Advanced Mechatronics Project 2: 2-Revolute Manipulator Kinetic Sand Table

Greta Perez-Haiek, Aadhav Sivakumar, Nathan Smurthwaite

April 3rd, 2025

1 Abstract

The "Sisyphus Table," Created by Sisyphus Industries, is a mesmerizing coffee table that can enhance the aesthetics of any living space. The table works by magnetically driving a small ball around in sand (using a prismatic-revolute manipulator), drawing out intricate patterns as part of an evolving art piece. This project seeks to replicate this relaxing art installation using Parallax's Propeller as the micro-controller and Two-Revolute Joints (with a strong magnet at the end-effector) as the Manipulators responsible for driving the ball around. Through programming techniques and electronic components, we developed a functional, interactive version of a Kinetic Sand Table that maintains the spirit of the original table while enabling the user to draw out their own shapes in the sand using a joystick and LED Matrix as an interface. This report goes through the design, implementation, and challenges faced during the development process, including all of the displays, cog optimization, and construction. The final demonstration of the project showcases the functionality of all sub-systems, resulting in an entertaining and transportable Kinetic Sand Table that both accepts a user's own drawings as a design, or generates one on it's own.

2 Market and History

Figure 1: Sisyphus Table, by Sisyphus Industries

In the art space, art installations involving pattern-making and sand are everywhere. In the commercial market, however, most notable one is the "Sisyphus table" by Sisyphus industries (See Figure 1). This table is one of many products that they sell, including furniture and tabletops, that has a pattern making system within driven by a marble in a bed of sand. This pattern can be seen evolving in real time by a glass viewing window onto, making it a beautiful

centerpiece for any home. The naming of the table alludes to the ancient Greek Myth of Sisyphus, who was cursed for all eternity to roll a boulder up a hill, only to watch it roll down to the bottom when he reaches the top. The marble ball, rolling for "all eternity" around and around the sand-bed makes the name quite a fitting and interesting reference. These beautiful tables, however, are expensive (and out of reach to the average consumer): the cheapest table on Sisyphus Industries's website (https://sisyphus-industries.com/shop/) is around 700 dollars, and the price could go upwards to 15,000 dollars. The purpose of this project is to manufacture a Kinetic Sand Table that could create equally beautiful designs at a small fraction of the price.

3 Components

3.1 Joysticks

For this project, there are only 1 input being considered (in the form of a joystick module).

Figure 2: Singular joystick module

This module requires a 5V power supply as well as a ground for inputs. There are two analog outputs for the signal, one for the movement in the x axis and one for the movement in the y axis. There is also a digital output, which is a pull-down switch, leaving the voltage floating when the joystick isn't pressed in and pulling it to ground when it is. In the circuit there is a large pull-up resistor connected to the pin such that the voltage isn't floating when unpressed.

3.2 LED Matrix

We initially tried to replicate the protocol necessary to control each of the 256 WS2812Bs that are a part of the matrix. This required precise timing as specified in its datasheet. When attempting to get the matrix working for the first time, there was immediately a problem. An initial approach assigned each LED a corresponding index in an array, with each space in the array containing an object that contains red, green, and blue values for each pixel. However, the sequence goes down the matrix in a snake pattern, instead of reaching the end of the row and restarting at the left side on a new row. To fix this, a map

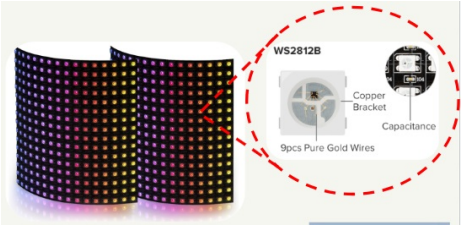


Figure 3: Initial LED matrix

transform was implemented, numbering each pixel in the snake pattern with a new number representing where it would be in a standard matrix setup (row by row). This allowed us to use a separate matrix to represent the whole board and then put that information through this new mapping matrix to ensure the display shows up as intended.

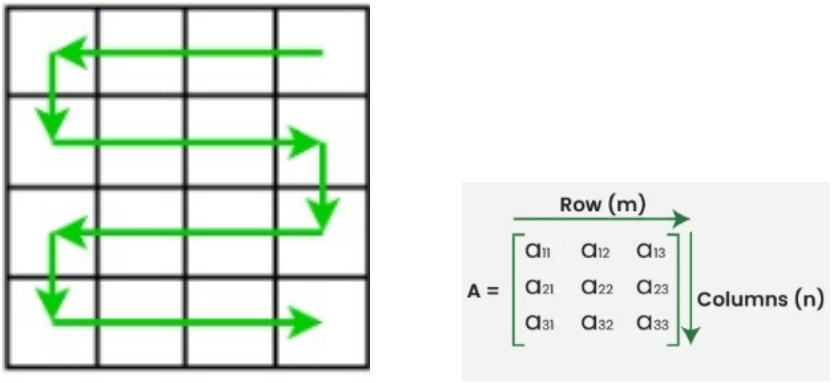


Figure 4: Original traversal sequence and ideal matrix

Creating a 16x16 matrix to represent the board initially caused some problems because the original datatype used to represent each pixel was an int. This effectively instantiated 256 ints for the board and another 256 ints for the mapping matrix, which caused some memory issues. Swapping out the integers with unsigned chars significantly reduced the amount of memory used.

3.3 Motors

The motors we had access to are 28BYJ stepper motors, controlled by the ULN2003 motor controller. We decided not to use servo motors because the standard ones have a range of only 270 degrees, which doesn't work for our purposes. A regular DC motor also wouldn't work since we don't have an encoder that has degree precision. Stepper motors seem to be the best choice because they have the best precision, even if they don't move as fast.

We controlled the motors with the half step method, which allows a precision of 4096 different angles. We used two of these motors to create a functional 2R planar manipulator.

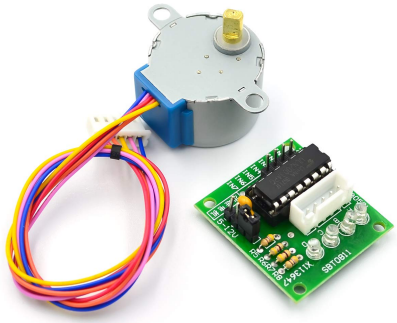


Figure 5: Stepper motor with controller

4 Programming

4.1 MATLAB

In order to ensure that the movements of the motors actually worked, we simulated it in Matlab. This simulation shows in real time where the end effector of the robot manipulator would be in real time.

