somatoBot

Design of a two link robot for demonstration of the human sensorimotor system

Team Members: Katherine Heidi Fehr & Stephanie Beth Hernández Hernández
Class: ME445, Fall 2021
Link to github repo: https://github.com/kheidi/somatoBot
Instructions to run: See README.md file in main repository.

Table of Contents

Objective

Using skills and techniques gained in ME445 design and build a small two link planar robot that demonstrates the neural control of the sensorimotor system in humans.

A two link robot was designed to represent upper limb human motor control and the different aspects that affect precision and movement. This robot will be used in outreach activities to help students understand some of the research work done in UW BADGER Lab. The robot has two links representing a human arm and forearm (with hand). The robot will be either extending or flexing the arm based on the command sent to the motors.

To introduce the students to challenges in human sensorimotor control (Figure 1) the robot operates in different modes, feedback control (Figure 2) with no velocity control, feedback control with velocity control, and sensory noise mode. Users will be able to toggle between modes using buttons and information will be displayed on an LCD.

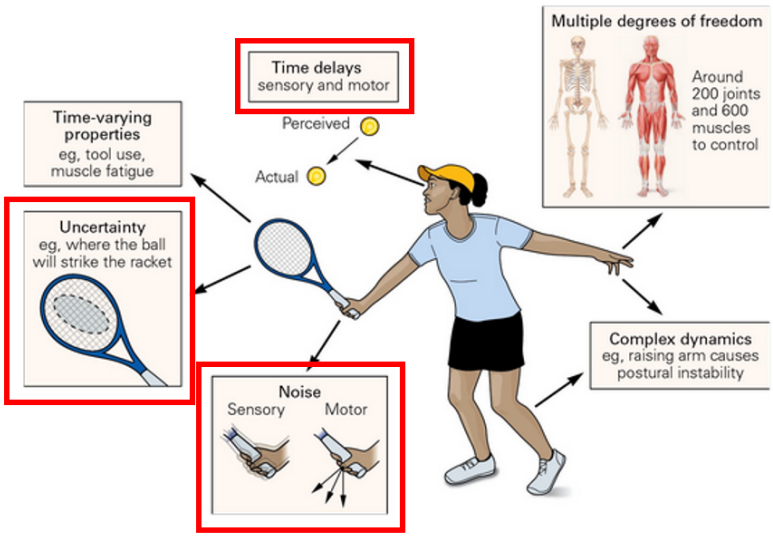


Figure 1. Challenges of sensorimotor control. Red boxes indicate potential challenges to simulate via our RR robot. Illustration adapted from Kandel et al. 2021, Principles of Neuroscience Fig. 30-1.

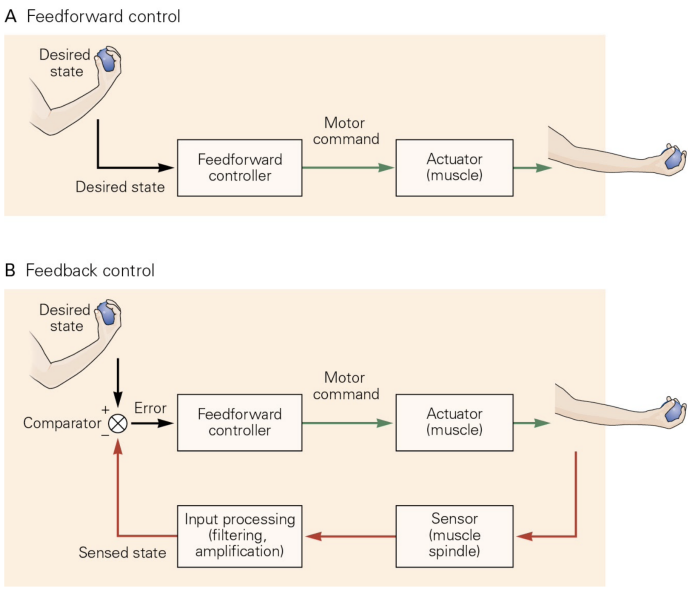


Figure 2. Feedforward and feedback control in the sensorimotor system. Illustration from Kandel et al. 2021, Principles of Neuroscience Fig. 30-2.

Goals

1. Complete physical design of robot and integrate all electrical components. That is, have all the different components wired correctly and communicate with the microcontroller.
2. Implement a feedback (closed loop) control algorithm to move our motor arms to a desired position.
3. Implement a demonstration of a potential challenge to sensorimotor control.

Robot Design

Mechanical Design

Our two link robot—designed in Autodesk Fusion 360—consists of two links, "arms" that are attached at their bases by screws that are acting as pins. On the base of each arm there is a pulley that connects each link to its corresponding motor. The motors are attached to a box that holds the components and an LCD screen. Screen captures of the CAD design can be seen below.

