DoodleBot

ME 5643 Final Project Report

Matt Galligan, Dustyn Roberts, Hussein Saab

December 19, 2011

1 Motivation

The motivation for the DoodleBot stemmed from a MATLAB product demonstration given at NYU-Poly on September 23, 2011 where they demonstrated closed loop control of a robotic arm through MATLAB with a video game controller. When the final project guidelines included "robotic manipulator" in the list of suggested projects, we decided to move forward with some sort of robotic arm. Two of us had taken more than one robotics class, and all had experience in control theory, but we had never implemented closed loop control on actual hardware. By building a robotic arm from scratch - from hardware through software design - we were able to not only meet project requirements but also provide ourselves with a learning platform.

After initial research, we realized that a generic robotic arm with modular parts and expandable control software could have several potential uses. For example, the Meal Mate robotic feeder [1] and the Meal Buddy Robotic Assistive Feeder [2]are both robotic arms designed to help those with cerebral palsy, head/spinal injuries, rheumatoid arthritis, muscular dystrophy, upper limb amputations etc, and many other persons with learning difficulties or neurological conditions eat independently (Figure 1). A device like this would satisfy the "smart kitchen aids, smart home appliances," and "projects that respond to a societal need" sections in the list of suggested projects.

Figure 1: The Meal Mate robotic feeder [1] (left) and the Meal Buddy Robotic Assistive Feeder [2](right)

Another application of robotic arms can address two other sections in the suggested project guidelines: "enhance laboratory education in grades K-12" and "mechatronics-enabled pre-college level science experiments (physics)". A generic robotic arm could be used as an educational platform. LEGOs are already a popular way to teach mechatronics to this age group, and LEGO robotic arms already exist (Figure 2). The company has also recently released new sets aimed at encouraging young women to explore robotics.

Figure 2: LEGO robotic arm [3] (left) and a new LEGO kit [4] (right) targeted at encouraging young women to explore robotics

Additionally, toy robotic arms are becoming increasingly affordable, with one particular example from OWI Robotics retailing for just $40 (Figure 3). However, both the LEGO arm and toys like the OWI Robot Edge obscure the actual motors and electronics that go into such a project. It is difficult to make the leap from playing with LEGOs and toy robots to designing real world robotic systems.

Figure 3: OWI Robotic Arm Edge

It follows from the above that a transparently designed real world robotic arm might serve as an engaging platform for mechatronics learning. Robotics can be intimidating, but it's clear from numerous studies that the US needs to encourage interest in STEM education in order to maintain a globally competitive work force [5]. One way to mask a seemingly intimidating subject like robotics is to include familiar functionality. We chose drawing as our means of making the robotic arm more interesting and engaging to those who might not otherwise be drawn to explore robotics.

Several robots already exist that have been deployed in museums, galleries, and public outreach events to engage audiences both young and old (Figure 4). We designed our robotic arm to work in similar environments. This will help address a societal need to encourage students to pursue STEM education.

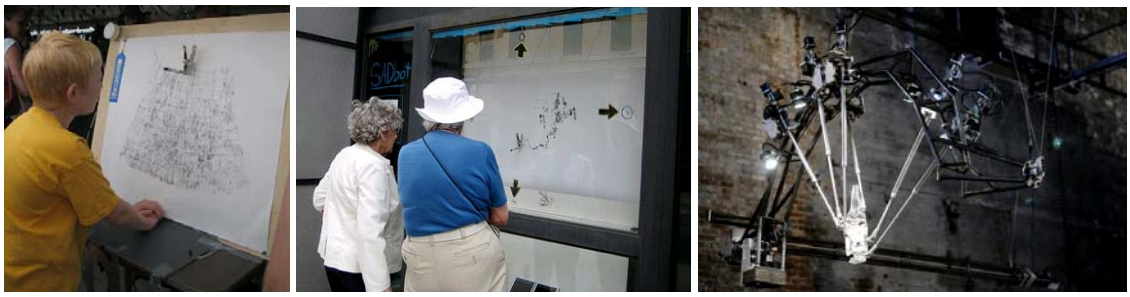

Figure 4: (Left) SADbot at Maker Faire (Middle) SADbot in Eyebeam's window gallery [6] (Right) Robotic Installation art [7]

2 Theory/Mathematical Background

2.1 Two link manipulator

The DoodleBot is a two link planar manipulator. One of the links is connected to a fixed point while the other link is connected to the end point of the first one. The end point of the second link (the end effector) is free to move. A diagram for a two link planar manipulator is shown in Figure 5. Two problems are encountered when trying to find the values of the angles and the position of the end point. The problems are known as the forward kinematics and inverse kinematics problems.