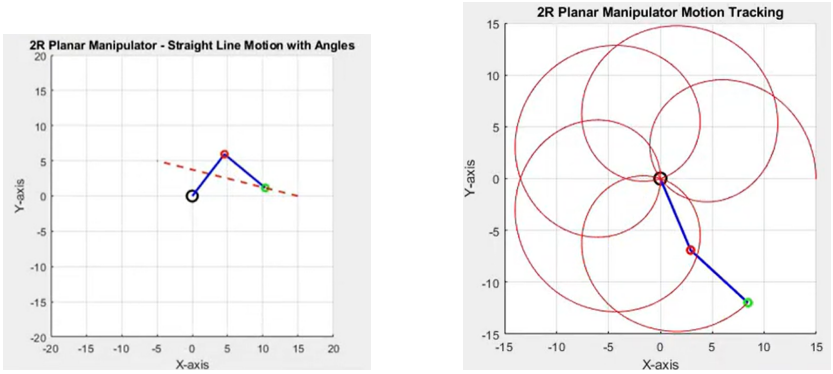


Figure 6: Screenshots of the results of the simulation.

4.2 State Machine

We decided to use a state machine to keep track of the current mode of operation, and used 5 distinct states. In order to move between states, you have to hold the joystick button down for more than one second.

4.2.1 ManualThetas

ManualThetas is the default mode of operation, and allows the user manually control the angle of each motor individually. It's mainly used for resetting the arm to the starting position after a run, ensuring that the straight line or that pattern being drawn is more accurate. Pressing down the joystick button resets where the starting point is in the code, x=15cm and y=0cm, so you would have to manually move it to that position before doing anything else.

4.2.2 StraightSelection

This mode allows you to select a point for the arm to move to, shown in green. Each dot represents 2cm, and the exact point is at the center of the four dots. The current point is represented by the purple dots, while the desired point is represented with a cluster of green dots. pressing the joystick briefly sends the command to move, and changes the state to StraightLines.

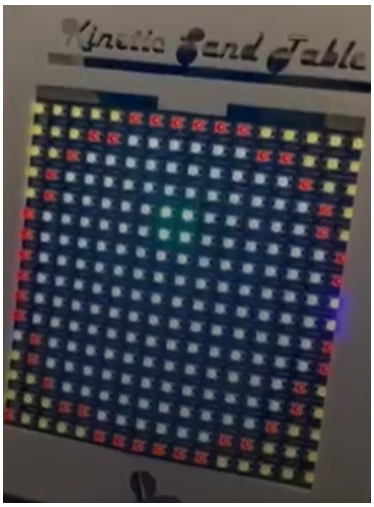


Figure 7: Purple on the right is current position, green is where it will move

4.2.3 StraightLines

This mode draws a straight line from where the end effector currently is to where it's selected to be. It does this by calculating 100 different points along the path, and incrementally moving the arms there one by one by making tiny changes in their degrees. This calculation takes a few seconds, and the arm starts moving after it's finished. After the arm moves and the line is drawn, it automatically goes back to the StraightSelection state and updates the current x and y position.

Figure 8: Straight line drawn from the right side

4.2.4 PatternSelection

This mode allows you to select what kind of pattern you want to draw. The left bar is the delay time between moves of the first motor in ms, and the right controls the outer motor. This makes it slightly counterintuitive, because a smaller bar is connected to a faster speed and vice versa. The ratio between these two numbers determines what kind of pattern is drawn, and at what speed. Pressing the joystick sends the command to start drawing the patterns, and changes the state to CircularPatterns.

Figure 9: Setting the speeds, motor 2 will run twice as fast

4.2.5 CircularPatterns

Once the pattern variables are selected, the motors continuously turn at their pre selected speeds. If the outer motor spins twice as fast, it creates 2 circles; 3 times as fast and it creates 3 circles and so on. There are also some ratios that aren't whole numbers, which will result in the second arm not fully returning to the starting position/completing the pattern. Due to the wires being constraining, it's limited to one full rotation of the inner motor. After the run, it will automatically switch back to the PatternSelection state.

Figure 10: Pattern drawn with 3 circles

5 Construction

The base of the manipulator is made out of one solid block, with a cutout so the first motor can be placed in without rotating. The manipulator base has been manufactured using 3D printing, along with the first and second manipulator links.

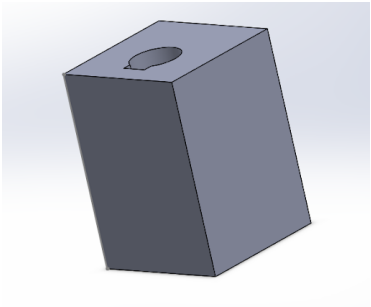


Figure 11: Base of the manipulator

The first link of the manipulator has a counterweight that goes beneath it's own plane to offset the weight of the rest of the arm. It also has a slot for the second motor to go in.

The second link of the manipulator also has a counterweight, but this time it's above the plane of the link so that it doesn't interfere with the previous one. It also has a space for the magnet to slot in.

The housing, which as designed using Fusion 360, was cut out of 1/4th inch wood. Designed to replicate the shape of a table, it consists of 8 side parts 6 inches tall as well as an octagonal base piece with a 14 inch inner diameter. The size was intentional, as the manipulator arm has a 6 inches reachable workspace

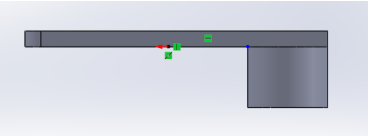


Figure 12: First link of the manipulator

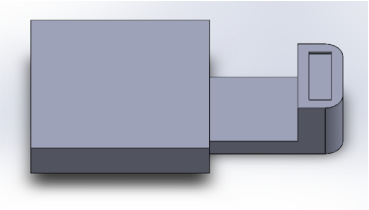


Figure 13: Second link of the manipulator

radius from the center of the octogon. The border is marked by a raised circular barrier (which is also used to prevent sand from spilling off the sides of the table). The top of the housing is made of 1/8 inch cardboard, and was found that it performed better at enabling magnification between the ball and the magnet due to it's thinness. A 1/4 inch wooden table top was attempted, but the ball did not behave as smoothly. It would be better to replace the top with acrylic to prevent friction between the ball and the table top... however, only 1/4 inch acrylic was found, which was too thick for the purpose at hand.

The table was assembled using wood glue and precisely slotted together using the toothed edges of all of the pieces. Sand was sprinkled on top to mark the design generated from the ball (controlled by the manipulator arm). If a user wants to control the ball, they have the option to do so either manually, generate a straight line calculation, or generate a pattern by adjusting the relative speeds of the motors.