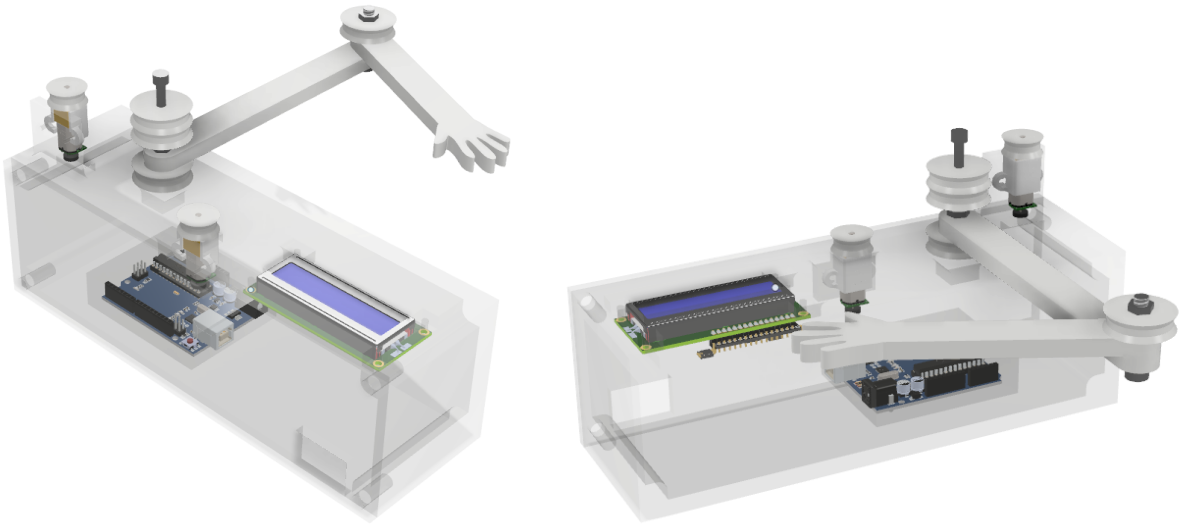


Figure 3. Conceptual CAD renderings of somatoBot.

Based on the mechanical design we calculated the mass and inertia of each link in order to size our motors accordingly. Using a MATLAB script (Appendix A), we calculated the torque needed to move our links based on different velocities. Seeing that the torque needed was low, we selected two small 75:1 micro metal gearmotor from Pololu to move our robot arms.

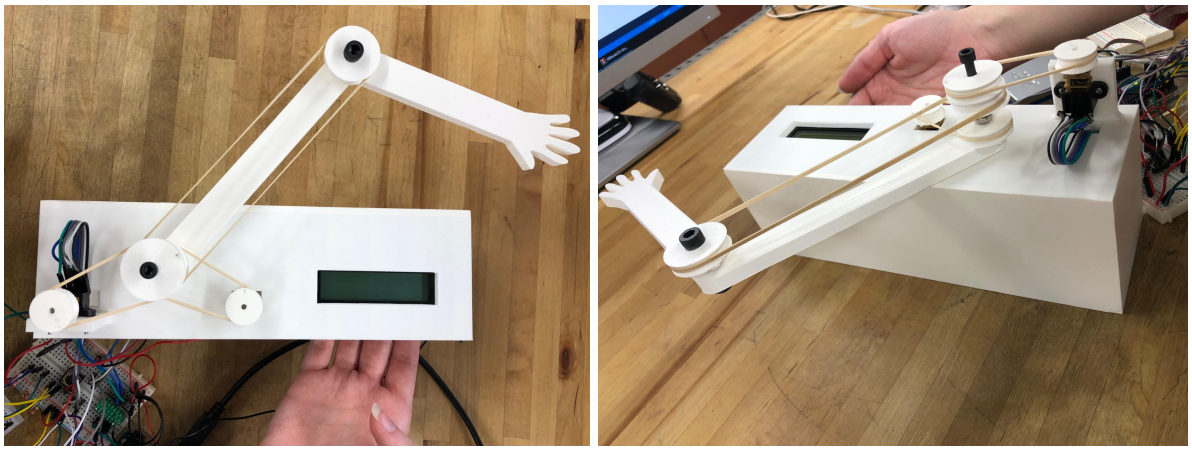


Figure 4. Assembled somatoBot.

