE2SIDE);   // Carpel Motor range 'C': 0 (in) t0 90 (out)
   delay(1000);
```

```
    // Save all current servo positions
    presentFState = INIT_FINGERS;
    presentAState = INIT_WRIST_UD;
    presentCState = INIT_WRIST_SIDE2SIDE;
    presentWState = INIT_WRIST_ROTATE;

}

  // This loop goes on for ever.
  void loop() {

    int pos, inputReadValue; // variables to be used to send commands

    int currentServoPosition;               // This keeps the information of servo position
    int lastServoPosition;                  // Last Servo position

    // check serial buffer has something
    if (Serial.available() > 0) {
      // read first byte on serial and store it f= 90

      int inByte = Serial.read();

      // check if it is one the defined cases
      switch (inByte) {

      //=============================================================================
      case 'f':
          // Valid case, wait for an int and store it in pos
          inputReadValue = Serial.parseInt();

          // provides feedback for debuf if needed
          #ifndef Silent
          Serial.print("Servo Fingers=");
          Serial.println(pos);
          #endif /* Silent */
              // only two aceptables values for inputReadValue 180 and 0
        if (inputReadValue<=presentFState)
        {
          // loop from 0-90 using steps of 2 has a defined delay
          for (int i=presentFState; i>=inputReadValue; i=i-2)
          {
            fingers.write(i);
           delay(inputReadValue_delay);
          }
          presentFState = inputReadValue;
        }
         else if( inputReadValue>=presentFState){
          // loop from 90 back to 0 with steps of 2 has a defined delay
           for (int i=presentFState; i <=inputReadValue; i=i+2)
          {
            fingers.write(i);
           delay(inputReadValue_delay);
          }
          presentFState = inputReadValue;
         }

         else
           fingers.write(inputReadValue);
        break;
        //=============================================================================
```

```
    //=========================================================================
    case 'a':
      // Valid case, wait for an int and store it in pos
      inputReadValue = Serial.parseInt();

      // provides feedback for debuf if needed
      #ifndef Silent
      Serial.print("WristUpDown position=");
      Serial.println(pos);
      #endif /* Silent */
          // only two aceptables values for inputReadValue 180 and 0
     if ( inputReadValue <=presentAState)
     {
      // loop from 0-90 using steps of 2 has a defined delay
      for (int i=presentAState; i>=inputReadValue; i=i-1)
      {
        wristUpDown.write(i);
       delay(inputReadValue_delay);
      }
      presentAState = inputReadValue;
     }
      else if( inputReadValue >=presentAState){
      // loop from 90 back to 0 with steps of 2 has a defined delay
       for (int i=presentAState; i <= inputReadValue; i=i+1)
      {
        wristUpDown.write(i);
       delay(inputReadValue_delay);
      }
      presentAState = inputReadValue;
     }
     break;
     //=========================================================================
  case 'w':
     inputReadValue = Serial.parseInt();

     #ifndef Silent
     Serial.print("Servo wrist_position=");
     Serial.println(pos);
     #endif /* Silent */
         // only two aceptables values for inputReadValue 180 and 0
   if (inputReadValue <=presentWState)
   {
     // loop from 0-90 using steps of 2 has a defined delay
     for (int i=presentWState; i>=inputReadValue; i=i-2)
     {
       wristRotation.write(i);
      delay(inputReadValue_delay);
     }
    presentWState = inputReadValue;
   }
    else if( inputReadValue >=presentWState){
     // loop from 90 back to 0 with steps of 2 has a defined delay
      for (int i=presentWState; i <=inputReadValue; i=i+2)
     {
       wristRotation.write(i);
      delay(inputReadValue_delay);
     }
     presentWState = inputReadValue;
    }
   break;
   //=========================================================================
```

```
//=============================================================================

  case 'c':
      inputReadValue = Serial.parseInt();

      #ifndef Silent
      Serial.print(" wrist_side2side position=");
      Serial.println(pos);
      #endif /* Silent */


    // only two aceptables values for inputReadValue 90 and 0
    if ( inputReadValue <presentCState)
    {
      // loop from 0-90 using steps of 2 has a defined delay
      for (int i=presentCState; i>=inputReadValue; i=i-2)
      {
        wristSide2Side.write(i);
       delay(inputReadValue_delay);
      }
     presentCState = inputReadValue;
    }
     else if( inputReadValue >=presentCState){
      // loop from 90 back to 0 with steps of 2 has a defined delay
       for (int i=presentCState; i <=inputReadValue; i=i+2)
      {
       wristSide2Side.write(i);
       delay(inputReadValue_delay);
      }
      presentCState = inputReadValue;
     }
    break;
//=============================================================================
   // if no case define it doesn't do anything
    default:
      break;

  }/* end switch*/
  }/* end if */
  }/* end while */
```

Motor Controller Code

```
/*****************************************************************************************
  Program to control Gear Motor using Hitechnic DC Motor Controller,
  Arduino, and Potentiometer.