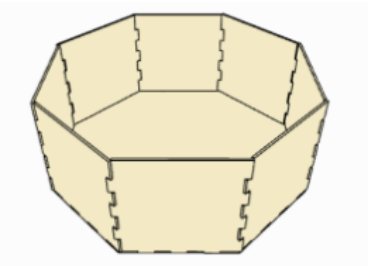


Figure 14: Housing

6 Appendix

6.1 Project Presentation

AUTOZEN

Greta Perez-Haiek
Aadhav Sivakumar
Nathan Smurthwaite

NYU, Tandon, Advanced Mechatronics, Project 2, Spring 2025

The Problem

A lack of artistic expression in zen gardens without the time, patience, and effort accumulated by the monks that have perfected the artform.

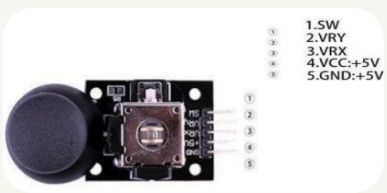
Some Kinetic Sand Tables that automate this beautiful artform are usually, $2,500 dollars, and could go upwards to $15,000 (Sisyphus Industries' TableTops).

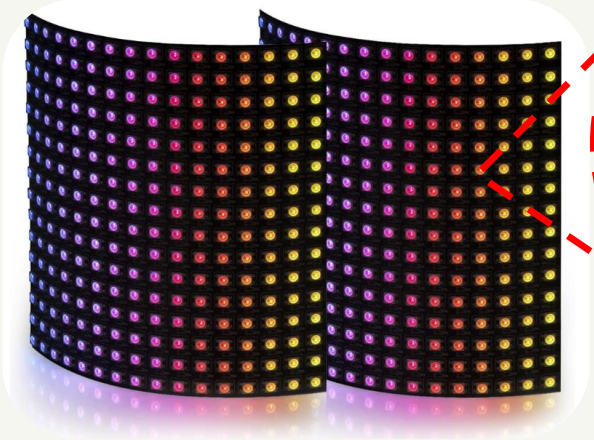
The Solution

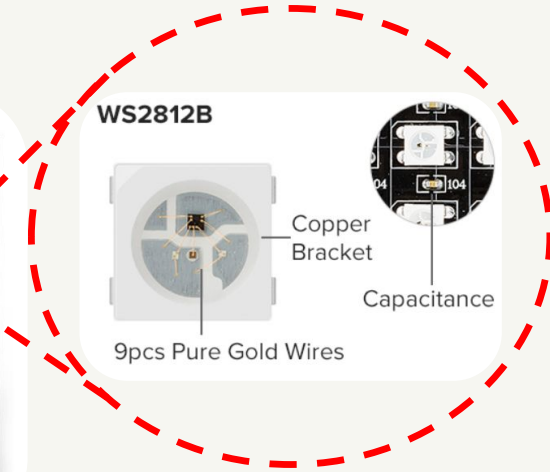
A 2-Arm Revolute Manipulator, with an interconnected user interface, that can allow for easy, minimally invasive drawings on a Zen Garden… all without breaking the bank.

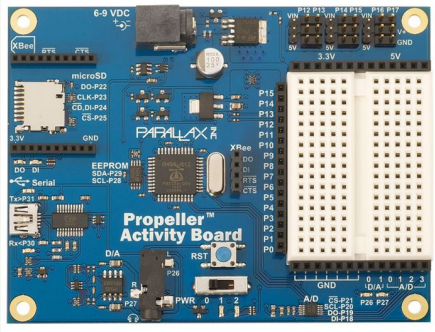
Our prototype, the Kinetic Sand Table “Autozen,” was built with a budget of less than $200