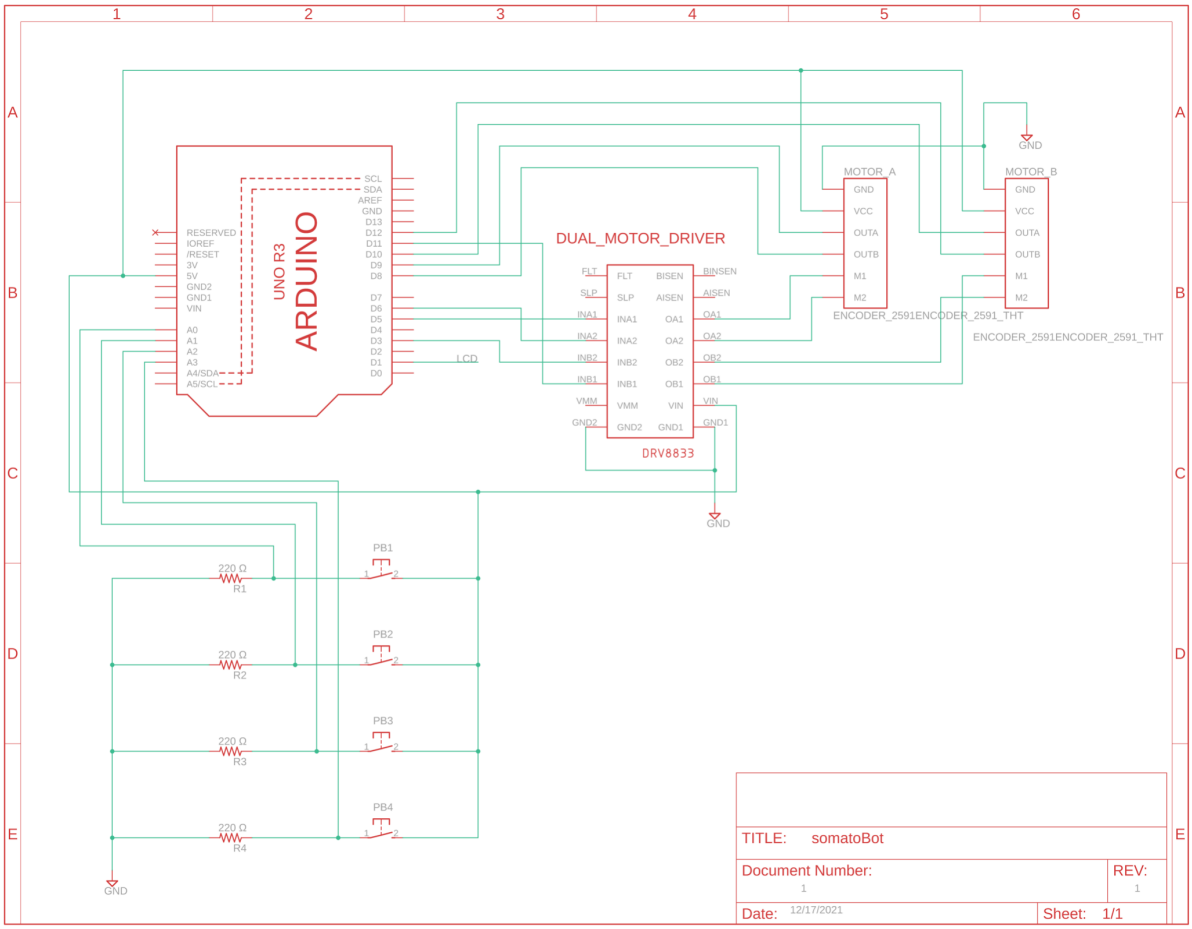
Electronic Hardware Design

Bill of Materials (BOM)

Part Number:	Description:	Quantity:	Price:	Total Price:
782	LCD	1	$24.95	$24.95
2598	Encoder	2	$9.95	$19.90
2130	Dual Motor Driver	1	$6.95	$6.95
759	Jumper Cables	1	$3.95	$3.95
A000066	Uno Microcontoller	1	$23	$23
3064	Motor	2	$17.95	$35.90
64	Bread board	1	$5.00	$5.00
367	Buttons	1	$2.50	$2.50
2780	Resistors	1	$0.75	$0.75
			Total:	**$122.90**

Schematic

Schematic below shows connection between the Arduino, dual motor driver and encoders that are connected to the motors. Not shown is the LCD, see Appendix D for more details.



Software Design

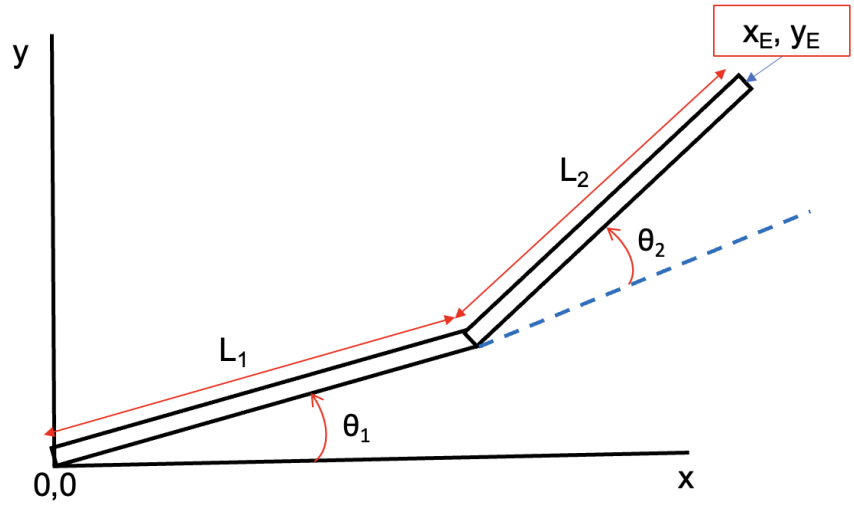
Motor Control

Trajectory Calculations

An inverse kinematics analysis was performed to determine the position of the links to reach an endpoint. The geometric solution approach was used to determine the angles each link had to be at to reach the target point. Equation 1 and 2 were coded to move the links (Refer to Appendix C for detailed calculation). To send this information to the motor, we converted the result to degrees and knowing that a full revolution of the motor was 900 counts we calculated a ratio that allowed us to only move the motor the degrees we wanted each link to move based on encoder counts.

$$\theta_2 = cos^{-1}\left(\frac{x_E^2 + y_E^2 - L_1^2 - L_2^2}{2*L_1*L_2}\right) \quad (1)$$

$$\theta_1 = tan^{-1}\left(\frac{y}{x}\right) - tan^{-1}\left(\frac{L_1*sin(\theta_2)}{L_1 + L_2*cos(\theta_2)}\right) \quad (2)$$