  Saroj Bardewa
  March 19, 2016
  Portland State University
******************************************************************************************/

#include <Wire.h>

#define DEVICE_ADDRESS 0x05
#define NUM_BYTES 8               //D7-D0
#define M2_POWER 0x46             // Motor 2 Power
#define M1_MODE  0x44             // Motor 1 Mode
#define M1_POWER 0x45             // Motor 1 Power
#define BRAKE    0                // This brakes the motor controller

int potPin6 = 6;
int potPin7 = 7;
int i = 0;
int lastPotPosition1 = 0;
int lastPotPosition2 = 0;
int testVar = 0;
void setup()
{
 Serial.begin(9600);
 analogReference(DEFAULT);
 Wire.begin(); // join i2c bus (address optional for master)

 Wire.beginTransmission(DEVICE_ADDRESS); //begin transmission at DEVICE_ADDRESS (connect to device 0x05)
 Wire.write(M1_MODE); // Motor 1 Mode register
 Wire.write(0);        // Run to position mode
 Wire.endTransmission(0); // send a restart message to keep the connection alive (false(0)); master will not release bus

}
 void loop()
 {
   char userInput;
   int pos;
   int moveToPosition;

  // Read current Pot position
 int currentPotPosition1 = analogRead(potPin6);
    currentPotPosition1 = map(currentPotPosition1, 0, 1023,0,360);
 int currentPotPosition2 = analogRead(potPin7);
    currentPotPosition2 = map(currentPotPosition2, 0, 1023,0,360);

  //   Serial.println(currentPotPosition);

 // Read command from the user if any command is entered
   if(Serial.available() > 0)
   {
        userInput = Serial.read();

        switch(userInput)
        {
          case 's':
                pos = Serial.parseInt();
                 Serial.print("\nStopping!\n");
                 //Stop the motor
                Wire.beginTransmission(DEVICE_ADDRESS); //begin transmission at DEVICE_ADDRESS
                Wire.write(M1_POWER); // write to motor 1 power address
                Wire.write(0);          // Value to write to the motor
                Wire.endTransmission(0);// End of Transmission
                delay(1000);  // Wait for 1 sec
                break;

          case 'r':
                  pos = Serial.parseInt();
                  Serial.print("\nResuming!\n");
                  break;
          case 'p':
                   lastPotPosition1 = Serial.parseInt();
                   Serial.print("Set position to: \n");
                   Serial.println(lastPotPosition1);
                  break;

          case 'k':
                   lastPotPosition2 = Serial.parseInt();
                   Serial.print("Set position to: \n");
                   Serial.println(lastPotPosition2);
                   break;
          default:
                 break;
```

```
        }
       // delay(100);
        // Call the motorController Function
    }

          // Call the function
    while(!testVar)
     {
         testVar = motorController(lastPotPosition1,currentPotPosition1,M1_POWER);
          // Read current Pot position
         currentPotPosition1 = analogRead(potPin6);
         currentPotPosition1 = map(currentPotPosition1, 0, 1023,0,360);

     }

     testVar = 0;
 /*
    while(!testVar)
     {
       testVar = motorController(lastPotPosition2,currentPotPosition2,M2_POWER);
       currentPotPosition2 = analogRead(potPin7);
       currentPotPosition2 = map(currentPotPosition2, 0, 1023,0,360);
     }

    testVar = 0;

    */
} // End loop

//=============================================================================================
// This function gets the information on the current and last position of potentiometer and moves the motor
//    to the defined position
```

```
int motorController(int lastPotPosition, int currentPotPosition,int motor)
{
                    Serial.print("In Motor Func Last Pos =: \n");
                    Serial.println(lastPotPosition);
                    Serial.print("\n CurrFunc Pos =: \n");
                    Serial.println(currentPotPosition);

    if((lastPotPosition ==currentPotPosition) || abs(lastPotPosition - currentPotPosition) <7)
    {
        delay(100);
        return 1;
    }

    else if (lastPotPosition > currentPotPosition)
    {
        // That means last position was lower, so raise it higher by clockwise rotation