The Components

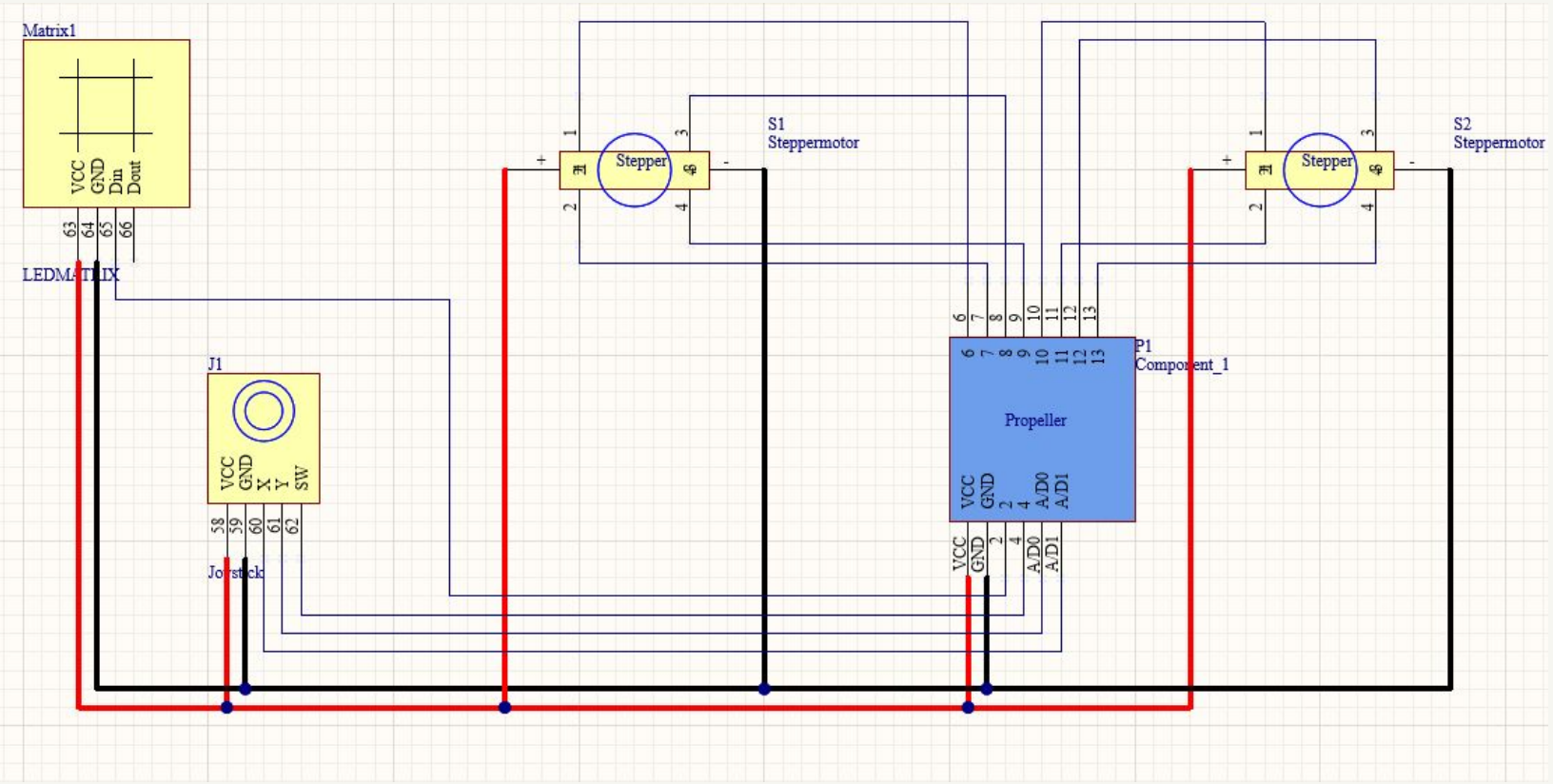




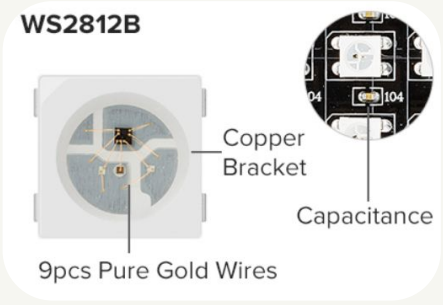




Circuit Diagram

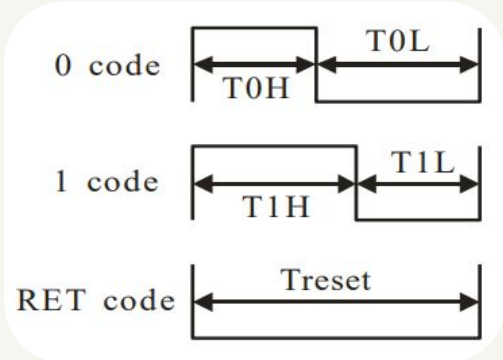


WS2812B LEDs



- A single WS2812B takes 3 bytes of data to control
- Precise timing at the nanosecond level is required to accurately control its brightness and color
- SimpleIDE contains a library that can easily control the brightness and color of these LEDs

T0H	0 code, high voltage time	0.4 us	±150 ns
T1H	1 code, high voltage time	0.8 us	±150 ns
T0L	0 code, low voltage time	0.85 us	±150 ns
T1L	1 code, low voltage time	0.45 us	±150 ns
RES	low voltage time	> 50 us	

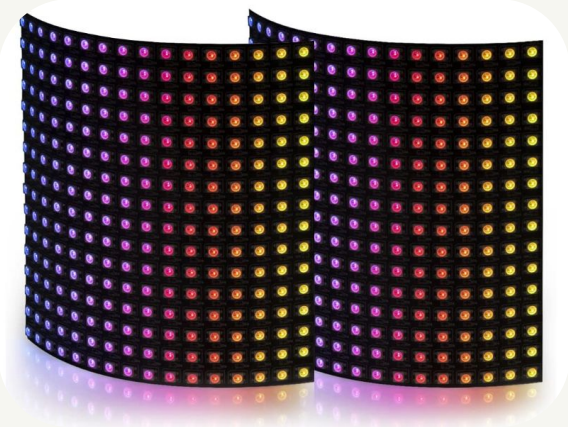


- `ws2812_t *ws2812_open(void);`
- `int ws_start(ws2812_t *driver, int usreset, int ns0h, int ns0l, int ns1h, int ns1l, int type);`
- `void ws2812_set(ws2812_t *driver, int pin, uint32_t *colors, int count);`
- `uint32_t ws2812_wheel(int pos);`
- `uint32_t ws2812_wheel_dim(int pos, int brightness);`
- `void ws2812_stop(ws2812_t *driver);`

Composition of 24bit data:

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

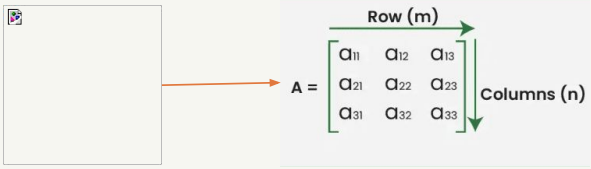
16x16 LED Matrix

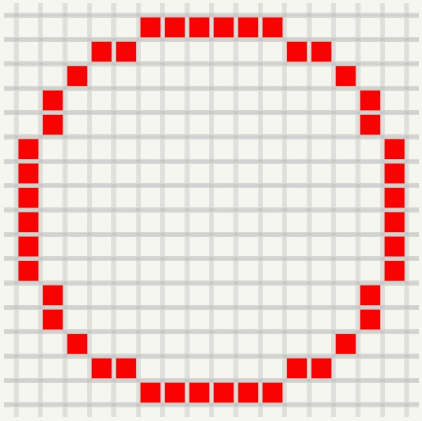


- 256 LEDs
- Max Draw of 75 Watts,

  lower brightness level to stay under current limit of propeller

- Color Order : **GRB**(**NOT RGB**)
- When you light up **1 color**(red or green or blue, **0.1W/LED** .





SN74AHCT125N Quadruple Bus Buffer Gates

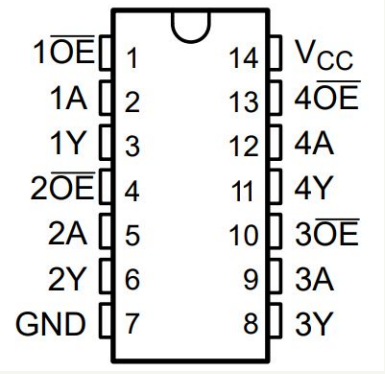
- Propeller GPIO Pins emit 3.3 V
- LED Matrix Data Line requires 5 V and precise control
- 5 V Output of Propeller is constant
- Need for precise timing of the WS2182B LEDs

5.5 Switching Characteristics

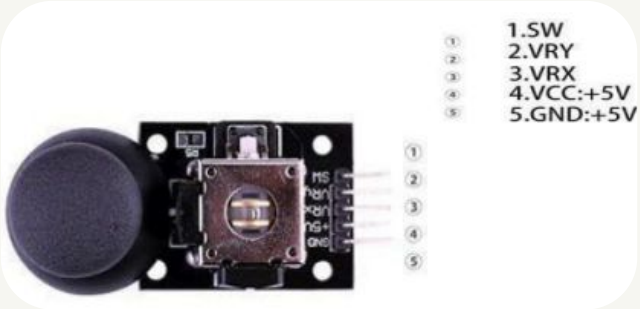
over recommended operating free-air temperature range, V_{CC} = 5 V ± 0.5 V (unless otherwise noted) (see Figure 6-1)