Code:

```
struct theta Trajectory(float x, float y)
{
   float l1, l2, theta1Rad, theta2Rad; // variable will be time dependent
   float temp;
   struct theta mytheta;

   //circular trajectory
   l1 = 0.153; //[m]
   l2 = 0.140; //[m]

   // Inverse Kinematics (wrist down) - thetas in radians
   theta2Rad = acos(((x*x)+(y*y)-(l1*l1)-(l2*l2))/(2*l1*l2));
   theta1Rad = atan(y/x) - atan((l2*sin(theta2Rad))/(l1+(l2*cos(theta2Rad)))) ;
```

```
   // Radians to angle conversion:
   mytheta.theta1 = (theta1Rad*(180/PI));
   mytheta.theta2 = (theta2Rad*(180/PI));

   return mytheta;

}

int AngleToCountsConversion (float theta)
{
    float percentage;
    int theta_counts;
    percentage = theta/360;
    theta_counts = percentage*900;

   return theta_counts;
}
```

PWM

Four PWM pins on the Arduino Uno were used to run the motors through a dual motor driver. These pins were controlled with the 8-bit timers timer0 and timer2 using the non-inverted fast PWM mode. Code was written to run the motors either clockwise or counterclockwise depending on the command and the PWM output pin. The PWM timers were initialized with a prescale value of 1. The function input mapped the desired percent of max speed to the corresponding voltage which resulted in the PWM duty cycle percent.

Code:

```
void runMotor(long percentMaxPower, int motorID, int direction)
{
   int sendToMotor;
   int speed;
   sendToMotor = (int)map(percentMaxPower,0,100,153,170); //Converts the percent power
to a value from 0 255 for the 8-bit timers

   // ------- MOVE MOTOR A
   if (motorID == 0) //
   {
       if (direction == 0) //Port D, pin 6 set duty cycle, direction A
       {
           for (speed = 0; speed < sendToMotor; speed++)
           {
               OCR0A = speed;
               OCR0B = 0;
               delay_us(delayTime);
           }
       }
       if (direction == 1) //Port D, pin 5 set duty cycle, direction B
       {
           for (speed = 0; speed < sendToMotor; speed++)
           {
               OCR0A = 0;
               OCR0B = speed;
               delay_us(delayTime);
```

```
            }
        }

    // ------- MOVE MOTOR B
    } else if ((motorID == 1) != 0) // Move motor B
    {
        if ((direction == 0) != 0) //Port B, pin 3 set duty cycle, direction A
        {
            for (speed = 0; speed < sendToMotor; speed++)
            {
                OCR2A = speed;
                OCR2B = 0;
                delay_us(delayTime);
            }
        }
        if (direction == 1) //Port D, pin 3 set duty cycle, direction B
        {
            for (speed = 0; speed < sendToMotor; speed++)
            {
                OCR2A = 0;
                OCR2B = speed;
                delay_us(delayTime);
            }
        }
    }

}
```

Encoder Counter

To measure the rotation of the motors we used a 12-bit encoder. Using a pin change interrupt we were able to record the encoder pulses as the motor spun using the code below. To update the state of the global encoder counters we used a state change table to keep track of the direction of the pulses.

State Change Table			
0	-1	1	0
1	0	0	-1
-1	0	0	1
0	1	-1	0

Code:

```
// Pin change 8-14 interrupt service routine
interrupt [PC_INT0] void pin_change_isr0(void)
{
    // Only the following PCINTs are enabled
    // as they are where the encoders are connected.
    //
    // PCINT0 / D8 / PB0
    // PCINT1 / D9 / PB1
    // PCINT2 / D10 / PB2
    // PCINT4 / D12 / PB4
```

```
    // Adapted from encoderCounter.ino by Alex Dawson-Elli

    unsigned char changeIA = 0;
    unsigned char changeIB = 0;
    unsigned char changeJA = 0;
    unsigned char changeJB = 0;

    //history and current pin terms
    unsigned char PINBcurrent = PINB & 0b00010111; //grab only the relevant pins
    unsigned char changedbits = PINBcurrent ^ PINBhistory;

    //mask bits
    char motorAMask = 0b00000011;
    char motorBMask = 0b00010100;

    //read and update Crank
     if(changedbits & motorAMask) //something has changed in Crank's state
    {
        motorACount += stateChangeTable[(PINBhistory & motorAMask)][(PINBcurrent &
motorAMask)];
    }

    //read and update RightPedal
    if(changedbits & motorBMask) //something has changed in RightPedal's state
    {
        changeIA = ((PINBhistory & motorBMask) & 0b00000100) >> 2;
        changeIB = ((PINBhistory & motorBMask) & 0b00010000) >> 3;
        changeJA = ((PINBcurrent & motorBMask) & 0b00000100) >> 2;
        changeJB = ((PINBcurrent & motorBMask) & 0b00010000) >> 3;

        motorBCount += stateChangeTable[(changeIA | changeIB)][(changeJA | changeJB)];
    }

    PINBhistory = PINBcurrent;

}
```

Control Laws

Basic Control (Constant Velocity) - Normal Mode:

A very simple control law was used to move the motors to the target position. This control consisted in comparing the encoder counts to the counts it takes the motor to reach the target position. We would command the motors to run until that position was reached. For mode 1, theta1_counts was based on the inverse kinematics of the robot, where we aimed to move the links to that x and y end point. In mode 2, we had motor A run two full revolutions to be able to compare how different it is to the PD (E on Error) controller.

```
    if (mode == 1 || mode == 2)
    {
         if ((abs(motorACount))<theta1_counts)
         {
            runMotor(70, MOTOR_A, CCW);
         }
         else
         {
             StopMotorA();
         }
         if(abs(motorBCount)<theta2_counts)
```

```
        {
            runMotor(40, MOTOR_B, CW);
        }
        else
        {
            StopMotorB();
        }
    }
```

PD (E on Error):

To test the PD (E on Error) controller we had motor A move two full revolutions (theta1_counts = 900). We iterated the proportional and derivative gain until the motor rotated and stopped at two revolutions, these gains resulted in 0.9 and 0.15 respectively. The resulting manipulation was sent to the motor to control its velocity. We could see that the speed of the motor would keep reducing until it reached the two revolution target.

```
delta_t = 0.001;
kp_1 = 0.9;
kd_1 = 0.15;
k0_1 = kp_1 + kd_1/delta_t;
k1_1 = - kd_1/delta_t;
k0_1 = flt2fxd(k0_1);
k1_1 = flt2fxd(k1_1);
if (mode == 3)
    {
        e1 = theta1_counts - (motorACount); // our end position was calculated with IK
        m1 = kp_1*e1;
        e1 = theta1_counts - motorACount; // our end position was calculated with IK
        m1 = ((long)k0_1*e1 + (long)k1_1*e1_prev)>>7;

         if (m1>100)
         {
             m1 = 100;
         }else if (m1<=0)
         {
             m1 = 0;
         }

         runMotor(m1, MOTOR_A, CW);
         e1_prev = e1;

    }
```

Fixed Point:

A function was made to calculate the Q-point and fixed point for the gains used in the PD (D on Error) controller. With these values the following calculations were done to command the motor using fixed point:

$m_n = k_0 * e_n + k_1 * e_{n-1}$ where $k_0 = k_1 = Q\,9.7$, $e_n = e_{n-1} = m_n = Q\,16.0$

$$Product\ 1:\ k_0 * e_n = Q\ 25.7$$
$$Product\ 2:\ k_1 * e_{n-1} = Q\ 25.7$$
$$m_n = ((long)k_0 * e_n + (long)\ k_1 * e_{n-1}) >> 7$$

Code:

```
int flt2fxd(float x){
   int QI;
   int WL;
   int QF;
   int fixedPoint;

   WL = 16;

   QI = floor(log(fabs(x))/log(2)+2);
   QF = WL-QI;
   fixedPoint = (int)(x* pow(2,QF));

   return fixedPoint;

}
```

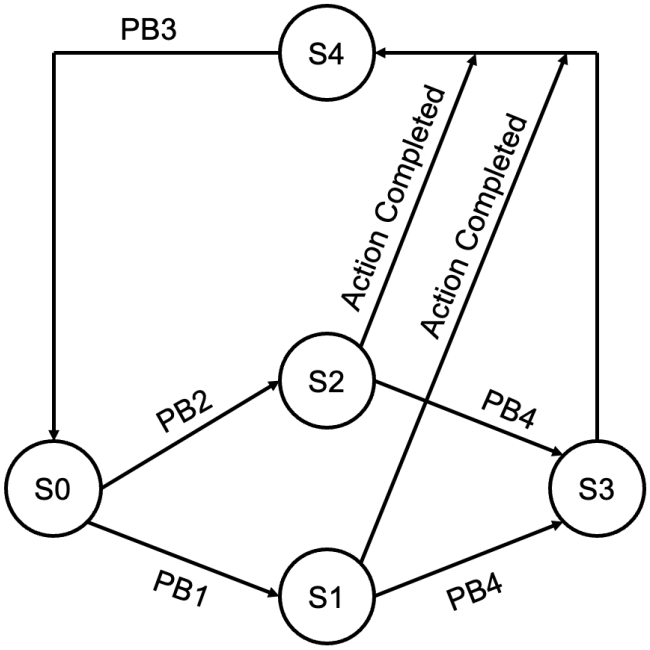
(In process) Noise Mode:

A rough idea of a system where noise is added was coded. The idea was to add 10 counts to the number of encoder counts the code was reading to confuse the motor and for it to not know exactly where it is. This idea is still under development since the code doesn't work with the way we have our control laws.

Preliminary Code:

```
void NoiseMode (int LOCALmotorACount, int LOCALtheta1_counts, int LOCALmotorBCount,
int LOCALtheta2_counts)
{
   if (abs((LOCALmotorACount+10))<LOCALtheta1_counts)
   {
       runMotor(20, MOTOR_A, CCW);
   }
   else
   {
       StopMotorA();
   }
   if((LOCALmotorBCount+10)<LOCALtheta2_counts)
   {

       runMotor(20, MOTOR_B, CW);

   }
   else
   {
       StopMotorB();
   }
}
```

State Machine



S0: Ready to Start
S1: Normal Mode
S2: Noise Mode
S3: E-Stop
S4: Move arm to initial position

A state machine was added to be able to change between modes, add an emergency stop and inform the user that the robot needs to be moved to its “home” or initial position to be able to accurately move. State 0 is the initial state where all the variables are initialized to zero and the robot is waiting to know what it has to do. When the first button is pressed the robot runs at normal mode and when the second button is pressed it runs in noise mode. At any point, when executing the modes, the fourth button can be pressed and it will stop the system (E-stop). Lastly, when it enters state 4, a message will display telling the user that it needs to move the robot to its initial position and to press button 3 when it's located at that position to let the system know it is ready to run again.