        Wire.beginTransmission(DEVICE_ADDRESS); //begin transmission at DEVICE_ADDRESS (connect to device 0x05
        Wire.write(motor); // write to motor 1 power address
        Wire.write(20); // send the data byte; write the percent specified when function is called to above motor
        Wire.endTransmission(0); // send a restart message to keep the connection alive (false(0)); master will not release bus
        delay(lastPotPosition-currentPotPosition+100);

        Wire.beginTransmission(DEVICE_ADDRESS); //begin transmission at DEVICE_ADDRESS (connect to device 0x0
        Wire.write(motor); // write to motor 1 power address
        Wire.write(0); // send the data byte; write the percent specified when function is called to above motor
        Wire.endTransmission(0); // send a restart message to keep the connection alive (false(0)); master will not release bus

     }
    else if(lastPotPosition < currentPotPosition)
    {
        // That means last position was higher, so lower it  by counter-clockwise rotation

        Wire.beginTransmission(DEVICE_ADDRESS); //begin transmission at DEVICE_ADDRESS (connect to device 0x05
        Wire.write(motor); // write to motor 1 power address
        Wire.write(-20); // send the data byte; write the percent specified when function is called to above motor
        Wire.endTransmission(0); // send a restart message to keep the connection alive (false(0)); master will not release bus
       delay(currentPotPosition-lastPotPosition +100);

        Wire.beginTransmission(DEVICE_ADDRESS); //begin transmission at DEVICE_ADDRESS (connect to device 0x0
        Wire.write(motor); // write to motor 1 power address

        Wire.write(0); // send the data byte; write the percent specified when function is called to above motor
        Wire.endTransmission(0); // send a restart message to keep the connection alive (false(0)); master will not release bus
       }

    return 0;
}
```

Appendix 2

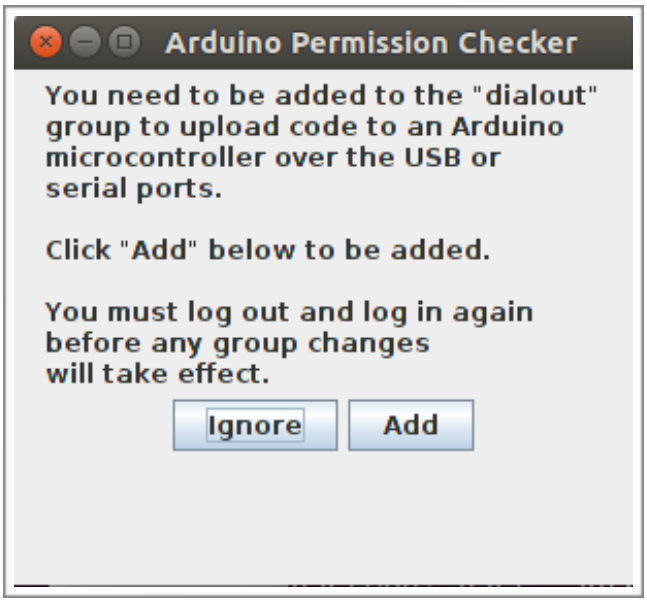
Installing Arduino IDE on Linux Ubuntu

The following steps will guide you through installing Arduino IDE on Linux Ubuntu

Step 1: Install Linux from ubuntu repository

```
sudo apt-get install arduino
```

Step 2: If you don't get an error like the one shown alongside when you run your Arduino IDE, you should are all set. If you get the error follow step 3



Step 3: Open your terminal (CTRL +ALT+t) and type this command before you plug in your Arduino board