PARAMETER	FROM (INPUT)	TO (OUTPUT)	LOAD CAPACITANCE	T_A = 25°C			SN54AHCT125		SN74AHCT125		UNIT
				MIN	TYP	MAX	MIN	MAX	MIN	MAX	
t_{PLH}	A	Y	C_L = 15 pF		3.8(1)	5.5(1)	1(1)	6.5(1)	1	6.5	ns
t_{PHL}					3.8(1)	5.5(1)	1(1)	6.5(1)	1	6.5	
t_{PZH}	$\overline{OE}$	Y	C_L = 15 pF		3.6(1)	5.1(1)	1(1)	6(1)	1	6	ns
t_{PZL}					3.6(1)	5.1(1)	1(1)	6(1)	1	6	
t_{PHZ}	$\overline{OE}$	Y	C_L = 15 pF		4.6(1)	6.8(1)	1(1)	8(1)	1	8	ns
t_{PLZ}					4.6(1)	6.8(1)	1(1)	8(1)	1	8	
t_{PLH}	A	Y	C_L = 50 pF		5.3	7.5	1	8.5	1	8.5	ns
t_{PHL}					5.3	7.5	1	8.5	1	8.5	
t_{PZH}	$\overline{OE}$	Y	C_L = 50 pF		5.1	7.1	1	8	1	8	ns
t_{PZL}					5.1	7.1	1	8	1	8	
t_{PHZ}	$\overline{OE}$	Y	C_L = 50 pF		6.1	8.8	1	10	1	10	ns
t_{PLZ}					6.1	8.8	1	10	1	10	
$t_{sk(o)}$			C_L = 50 pF			1(2)				1	ns

(1) On products compliant to MIL-PRF-38535, this parameter is not production tested.
(2) On products compliant to MIL-PRF-38535, this parameter does not apply.



Joystick



- Module requires Vcc (5V) & GND for Inputs.
- There are two analog outputs for the signal,
  - 1 for the movement in the x-axis
  - 1 for the movement in the y-axis
  - *y-axis pin not necessary given 1D movement
- 1 Digital Output, which is a pull-down switch,
  - Leaves floating voltage when joystick isn't pressed and pulling it to ground when it is

In the circuit there is a **LARGE** pull-up resistor connected to the pin such that the voltage isn't floating when switch is not pressed.

5V Stepper Motor 28BYJ-48 + ULN2003 Driver Test Module

- Allows control of the stepper motor at 5 volts even through the parallax propeller which only provides 3.3 volts through it's output pins
- Using half step control, which follows this sequence

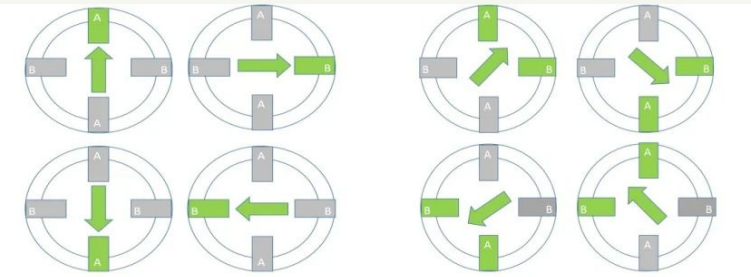
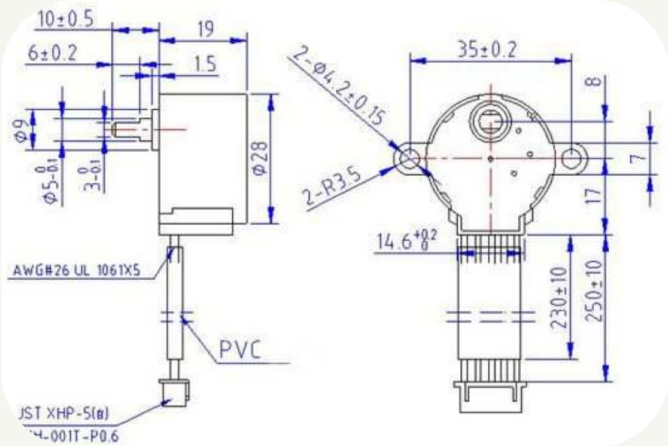


Fig 1 – One phase on – full step

Fig2 – Two phase on – full step

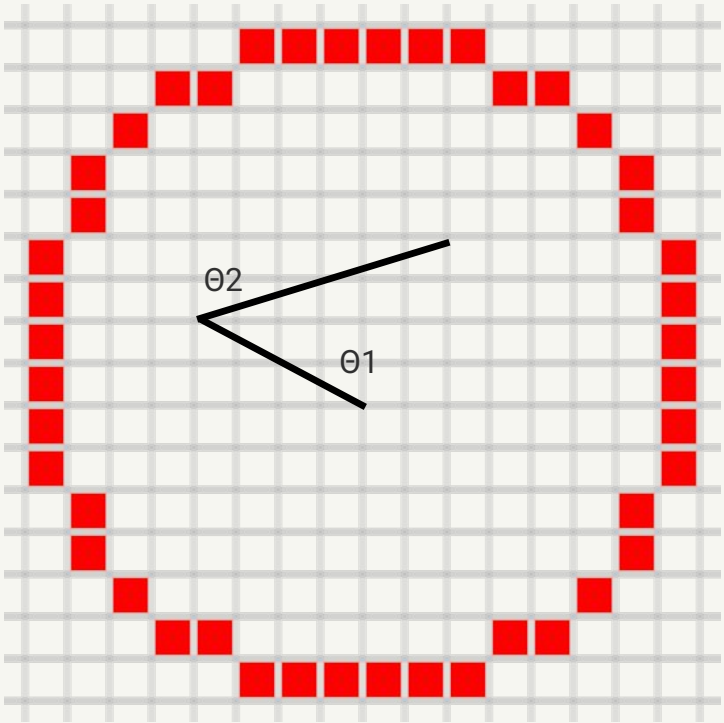
- 8 half steps is one cycle, and 8 cycles is 5.625 degrees, which is 1/64th of 360
- This gives a resolution of 4096 angles, allowing for precise control