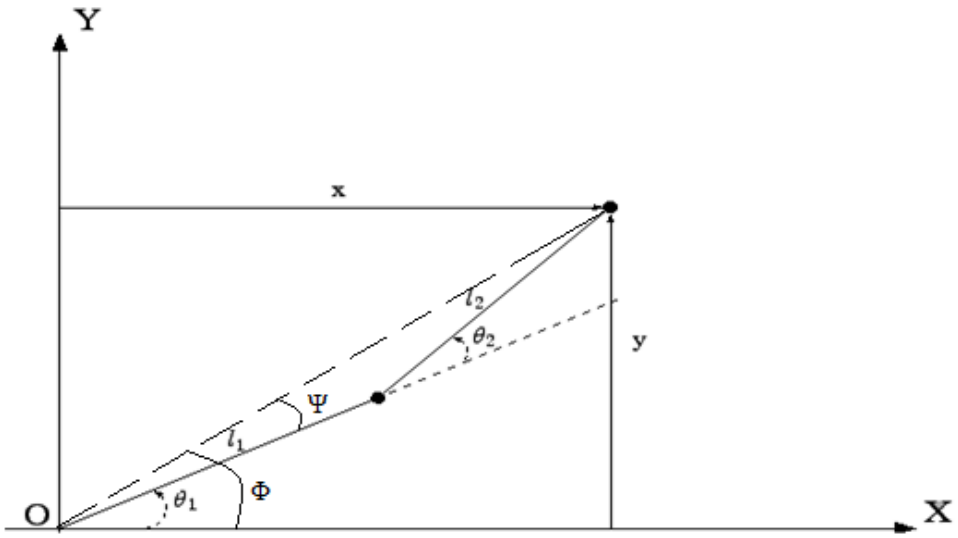

Figure 5: Kinematics of a two link manipulator

The inverse kinematics is the problem of determining the joints angles based on a given position of the end effector of the manipulator. The equations relating the joint angles for a two link manipulator are:

$$\theta_2 = \arccos\left(\frac{x^2+y^2-l_1^2-l_2^2}{2l_1l_2}\right)$$

$$\Phi = \text{atan2}\ (y, x)$$

$$\Psi = \text{atan2}\ (l_2 \sin\theta_2,\ l_1 + l_2\cos\theta_2)$$

$$\theta_1 = \Phi - \Psi$$

Where l_1 and l_2 are the link 1 and 2 lengths respectively and θ_1 and θ_2 are the angles corresponding to each link.

In contrast to inverse kinematics, forward kinematics gives the position of the end point based on the given joint angles. The equations for x and y coordinates of the end point are given below based on the nomenclature in Figure 5:

$$x = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$$
$$y = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)$$

By using these equations we can calculate the desired angles based on the desired location of the end point and use that as the reference input.

2.2 DC Motor Modeling

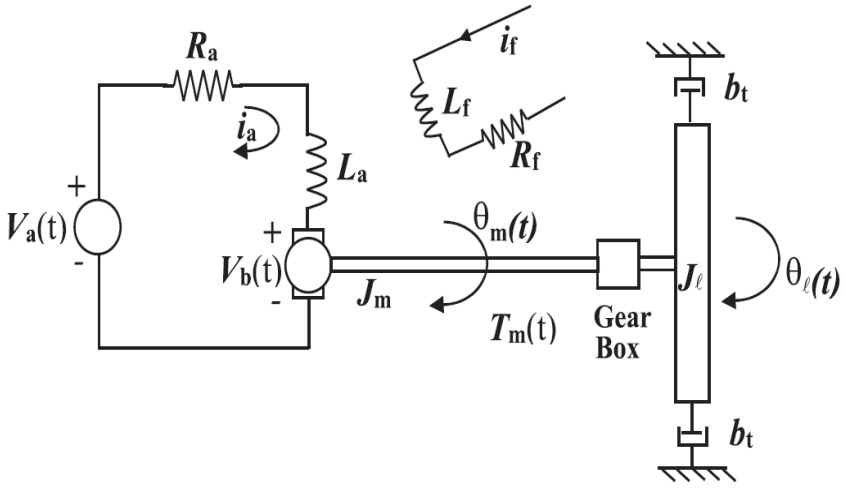

Figure 6: DC motor schematic

The DC-motors used in DoodleBot consist of two subsystems can be modeled as electrical and mechanical subsystems. The transfer function, that relates voltage input V_a to the DC-motor load angular velocity w_l is:

$$\frac{w_l(s)}{V_a(s)} = \frac{K_g K_T(s)}{R_a J_{eq} s + R_a b_t + K_g^2 K_b K_T}$$

where J_{eq} is the total load inertia reflected at the motor shaft, K_g is the gear ratio, K_b is the motor speed constant ((Vs)/rad), K_T is the motor torque constant ((Nm)/A), and b_t is the rotational viscous friction constant. The transfer function can be simplified in the following form:

$$\frac{w_l(s)}{V_a(s)} = \frac{K}{\tau s + 1}$$

Where τ and K are the mechanical time constant and the dc-gain of the DC servomotor respectively. To experimentally determine K and τ for the DC-motor, the transient and steady-state angular velocity step response of the DC-motor can be obtained by applying a step input voltage and measuring the angular velocity of the link. A plot of such a response is shown below Figure 7:

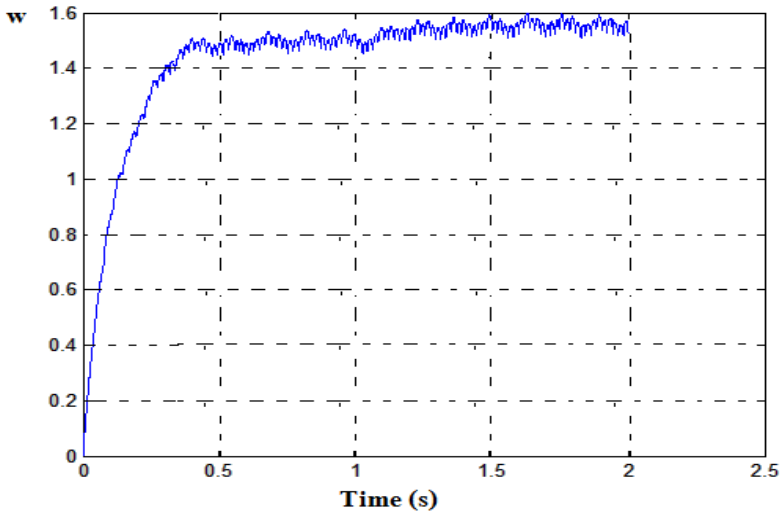

Figure 7: DC Motor angular velocity step response

The time constant is defined as the time required for the output to reach 63.2% of the final output value, which can be obtained using the plot in Figure 7. Then K is found by using the transfer function and applying the final value theorem (FVT): A step input yields V(s) = $\frac{1}{s}$

$$\omega = \frac{k}{s(\tau s+1)}$$

$$\text{FVT: } \omega_{ss} = \lim_{s\to 0} \frac{k}{s(\tau s+1)} = K$$

Where ω_{ss} is the steady state angular velocity.