Code:

```
//States:
S0 = (S0 || (S5&&BP3)) &&!S1 &&!S2;
S1 = (S1 || (S0&&BP1) &&!BP4) &&!S3;
S2 = (S2 || (S0&&BP2) &&!BP4) &&!S3;
S3 = (S3 || (S1&&BP4) || (S2&&BP4)) &&! S0;
S5 = (S5 || (S2&&ActionCompleted) || (S1&&ActionCompleted)) &&!S0;

// Outputs:
ReadyToStart = (S0)&&!S1&&!S2&&!S3&&!S4;
Normal = (S1)&&!S0&&!S2&&!S3&&!S4&!ActionCompleted;
Noise = (S2)&&!S0&&!S1&&!S3&&!S4&!ActionCompleted;
EStop = (S3)&&!S0&&!S1&&!S2&&!S4;
Restart = (S5)&&!S0 &&!S1 &&!S2 &&!S3 &&!S4;
```

Serial LCD

In order to limit the amount of pins needed in our design we used an Adafruit custom-designed PCB that attached to the back of our standard 16x2 LCD screen (schematic in Appendix D) to

allow us to communicate via serial. We developed custom functions to send packets over the TX pin using USART in the corresponding formats. The serial "backpack" would wait to receive "`0xFE`" and then another 8-bit character corresponding to the mode—examples as follow:

- 0xFE to start print following characters to the display
- 0x99 to change brightness to the following 8-bits delivered
- 0xD0 to change the color where the next 3 chars correspond to the red-green-blue hex codes.

Video demonstration of LCD: https://tinyurl.com/somatoBotLCD

Challenges & Future Work

During this project we overcame many challenges and learned a lot including:

- How to set up an Arduino uno to debug with debugWIRE
- How to use PWM as an alternative to a DAC. Learned how to use built in PWM provided by the Atmel chip
- How to communicate with an LCD via serial (USART)
- How to perform an inverse kinematic analysis for a two link robot
- How to set up an entire system from scratch!

Challenges still to overcome through future work

- Physical design of the robot—pulleys need to rotate more smoothly, likely replacing the elastics
- Unexpected malfunction of the LCD
- The ability to read encoders while running a state machine—likely an alternative encoder counter technique is needed
- Additional modes are needed for somatoBot to be a more educational tool
- Modify control laws to function with the different modes

Relevance to Course Content

Week	Lecture Material	Lab Material	Our Project	Done?
1	Passive Circuits, Filters, & Op-Amps	Op-Amp Signal Conditioning	Our project will require a circuit (TBD). We will also be using the oscilloscope for testing and development.	Yes
2	Transistors, Solid State Devices, & Relays	Biasing a Transistor and Driving A solenoid	We need to power our servo motors and will use lessons learned from this lab. (data sheets)	Yes
3	Processors, C Programming Fundamentals	Programming & Debugging Environment	We will use a similar Atmel chip for our process and use Atmel studio, the JTAG debugger (planning on borrowing from the mechatronics lab) and write our code in C.	Yes

4	C Programming Fundamentals, Digital I/O, Digital Logic & Computer Math	MCU Input/Output & Solenoid Driver	We will use digital i/o and logic in our project	Yes
5	Interrupts, Timers, & Advanced C Concepts	Timers, Interrupts, & Data structures	Driving our motors and state machine will use interrupts and timers.	Yes
6	Stepping Motors, Fixed Point DDA	DDA Technique Stepper Motor		
7	State Transition Diagram Techniques & Keypad/Displays or HMI/LCD	Keypad Driver & Display or HMI State Machine	Buttons paired with a state machine will be used to allow the user of our demo to switch between modes. Our robot will have a small LCD to let the user know its current state.	Yes
8	Midterm Exam	Hydraulic/Pneumatic Systems (Program a Press)	While our project will not incorporate hydraulics, the week 8 lab also focused heavily on the development of the state machine.	Yes
9	A/D, Sampling Theory, D/A, Reconstruction	A/D-D/A Lab with Aliasing		
10	Encoders, Encoder Counters, LVDTs	Encoder counter Driver lab - Encoder to LCD	We will use encoders to mimic proprioception in our model (our “arms” need to have an awareness of their position in space)	Yes
11	Communication Interfaces	Communication with a PC – Telemetry Logging	Used serial communication to communicate with the LCD	Yes
12	Close Loop Position Servo Control	Motor Control Lab with Data Logging	We will directly be programming servo motors to move our robots arms using techniques learned in class.	Yes
13	Fixed Point Math	Fixed Point Motor Controller	Will use fixed point math in the control code.	Yes
14	PWM, H-Bridges, Wireless, RTOSes, etc.		Will use PWM to communicate with the motor controller.	Yes
15	Project	Project	Project	Yes

Appendix A: Torque Estimates Matlab

Contents

Somato-Bot Torque Estimates

```
close all; clear;
```

Physical Properties

```
I_L1_o = 2.038E+05*(10^-9); %Izz at origen (kg m^2)
m1 = 30.263/1000; %kg

I_L2_o = 1.362E+05*(10^-9); %Izz at origen (kg m^2)
m2 = 29.988/1000; %g

l_L2 = 190.755/1000; %m
```