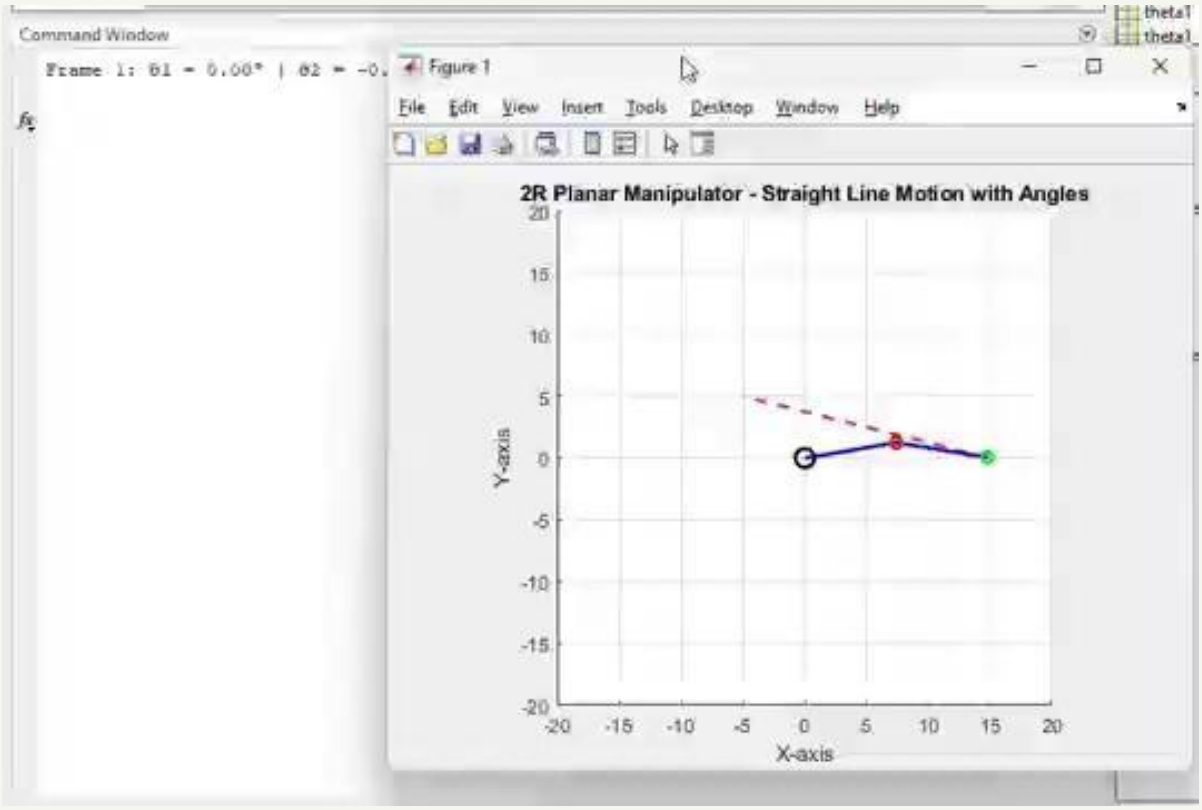
Motor Control of 2R manipulator

- The motor is controlled in 3 different ways
  - Manual control
    - This allows each individual motor to be controller by a single joystick
    - Moving up/down moves the first motor clockwise/counterclockwise
    - Moving left/right moves the second motor clockwise/counterclockwise
    - Pressing the button set the current point to the starting position
  - Straight line control
    - Allows you to select any point in the workspace
    - Once selected, calculates the inverse kinematics of 100 points along the path to move in a straight line
  - Pattern control
    - Allows you to select the ratio of speed, creating circular patterns