2.3 Proportional Integral Derivative (PID) controller

$$U(t) = K_p\, e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt}$$

As the name implies the PID controller has three gains, a proportional (K_p), derivative (K_D)and integral (K_I) gain. A PID controller calculates an error value as the difference between a measured process variable and a desired set point. The controller attempts to minimize the error by adjusting the process control inputs. The proportional, the integral and derivative values, denoted P, I, and D, can be interpreted in terms of time: P depends on the present error, I on the accumulation of past errors, and D is a prediction of future errors, based on thecurrent rate of change.

Once K and τ are determined, the PID controller gains can be determined after getting the closed loop transfer function corresponding to the closed-loop feedback system shown in Figure 8.

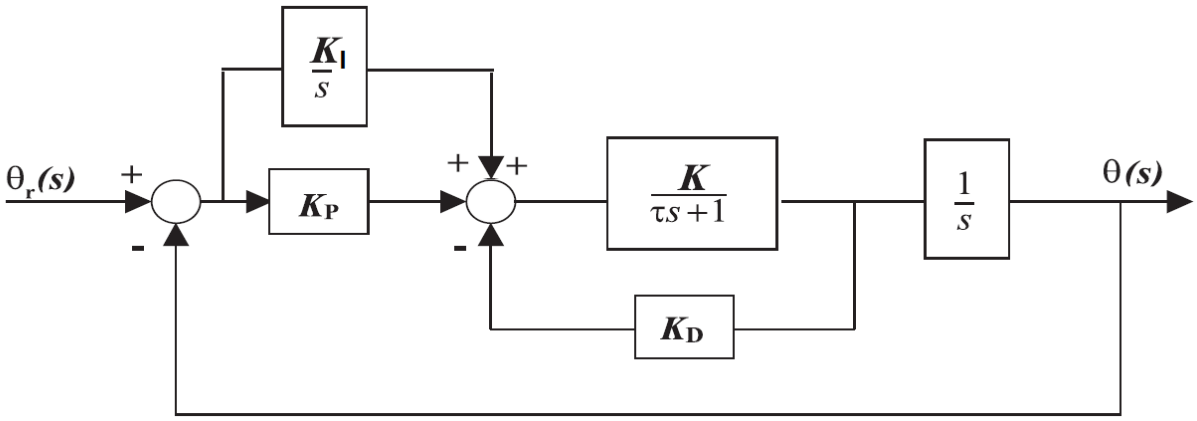

Figure 8: Closed-Loop Feedback PID control of the DC Motor

The closed loop transfer function is:

$$\frac{\theta_s}{\theta_r} = \frac{K_p K s + K_I K}{\tau s^2 + (1 + K K_D) s^2 + K K_p s + K K_I}$$

3 Project Specifications

The DoodleBot complies with all of the following project specifications. Details are given in subsequent sections.

- Your device will be controlled by BS2 (or another microcontroller of your choice)
- You **must** incorporate and document hardware and software features to prevent damage to the BS2 IC and other components on your device. In addition, you must provide guidelines for safe operation of your device.
- Include a provision for instantaneous shutdown of your device in case of incorrect/unsafe operation.
- Your project **must** include some form of a user interface so that a human user can monitor and control your device.
- Your project **must** utilize at least one actuator.
- Your project **must** utilize at least one analog and one digital sensor.
- Your actuator **must** be controlled using sensory feedback. You can use any primitive to advance control design methodology for this purpose.

4 Mechanical Design

In order to plan what parts to order and confirm everything would work as intended, all mechanical systems were modeled in Solidworks before purchase and assembly. Off the shelf parts were used whenever possible to keep the mechanical design and assembly time manageable and provide a modular system that could be easily duplicated. Solid models of some of the parts were already available online [8], and these were configured into a two degree of freedom (DOF) planar robotic arm actuated by standard size servomotors (Figure 9). A solenoid was mounted at a right angle to the planar arm to provide a way to extend and retract a drawing implement at the

end effector. This robotic arm is referred to as the slave arm, as opposed to the master arm, which will be described later. A base was machined out of an aluminum u-channel to mount the robotic arm as well as the electronics.