```
dmesg
```

Step 4: Now plug in your board again, and type the *dmesg* again to see the device ID for your Arduino board

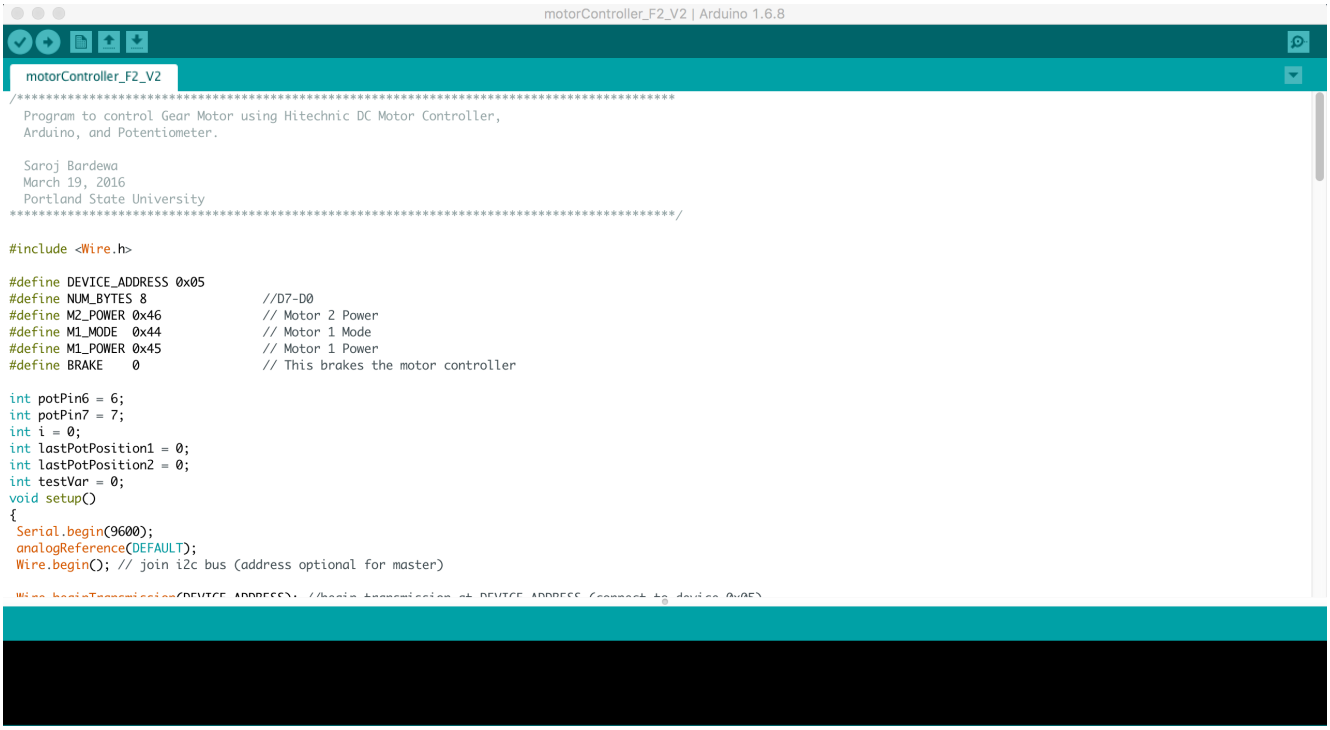
Step 5: Change the permission of your device as follows: *Note: ttyUSB0 is the deviceID.*