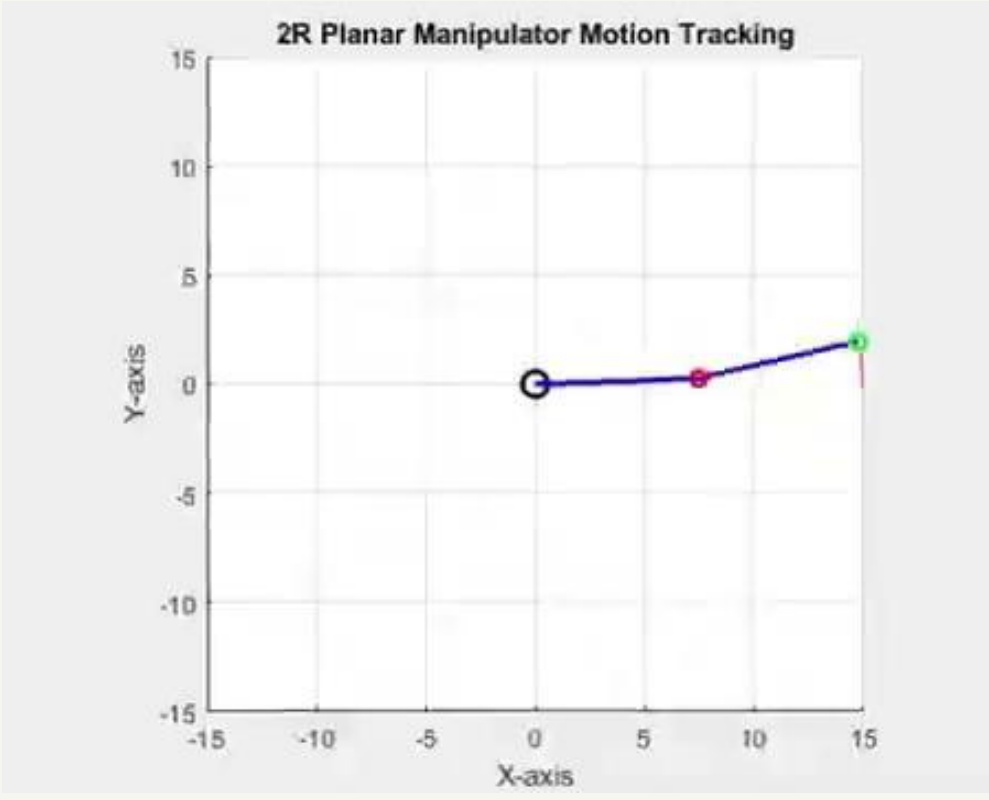
Manual control



Straight Line Control

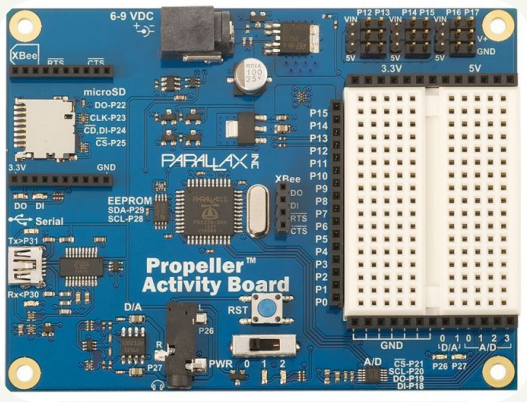


Pattern Control

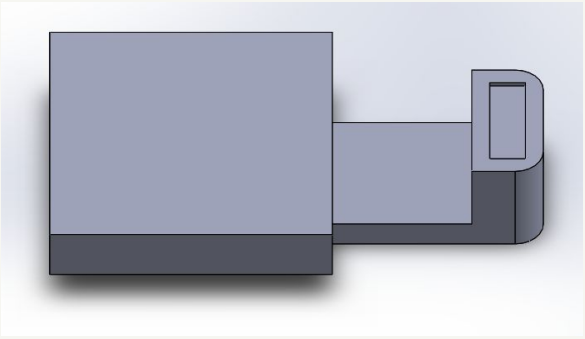


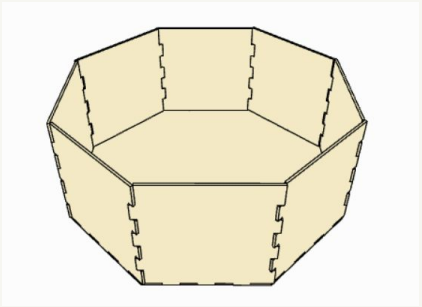
Parallax Propeller Activity Board

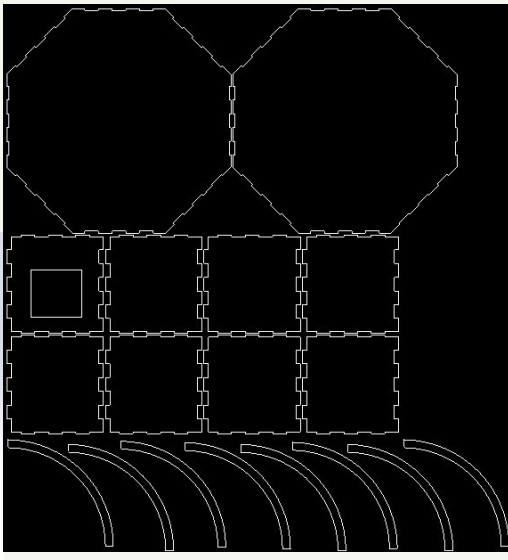
- **3.3v** output used to power the LED matrix
- 5V output used to power motors and joystick
- Pin 2 controls the LED matrix
- Pin 4 is used to take in the switch from the joystick
- Pins 6-13 are used to control the 2 stepper motors
- A/D pins 0 and 1 are used for the x and y inputs from the joystick