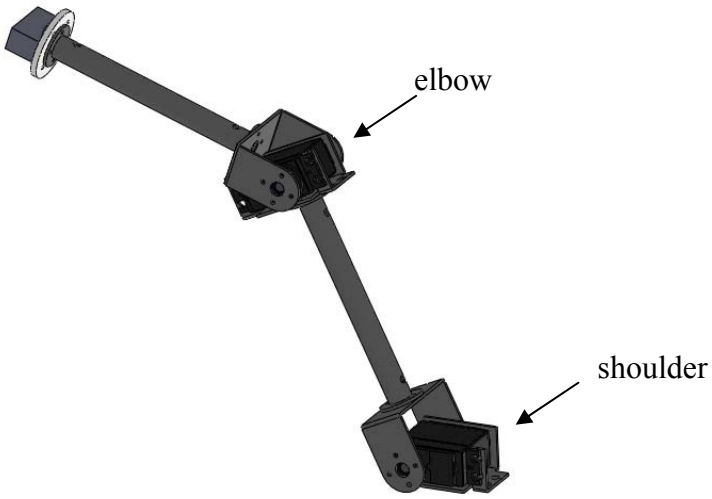

Figure 9: Screenshot from Solidworks with details of slave arm hardware design

4.1 User Interface (Master Arm)

A master arm was modeled and fabricated on a Dimension Elite 3D printer to act as the user interface to monitor and control the slave arm (Figure 10). Potentiometers were placed at each of the DOF to use as the reference signal for the commanded arm configuration.

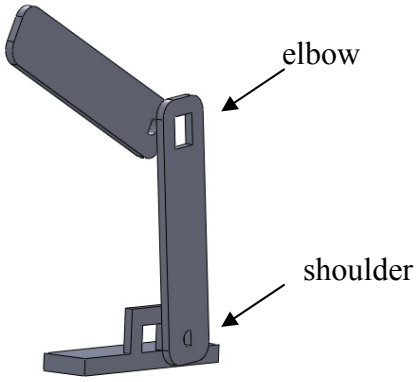

Figure 10: Screenshot from Solidworks showing basic master arm design

5 Electrical Design

In order to determine the required torque in the shoulder motor, we assumed that all the weight of the elbow motor, solenoid, and connecting components (approximately 5 ounces) was concentrated at the end effector, 14 inches from the axis of rotation of the shoulder motor. If the arm was fully extended horizontally, the shoulder model would need at least 70 in-oz of torque to lift the arm. We had some HS-7950TH high torque servomotors (344 in-oz), but the control boards had been destroyed by an accidental application of high voltage in a previous project. We chose to remove the control boards, effectively making them direct current (DC) gearhead motors with already integrated potentiometers to use as feedback (Figure 11). We left the mechanical hard stops inside the gearboxes intact because we did not need more than 180°

rotation in either DOF. This step allowed us to lower the cost of procuring parts while recycling otherwise useless motors. We then soldered wires from ribbon cable to each of the two DC motor leads and each of the three existing potentiometer wires to allow us to interface with the solenoid from down near the base of the robotic arm.

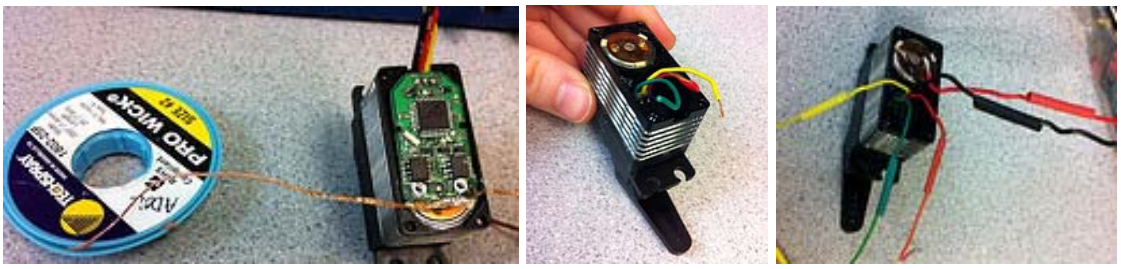

Figure 11: Hacking a servo motor to use as a DC gearhead motor

A solenoid was mounted to a custom designed washer on the end effector that was 3D printed. Epoxy putty was used to create a seat for a drawing implement on the plunger of the solenoid. Longer wires were soldered onto the existing ones to allow us to interface with the solenoid from down near the base of the robotic arm. A transistor was used to allow current to flow through the solenoid when the base was activated by a digital signal from the Arduino microcontroller (Figure 12).

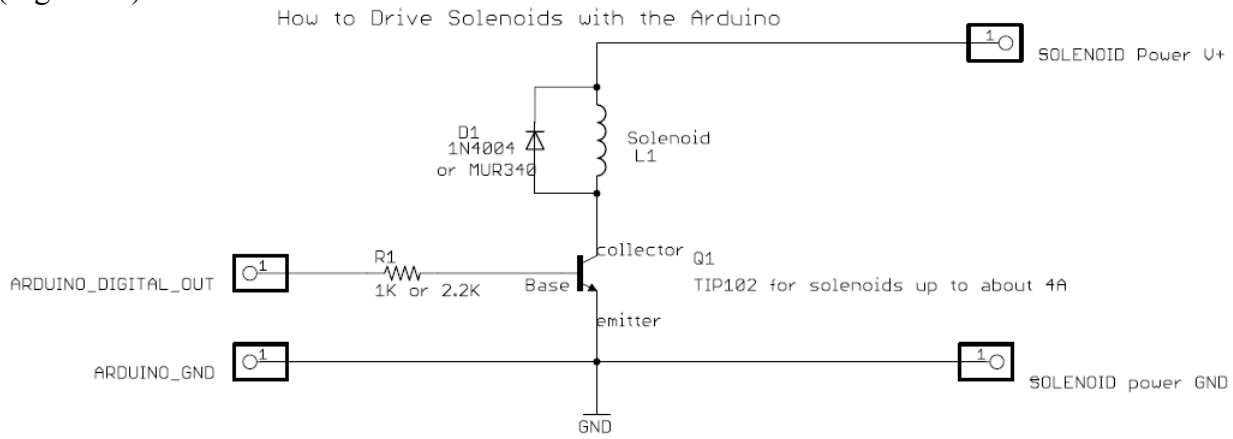

Figure 12: Example schematic of interfacing a solenoid with an Arduino [9]