```
root:saroj# ls -l /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 0 Mar 17 15:02 /dev/ttyUSB0
root:saroj# usermod -a -G dialout saroj
root:saroj#
```

Programming in Arduino IDE

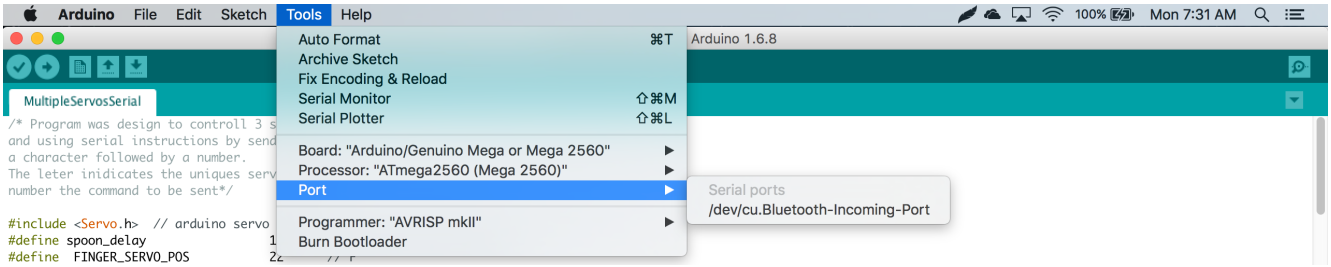
Follow these steps to compile and run your program in Arduino IDE:

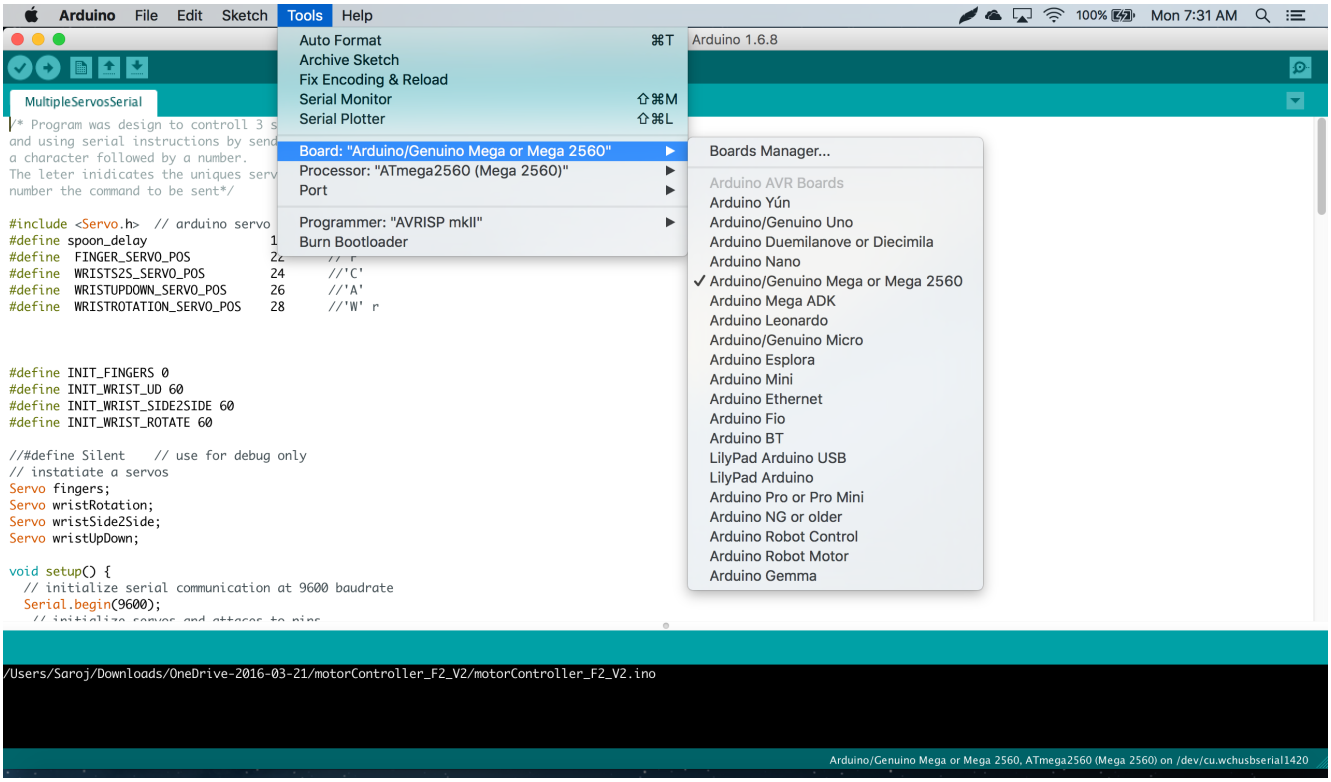
Step 1: Open your Arduino IDE
Step 2: Write your code on Arduino IDE



Step 3: Compile by code by typing CRTL+R or just click on  icon.
Step 4: Make corrections to your code if there is any error

Step 5: Make sure to select your serial Port to the Device port and your board name:





Step 6: Upload your code to the board by CTRL + U or click  icon.

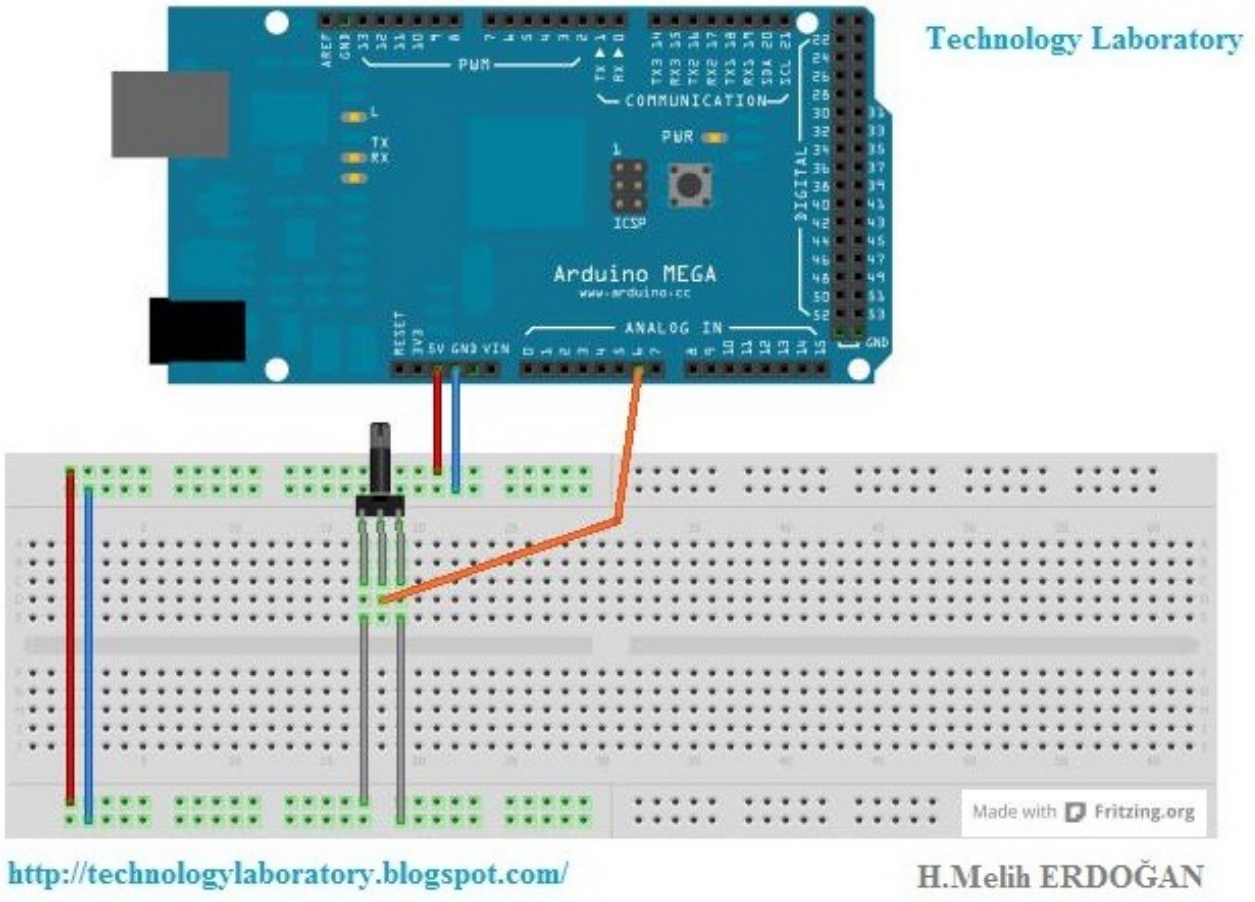
Connecting HiTechnic DC Motor Controller and Potentiometer on Arduino

I2C Protocol is used to communication between HiTechnic DC Motor Controller and Arduino. The following steps are for connecting HiTechnic motor controller and Potentiometer:

Step 1: Make sure your wires are connected properly. Connect the following wires properly:

Clock—> SCL on Arduino
Data—> SDA in Arduino
Vin —> 5V
GND—> Ground