Acceleration

```
t = 0.2:0.01:3; %s
theta_travelled = degtorad(180); %rad
alpha = (2*theta_travelled)./(t.^2); %rad/s2

% Average speed in RPM
RPM = ((.5*60)./(t));
```

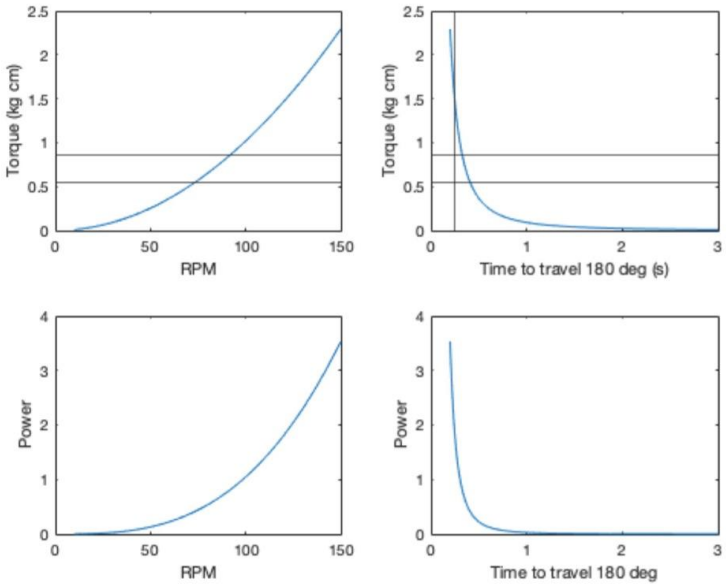
Inertia

```
I_sum = I_L1_o + I_L2_o + m2*(l_L2^2); %kg m^2
```

Minimum torque and power estimates

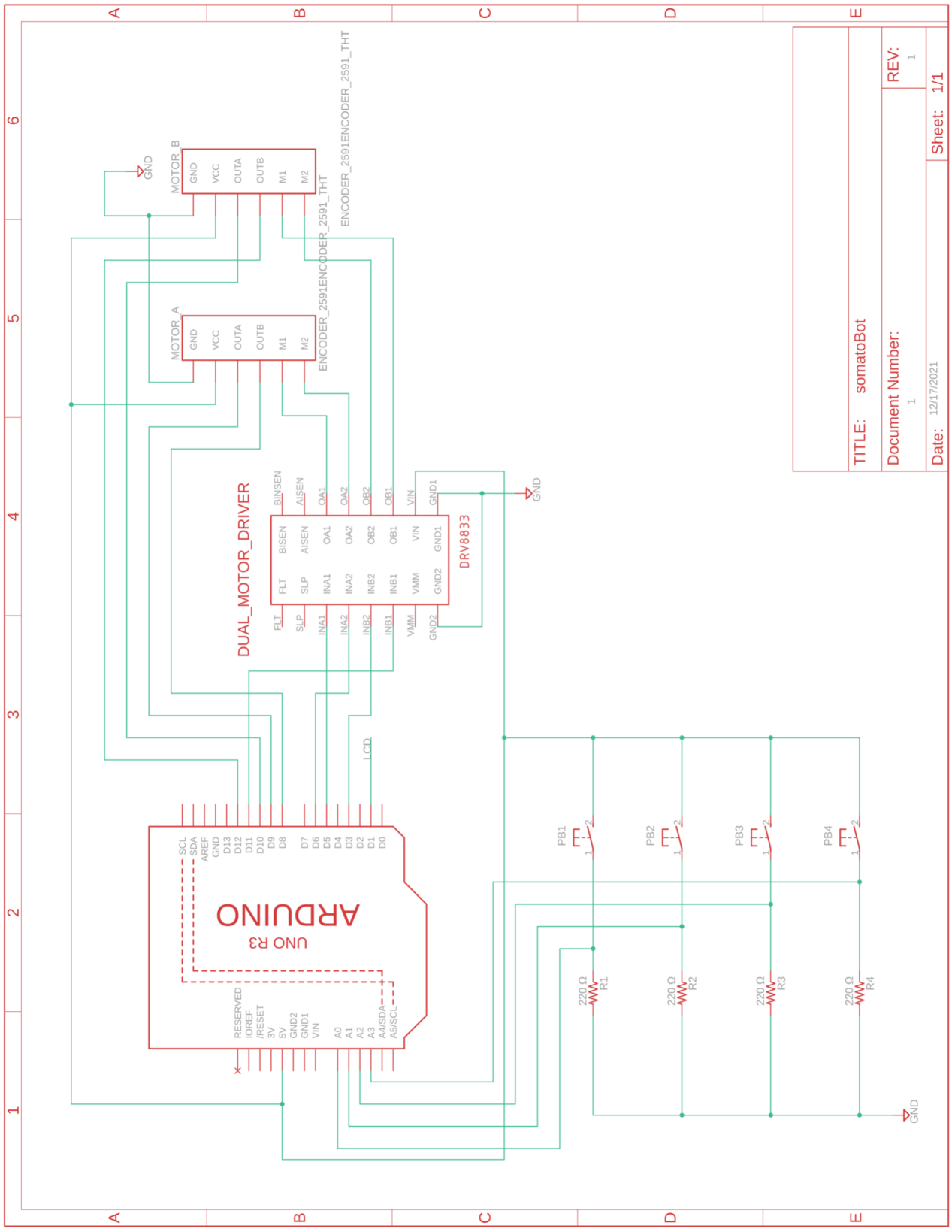
```
torque = (I_sum.*alpha); %Nm
power = torque.*(theta_travelled./t); %W

torque = torque * 10.197162129779; %convert to kg cm
figure
subplot(2,2,1)
plot(RPM,torque)
xlabel('RPM')
ylabel('Torque (kg cm)')
yline(0.54)
yline(0.86)
xline(0.25)

subplot(2,2,2)
plot(t,torque)
```

```
xlabel('Time to travel 180 deg (s)')
ylabel('Torque (kg cm)')
yline(0.54)
yline(0.86)
xline(0.25)

subplot(2,2,3)
plot(RPM, power)
xlabel('RPM')
ylabel('Power')

subplot(2,2,4)
plot(t, power)
xlabel('Time to travel 180 deg')
ylabel('Power')
```