The Arduino prototyping platform (Figure 13 - left) was used as the controller for the DoodleBot [10]. The on board ATmega328 has 14 digital input/output pins (of which 6 can be used as pulse width modulation (PWM) outputs) and 6 analog input pins. This was paired with the motor shield from Adafruit Industries (Figure 13 - right). The motor shield ships as a kit with two L293D h-bridges for assembly that allow a peak current of 600mA per motor. However, we replaced these with two SN74410 h-bridges to allow up to 1A of current draw per motor [11]. The motor shield also has a clocking chip to drive the PWM much faster than the 490Hz built into the native Arduino command library.

Figure 13: (Left) Arduino UNO prototyping platform (Right) Motor shield from Adafruit Industries

An single pole single throw switch was also incorporated into the design to kill power to both motors with a single emergency cutoff switch (Figure 14).

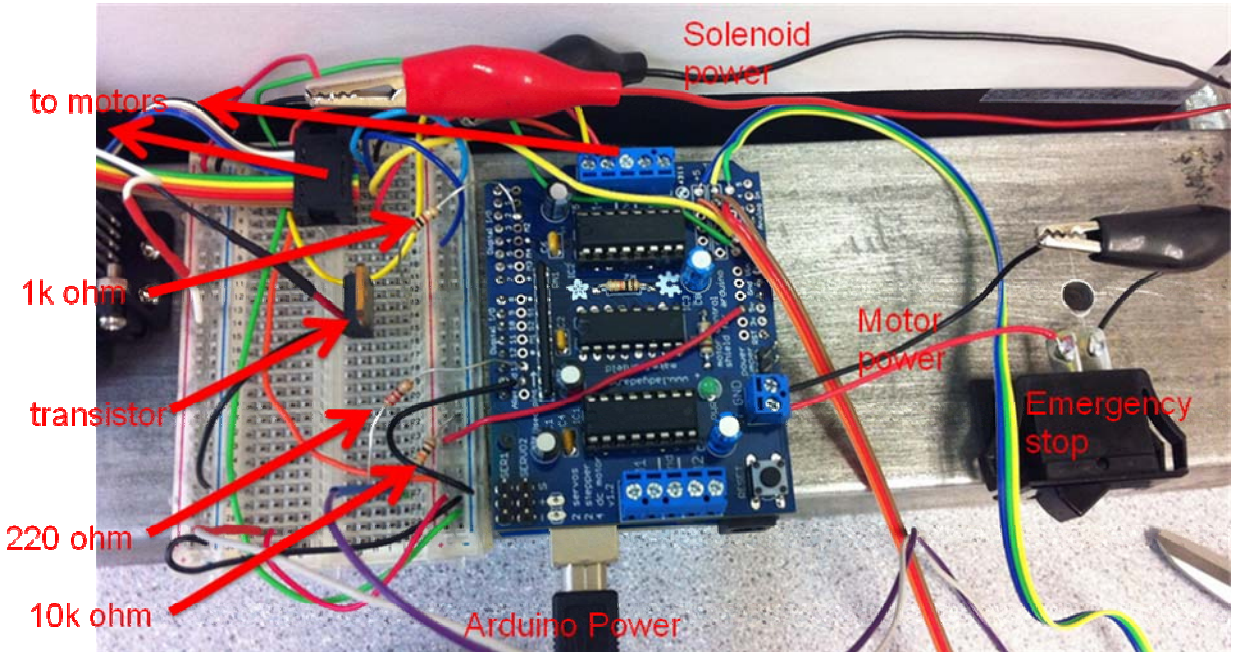

Figure 14: Actual electronics as wired

6 Controls and Software Design

We chose to use the Arduino Microcontroller instead of the BasicStamp. The Arduino has built in A2D conversion and considering our project was going to include a lot of A2D the Arduino was the better choice. There were also several shields available to easily integrate a motor controller along with an existing library of code to aid development (the AFMotor library from Adafruit Industries).

Our control mechanism is a miniature model of the real arm, referred to as the master arm. The slave robotic arm is programmed to follow the motions of the master robotic arm. We achieved

this by placing potentiometers at the joints of the master robotic arm. These provided us with position feedback and we then programmed the robotic arm to attempt to achieve that same position. The motors in the slave robotic arm provided position feedback using integrated potentiometers as well.

To get position from the master robotic arm we read the input from the two potentiometers on analog pins 4 and 5. These pins are capable of doing A2D conversion. After reading the input from these pins, we constrained them to not exceed the physical limits of our slave robotic arm. The motors have about 135 degrees of motion while the potentiometers have over 180 degrees. If the reading exceeds the allowable region, it is changed to be at the upper or lower limit. The elbow was constrained to operate only in 90 degrees because of the way the arm was constructed. After constraining the input, it is then mapped from the potentiometer reading to a degree reading. This is done using the built-in map command on the Arduino.

After getting the position of the master arm, the program retrieves the position of the slave arm. The potentiometers of the slave arm are the inputs for analog pins 0 and 1. These pins can do A2D input, and we constrain them to the physical maximum range. After that we map them to the degrees in the same manner that we mapped the master arm.