Step 2: Connect HiTechnic Motor Controller to a 12V external power supply:
Step 3: Connect the potentiometer as follows:



I2C Protocol to talk to HiTechnic Motor Controller

HiTechnic Motor Controller uses I2C protocol to interact with Arduino. Follow these steps for I2C protocol:

Step 1: Always sent Device ID to initiate the protocol followed by the register address and data to write[2]

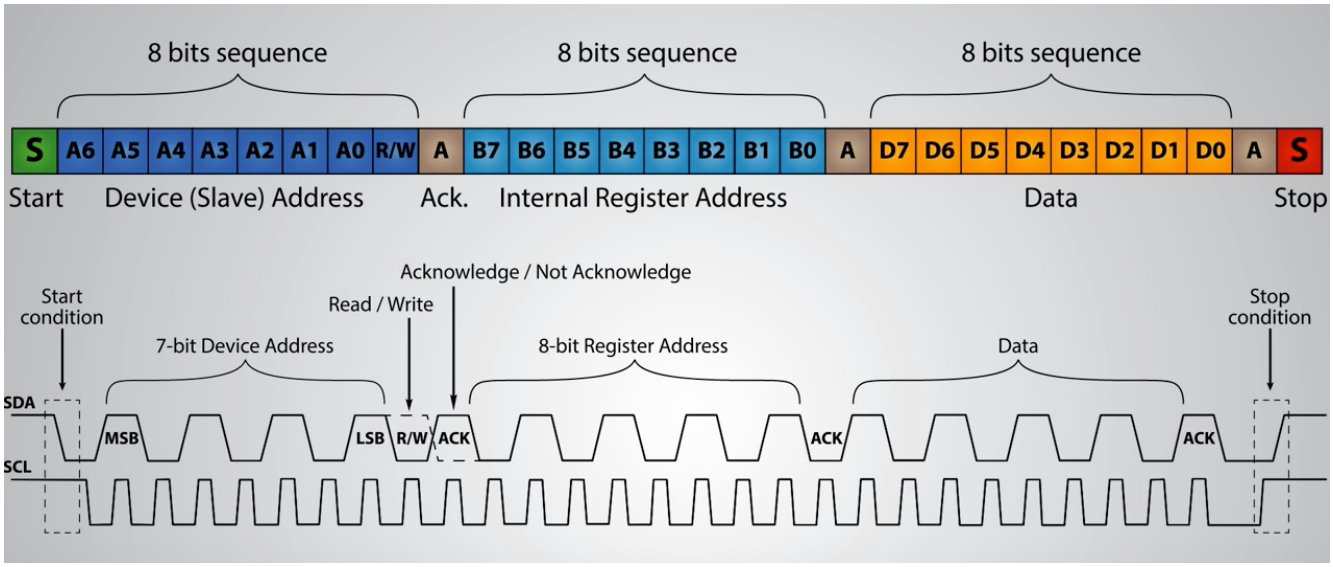


Figure: Showing I2C protocol [[2]

Step 2: To end the communication, always sent *end.Transmission(0)* in Arduino.

Registers in HiTechnic DC Motor Controller

The following are the registers in HiTechnic DC Motor Controller[3]. The mode register determines the mode of the motor. Hence, it is crucial to follow the addressing modes as shown below:

Address	Type	Contents
00 – 07H	chars	Sensor version number
08 – 0FH	chars	Manufacturer
10 – 17H	chars	Sensor type
18 – 3DH	bytes	Not used
3E, 3FH	chars	Reserved
40H – 43H	s/long	Motor 1 target encoder value, high byte first
44H	byte	Motor 1 mode
45H	s/byte	Motor 1 power
46H	s/byte	Motor 2 power
47H	byte	Motor 2 mode
48 – 4BH	s/long	Motor 2 target encoder value, high byte first