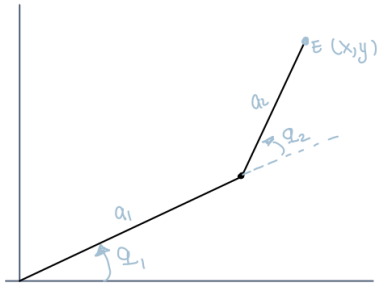
Published with MATLAB® R2021a

Appendix B: Schematic Large



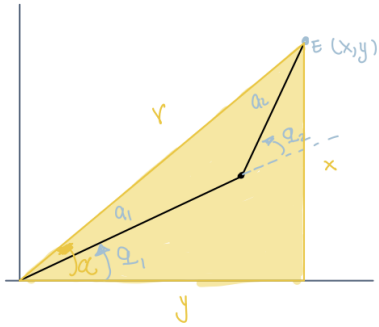
Appendix C: Inverse Kinematics

Inverse kinematics of a 2-joint robot arm using geometry:



} original problem.

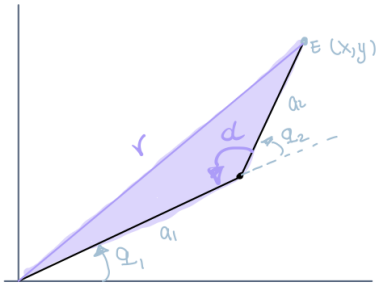
①



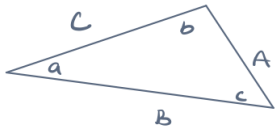
$r^2 = x^2 + y^2$ → pitagora.

$\alpha = \tan^{-1}\left(\frac{y}{x}\right)$

② lets get q_2:



using cosine law we can get α.



$A^2 = B^2 + C^2 - 2BC\cos a$

$\Rightarrow \quad r^2 = a_1^2 + a_2^2 - 2a_1a_2\cos d$

$x^2 + y^2 = a_1^2 + a_2^2 - 2a_1a_2\cos\alpha$

$\cos\alpha = \frac{a_1^2 + a_2^2 - x^2 - y^2}{2a_1a_2}$

$\cos q_2 = \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1a_2}$

↳ recall $r^2 = x^2 + y^2$

↳ from the shaded triangle we can see that:

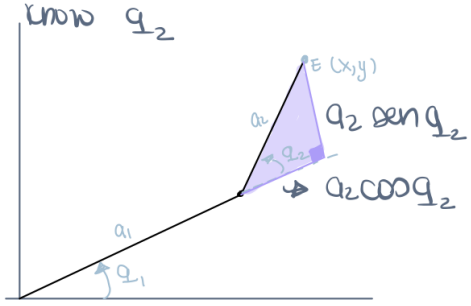
$\alpha + q_2 = 180^\circ$

$\alpha + q_2 = \pi$

$q_2 = \pi - \alpha$

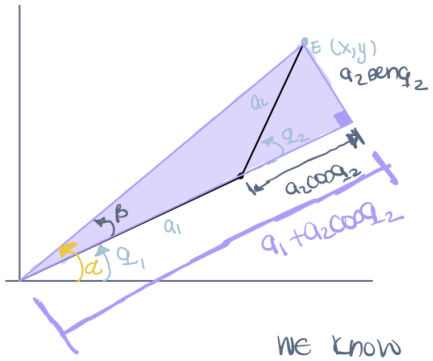
↳ $\cos q_2 = -\cos\alpha$

③ We know q_2

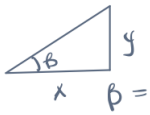


$$\operatorname{sen} q_2 = \sqrt{1-\cos^2 q_2}$$

④



to get β:



$$\beta = \tan^{-1}\left(\frac{y}{x}\right)$$

$$\beta = \tan^{-1}\left(\frac{a_2 \operatorname{sen} q_2}{q_1 + a_2 \cos q_2}\right)$$

We know α from Step #1:

$$q_1 = \alpha - \beta$$

$$\Rightarrow q_1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{a_2 \operatorname{sen} q_2}{q_1 + a_2 \cos q_2}\right)$$

Appendix D: Adafruit custom LCD serial "Backpack"

Retrieved from:
https://cdn-learn.adafruit.com/assets/assets/000/004/522/original/lcds___displays_usblcd.png?1396813921