At this point we now have the position of the master arm and the position of the slave arm. The control mechanism needs to make these two positions the same as quickly as it can. To do this, we implemented the proportional (P) and derivative (D) from PID control. Using the P and D with appropriate gains, we used pulse width modulation to control the voltage sent to the motors. This affects the speed with which the motors reacted. The maximum pulse width was 255 and we found that our motors needed a minimum pulse width of 40 to move. After experimenting with the PD control, we constrained our pulse width to be between 40 and 255 at all times. We prevent the motor from trying to move when the slave and master are in the same position by saying that if the positions were within 1 degree of each other, the motors are turned off.

Finally, to control the drawing mechanism there is a solenoid attached to the end of the slave arm. This solenoid extends to draw and retracts to move without drawing. There is a button on the end of the master arm that is linked to this motion. When this button is pressed, it sends a high signal to the Arduino. The Arduino then sends a high signal out to the solenoid pin, attached to the base of a transistor, and allows our power supply to extend the solenoid. When the button is released, the solenoid retracts. See Appendix A for a full listing of the Arduino code.

7 Bill of Materials

In order to keep procurement costs down, we utilized as many existing parts as we could and supplemented these with off the shelf hardware components. Our cost for one unit was significantly lower than it would have been to buy just one of all the necessary parts since we did not have to purchase motors or several of the components to breadboard the circuit (Table 1).

Table 1: Bill of Materials (our cost)

	Vendor	Part number	Description	# in pack	Quantity	Price (1)	Subtotals (1)
Slave arm hardware	Lynxmotion	ASB-04	Aluminum Multi-Purpose Servo Bracket	2	1	$11.95	$11.95
	Lynxmotion	ASB-09	Aluminum "C" Servo Bracket with Ball Bearings	2	1	$12.90	$12.90
	Lynxmotion	HUB-08	Aluminum Tubing Connector Hub	2	2	$8.00	$16.00
	Lynxmotion	AT-04	Aluminum Tubing - 6"	1	2	$3.60	$7.20
	Adafruit	ID: 412	Solenoid	1	1	$9.95	$9.95
	Servocity	HS-7950TH	High Torque Digital Servo Motor	1	4	$119.99	$0.00
	McMaster	1630T332	Aluminum c-channel base	1	1	$15.06	$0.00
Slave arm electronics	Adafruit	ID: 81	Motorshield	1	1	$19.50	$19.50
	Sparkfun	COM-00315	SN754410 h-bridge	1	2	$2.35	$4.70
	Adafruit	ID: 50	Arduino UNO	1	1	$30.00	$30.00
	Mouser	511-TIP102	TIP102 (for solenoid)	1	1	$0.80	$0.80
	Sparkfun	COM-08589	Diode 1N4001	1	1	$0.15	$0.15
	McMaster	7395K44	Illuminated rocker switch	1	1	$7.02	$7.02
Master arm	Sparkfun	COM-09288	Rotary Potentiometer - Linear (10k ohm)	1	2	$0.95	$1.90
	Sparkfun	COM-09190	Momentary pushbutton	1	1	$0.50	$0.50
	SolidConcepts		3D printed parts	1	1	$0.00	$0.00
Other	Sparkfun	CAB-10647	Ribbon Cable - 10 wire (15ft)	1	1	$0.99	$0.99
			Misc. jumper wire, resistors, connector,etc.			$0.00	$0.00
	Art Store		Foam core, padding, sketch paper, marker			$25.00	$25.00
						Total	$148.56

7.1 Cost Analysis for Mass Production

The cost analysis for mass production was completed for 100 units (Table 2). Servomotors were identified that would provide adequate torque (90 in-oz) but minimize cost. This cost does not include the foam core backing or sketch paper.

Table 2: Bill of Materials (cost per unit at 100 units)

	Vendor	Part number	Description	# in pack	Quantity	Price (100)	Subtotals (100)
Slave arm hardware	Lynxmotion	ASB-04	Aluminum Multi-Purpose Servo Bracket	2	1	$10.76	$10.76
	Lynxmotion	ASB-09	Aluminum "C" Servo Bracket with Ball Bearings	2	1	$11.61	$11.61
	Lynxmotion	HUB-08	Aluminum Tubing Connector Hub	2	2	$7.20	$14.40
	Lynxmotion	AT-04	Aluminum Tubing - 6"	1	2	$3.24	$6.48
	Adafruit	ID: 412	Solenoid	1	1	$7.96	$7.96
	Servocity	HS-7950TH	High Torque Digital Servo Motor	1	4	$16.99	$67.96
	McMaster	1630T332	Aluminum c-channel base	1	1	$15.06	$15.06
Slave arm electronics	Adafruit	ID: 81	Motorshield	1	1	$15.60	$15.60
	Sparkfun	COM-00315	SN754410 h-bridge	1	2	$1.88	$3.76
	Adafruit	ID: 50	Arduino UNO	1	1	$25.46	$25.46
	Mouser	511-TIP102	TIP102 (for solenoid)	1	1	$0.53	$0.53
	Sparkfun	COM-08589	Diode 1N4001	1	1	$0.12	$0.12
	McMaster	7395K44	Illuminated rocker switch	1	1	$7.02	$7.02
Master arm	Sparkfun	COM-09288	Rotary Potentiometer - Linear (10k ohm)	1	2	$0.76	$1.52
	Sparkfun	COM-09190	Momentary pushbutton	1	1	$0.40	$0.40
	SolidConcepts		3D printed parts	1	1	$5.00	$5.00
Other	Sparkfun	CAB-10647	Ribbon Cable - 10 wire (15ft)	1	1	$0.79	$0.79
			Misc. jumper wire, resistors, connector,etc.			$2.00	$1.50
	Art Store		Foam core, padding, sketch paper, marker				
						Total	$195.92

8 Operation Guidelines

For general operation, use the master arm to position the slave arm in similar postures. The button at the end of the master arm controls the solenoid at the end effector of the slave arm. To

extend the drawing implement press and hold the button on the master arm, and to retract the drawing implement release the button.