4C – 4FH	s/long	Motor 1 current encoder value, high byte first
50 – 53H	s/long	Motor 2 current encoder value, high byte first
54, 55H	word	Battery voltage 54H high byte, 55H low byte
56H	S/byte	Motor 1 gear ratio
57H	byte	Motor 1 P coefficient*
58H	byte	Motor 1 I coefficient*
59H	byte	Motor 1 D coefficient*
5AH	s/byte	Motor 2 gear ratio
5BH	byte	Motor 2 P coefficient*
5CH	byte	Motor 2 I coefficient*
5DH	byte	Motor 2 D coefficient*

D7	D6	D5	D4	D3	D2	D1	D0
Busy	Error	-	NTO	Rev	Lock	Sel 1	Sel 0

Figure: *Fields in Mode Register*

Sel	Action
00	Run with power control only
01	Run with constant speed
10	Run to position
11	Reset current encoder

Bibliography

[1]"Loki", Shinsel Robots, 2012. [Online]. Available: http://www.dshinsel.com/loki/. [Accessed: 21- Mar- 2016].

[2]"How I2C Communication Works and How To Use It with Arduino", YouTube, 2016. [Online]. Available: https://www.youtube.com/watch?v=6IAkYpmA1DQ. [Accessed: 21- Mar- 2016].

[3]"HiTechnic FIRST Motor Controller Specification", HiTechnic, 2016. [Online]. Available: http://www.hitechnic.com/blog/wp-content/uploads/HiTechnic-Motor-Controller-Specification.pdf. [Accessed: 21- Mar- 2016].