8.1 Document hardware and software features to prevent damage to Arduino or other components

The digital input/output pins on the Arduino can source 40mA and sink 50mA. Staying within these limits prevents damage to the board. The two digital I/O pins we use are both protected by safety resistors. There is a 220 ohm resistor from the master arm button to the Arduino, which limits the current that can be sourced to 23mA. Another 1k ohm resistor is used to connect a digital I/O pin to the base of the transistor, limiting the source current to 5mA. In software, the arm's motion is limited to remain within the allowable range. This prevents the master from sending commands to the slave arm that it cannot perform. Without the software limits, the motors would attempt to push the arm beyond its limits causing structural failure or motor burnout.

8.2 Provide Guidelines for safe operation

To operate the arm safely, make sure you have a clear work space. The two pads on the base show where the arm will make contact with the base if driven to the extremes. Do not place anything here as it will get hit by the arm. In addition, do not swing the arm rapidly to its extremes. This causes it to hit the pads with force and over time could cause the arm to eventually break. In addition, be careful when moving the arm near the Arduino board as the wires can reach up into the arms operational region. Needless to say, it is bad to hit wires or pull them out.

8.3 Include provision for instantaneous shutdown

There is a switch that can instantaneously cut power to the motors. Use this if the robot has deviated from normal working conditions or any time an instantaneous shutdown is desired.

9 Conclusion and Future Work

In this project we created a modular and extendable robotic arm and demonstrated it's use as a drawing mechanism through closed loop master-slave control. The advantages of design include modularity and the ability to easily extend the software and hardware. A disadvantage of the current design is that the proportional controller could be better tuned and/or augmented to a full PID controller to allow for better tracking of the output vs. input signal.

We plan to give back to the open hardware community we learned from in this project by publishing a tutorial on the build on Instructables.com, and perhaps pursuing other goals such as a completed robotic assistive feeder design. The Doodlebot will also be used as a research platform in the Applied Dynamics & Optimization Lab at NYU-Poly, and can be used for outreach activities both on and off campus such as Maker Faire and in open lab events.

Appendix A

```
/*
 Robot arm control using hacked servos and a master arm with
pots as control in
 master-slave system.
```

```
 */
#include <AFMotor.h>

// Instantiate both motors
AF_DCMotor shoulder(3);
AF_DCMotor elbow(4);

// declare pins
int MasterElbowPin = A4; // the pot in the robot arm elbow
int MasterShoulderPin = A5; // the pot in the robot arm shoulder
int SlaveElbowPin = A1;
int SlaveShoulderPin = A0;
int SolenoidPin = 2;  // pin solenoid transistor is attached to
int DrawPin = 13;  // pin button for drawing is attached to

// initialize constants
const int P_Elbow = 15;    // this is the proportional gain
const int P_Shoulder = 10;
const double difconstE = 60;
const double difconstS = 60;

const int lowL = 50;  // the lower bound of speed below which
the motor won't move

const int SlaveElbowCCW = 961;      // joint limits of slave
robotic arm
const int SlaveElbowCW = 100;
const int SlaveShoulderCCW = 220;
const int SlaveShoulderCW = 750;

const int MasterElbowCCW = 743;      // joint limits of master
robotic arm
const int MasterElbowCW = 0;
const int MasterShoulderCCW = 964;
const int MasterShoulderCW = 295;

// initialize variables
float range = 1; // use 1 to avoid any divide by 0 errors

int slaveElbowAngle = 0;      // initialize all angles to 0
int slaveShoulderAngle = 0;
int masterElbowAngle = 0;
int masterShoulderAngle = 0;

int pwmElbow = 0;  // sets effective voltage level of motor
which sets speed
int pwmShoulder = 0;
```

```
int elbowError = 0;  // difference between master and slave
int shoulderError = 0;

boolean isPressed = 0; // is the drawing button pressed or not

// PID
double delT;
unsigned long  lastT;
double deltaElbow;
double deltaShoulder;
double lastElbow, lastShoulder;

void setup() {
  Serial.begin(9600);           // set up Serial library at 9600
bps

  pinMode(SolenoidPin, OUTPUT);
  pinMode (DrawPin, INPUT);

  elbow.setSpeed(pwmElbow);     // set the speed to pwmElbow
  shoulder.setSpeed(pwmShoulder);     // set the speed to
pwmElbow
}

void loop() {
  // read the values from all the pots
  masterElbowAngle = analogRead(MasterElbowPin);
  slaveElbowAngle = analogRead(SlaveElbowPin);
  masterShoulderAngle = analogRead(MasterShoulderPin);
  slaveShoulderAngle = analogRead(SlaveShoulderPin);

  // constrain master and slave to ignore out of range values
  masterElbowAngle = constrain(masterElbowAngle, MasterElbowCW,
MasterElbowCCW);
  masterShoulderAngle = constrain(masterShoulderAngle,
MasterShoulderCW, MasterShoulderCCW);
  slaveElbowAngle = constrain(slaveElbowAngle, SlaveElbowCW,
SlaveElbowCCW);
  slaveShoulderAngle = constrain(slaveShoulderAngle,
SlaveShoulderCCW, SlaveShoulderCW);

  // now map the angles so they correspond correctly
  masterElbowAngle = map(masterElbowAngle, MasterElbowCCW,
MasterElbowCW, 45, 135);
  slaveElbowAngle = map(slaveElbowAngle, SlaveElbowCCW,
SlaveElbowCW, 45, 135);
```

```
  masterShoulderAngle = map(masterShoulderAngle,
MasterShoulderCCW, MasterShoulderCW, 0, 135);
  slaveShoulderAngle = map(slaveShoulderAngle, SlaveShoulderCCW,
SlaveShoulderCW, 0, 135);

  Serial.print(millis());    // debug value
  Serial.print(" ");
  Serial.print(masterElbowAngle);    // debug value
  Serial.print(" ");
  Serial.print(slaveElbowAngle);    // debug value
  Serial.print(" ");
  Serial.println(pwmElbow);    // debug value

  // DRIVE ELBOW
  unsigned long now=millis();
  delT=double(now-lastT);
  lastT=now;
  elbowError = (slaveElbowAngle - masterElbowAngle);
  deltaElbow = (elbowError-lastElbow)/delT;
  pwmElbow = P_Elbow * elbowError + difconstE*deltaElbow;
  lastElbow=elbowError;
  pwmElbow = abs(pwmElbow);
  pwmElbow = constrain(pwmElbow, lowL, 255);

  elbow.setSpeed(pwmElbow);     // set the speed

  if ( slaveElbowAngle > (masterElbowAngle + range)) {
    elbow.run(FORWARD);      // turn it on going forward
  }
  else if ( slaveElbowAngle < (masterElbowAngle - range)) {
    elbow.run(BACKWARD);      // turn it on going forward
  }
  else elbow.run(RELEASE);      // stopped

  // DRIVE SHOULDER
  shoulderError = (slaveShoulderAngle - masterShoulderAngle);
  deltaShoulder = (shoulderError-lastShoulder)/delT;
  pwmShoulder = P_Shoulder * shoulderError;//  +
difconstS*deltaShoulder;
  lastShoulder=shoulderError;
  pwmShoulder = abs(pwmShoulder);
  pwmShoulder = constrain(pwmShoulder, lowL, 255);

  shoulder.setSpeed(pwmShoulder);     // set the speed

  if ( slaveShoulderAngle > (masterShoulderAngle + range) ) {
    shoulder.run(BACKWARD);      // turn it on going forward
```

```
  }
  else if ( slaveShoulderAngle < (masterShoulderAngle - range) )
{
    shoulder.run(FORWARD);      // turn it on going forward
  }
  else  shoulder.run(RELEASE);      // stopped

  // DRIVE SOLENOID
  isPressed = digitalRead(DrawPin);
  digitalWrite(SolenoidPin, isPressed);
}
```

References

[1] "Meal Mate Robotic Feeder | Arm Supports & Feeding | Medifab." [Online]. Available: http://www.medifab.co.nz/products/arm-supports-feeding/meal-mate-robotic-feeder. [Accessed: 19-Dec-2011].

[2] "Richardson Products Incorporated - Meal Buddy." [Online]. Available: http://www.richardsonproducts.com/mealbuddy.html. [Accessed: 19-Dec-2011].

[3] "NXT Robot Arm." [Online]. Available: http://www.nxtprograms.com/robot_arm/steps.html. [Accessed: 19-Dec-2011].

[4] "LEGO 3933-1: Olivia's Inventor's Workshop | Brickset: LEGO set guide and database." [Online]. Available: http://www.brickset.com/detail/?set=3933-1. [Accessed: 19-Dec-2011].

[5] L. Darling-Hammond, "President Obama and education: The possibility for dramatic improvements in teaching and learning," *Harvard Educational Review*, vol. 79, no. 2, pp. 210–223, 2009.

[6] "The 10 Robots That Rocked in 2010 - The Next Web." [Online]. Available: http://thenextweb.com/shareables/2010/12/31/the-10-robots-that-rocked-in-2010/. [Accessed: 19-Dec-2011].

[7] "Measuring Angst – Robotic installation « adafruit industries blog." [Online]. Available: http://www.adafruit.com/blog/2011/12/12/measuring-angst-robotic-installation. [Accessed: 19-Dec-2011].

[8] "Lynxmotion - SES 3D Models." [Online]. Available: http://www.lynxmotion.com/s-5-ses-3d-models.aspx. [Accessed: 19-Dec-2011].

[9] "Small push-pull solenoid ID: 412 - $9.95 : Adafruit Industries, Unique & fun DIY electronics and kits." [Online]. Available: https://www.adafruit.com/products/412. [Accessed: 19-Dec-2011].

[10] "Arduino - HomePage." [Online]. Available: http://www.arduino.cc/. [Accessed: 19-Dec-2011].

[11] "Arduino motor/stepper/servo control - How to use." [Online]. Available: http://www.ladyada.net/make/mshield/use.html. [Accessed: 19-Dec-2011].

[12] Mark W.Spong, Robot Modeling And Control, John Wiley & sons, Inc. 2006

[13] S. Lee, Polytechnic. Experiment 2: System Identification and Control of an Electrical Network, Institute of New York University, 2010.

[14] Richard C.Dorf , Modern Control systems, 7th edition, Addison-Wesley, 1995.