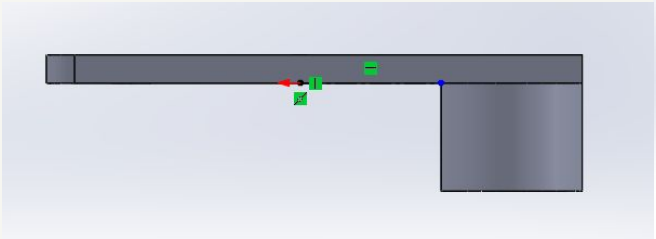


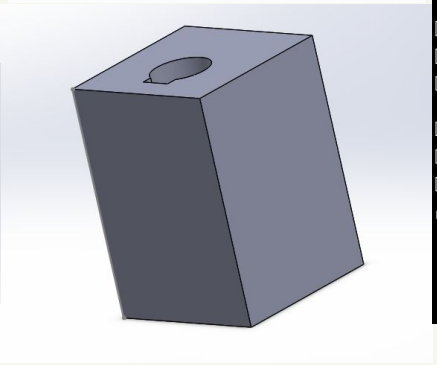
Housing + Construction



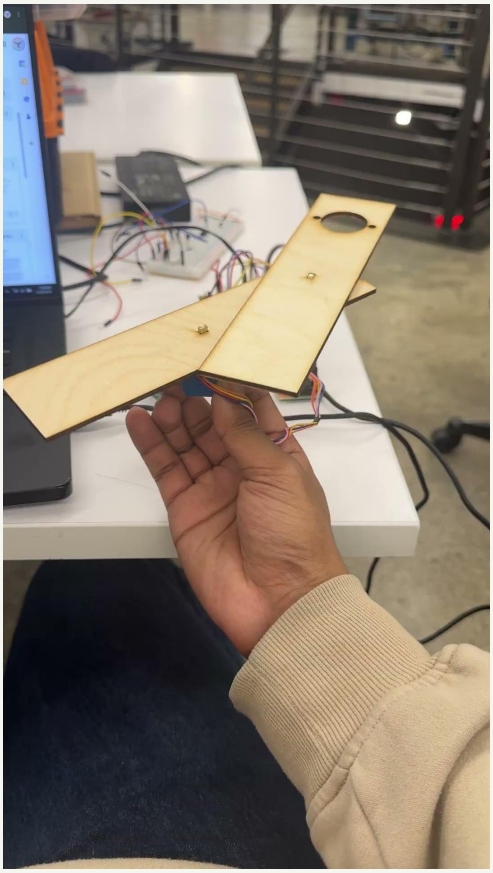




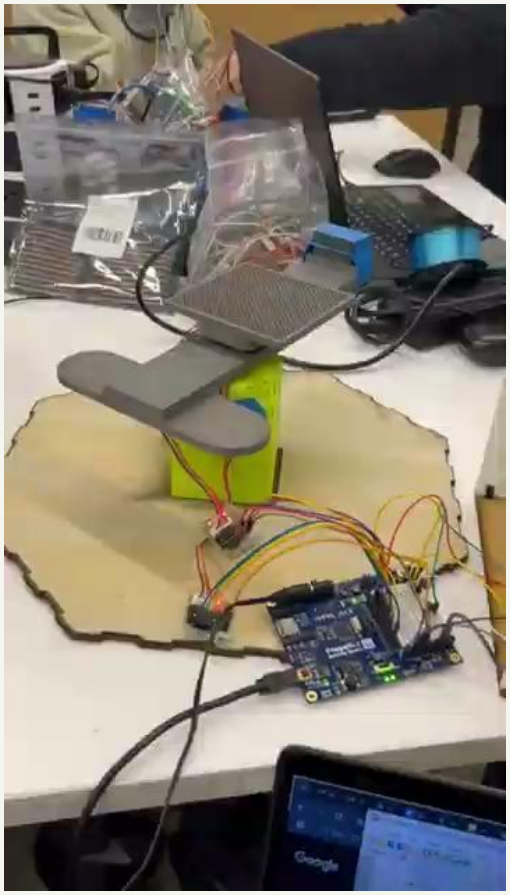




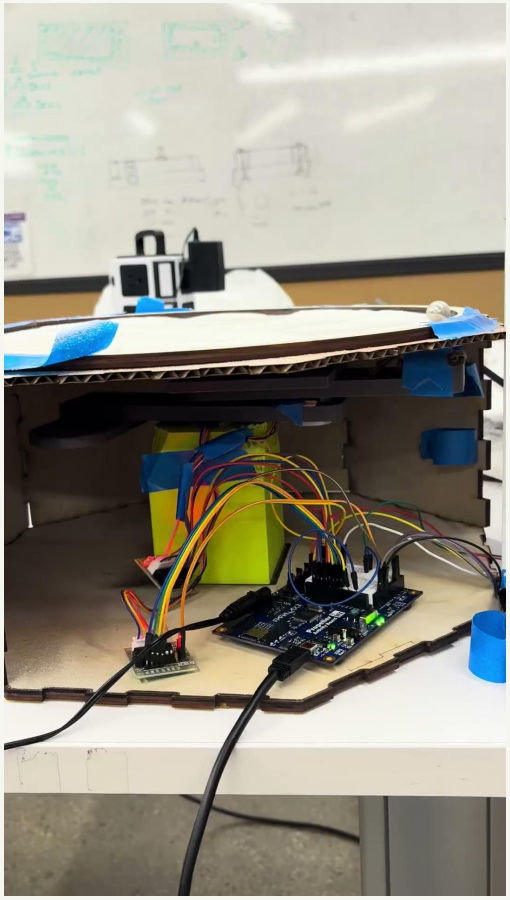
Progress



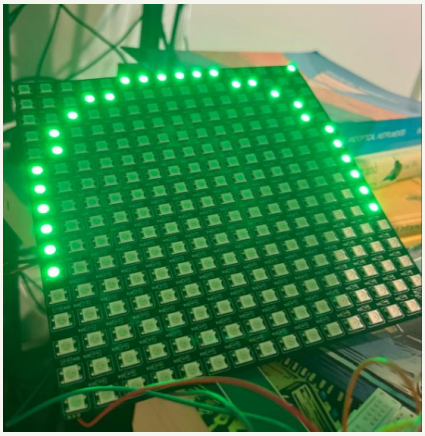
Progress

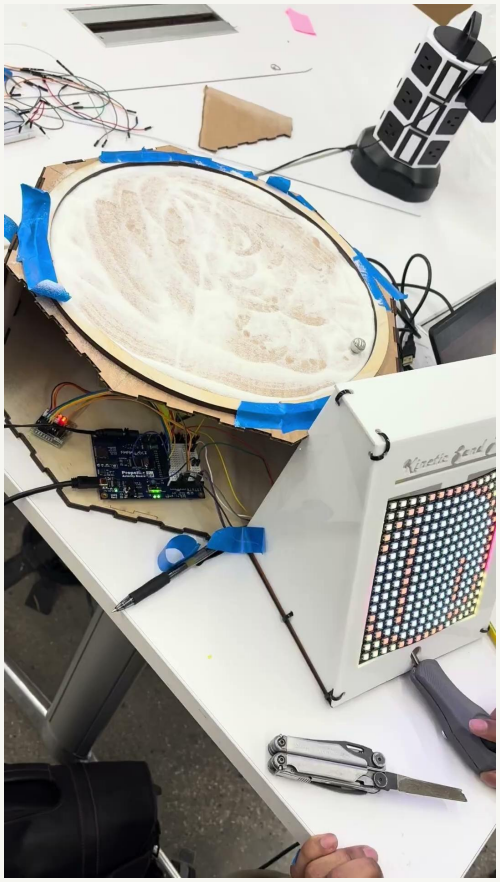


Progress



Video Demo (Pattern)





Video Demo (Straight line)

GANTT Chart

SPRING 2025	Kinetic Sand Table! GANTT CHART (Collaboration Plan)			
	Task	Team-Member	Week 1	Week 2
	Planned and ordered parts	Greta, Aadhav, Nate		
	Brainstormed Project Themes	Greta, Aadhav, Nate		
	Created a preliminary Blueprint	Nate		
	Created proof of concept: 2R Manipulator	Nate, Aadhav		
	Designing Housing	Greta		
	Manufacturing Housing	Greta		
	Programming	Aadhav, Nate		
	Assembly	Greta		
	Testing (Verification and Validation)	Greta, Aadhav, Nate		

Future Work/ Problems faced

Problems faced

- Ensuring constant contact of the magnet with the ceiling
- Ridges and imperfections in the surface that the ball is rolling on
- The wires of the second motor winding around the center, only allowing one rotation

Future work

- Hall Effect Sensors to better track exact motor positions
- Vibrational actuator to reset the sand floor
- Higher resolution LED Matrix
- More shapes to draw
- Motors with built in encoders to minimize drift over time
- Solution for wire winding problem (RP manipulator instead of 2R)

Thank You, and stay Zen

6.2 Code

```
#include "simpletools.h"
#include "adcDCpropab.h"
#include "ws2812.h"
#include <stdio.h>

#define NUM_LEDS 256
#define OD_PIN 2 // OUTPUT_DATA = OD
#define ED_PIN 3 // ENABLE_DATA = ED
#define bright 1

#define PI 3.141592654

#define L1 7.5
#define L2 7.5

double arg = 60;
double result;

double theta1;
double theta2;

double targetx=15;
double targety=0;

double currentx=15;
double currenty=0;

double currenttheta1=0;
double currenttheta2=0;

uint32_t board[16][16];
unsigned char LEDmapmatrix[16][16];


enum States {
  ManualThetas,
  Manualxy,
  StraightLines,
  CircularPatterns,
  Allpatterns,
  StraightSelection,
  PatternSelection
};

int updown;
```

```
int leftright;
int joyswitch;
float selectorcenterx=7.5;
float selectorcentery=7.5;
float selectorcheckx=7.5;
float selectorchecky=7.5;
int patternM1speed=5;
int patternM2speed=5;

int Current_State = ManualThetas;

int recoup_pause=0;

//L1 and L2 is 75 millimeters
//x = L1 * cos(θ1) + L2 * cos(θ1 + θ2)
//y = L1 * sin(θ1) + L2 * sin(θ1 + θ2)

//1st motor
#define M1p1 6
#define M1p2 7
#define M1p3 8
#define M1p4 9

//2nd motor
#define M2p1 10
#define M2p2 11
#define M2p3 12
#define M2p4 13

int motor1stage=1;
int motor2stage=1;
int MotorPin1=0;
int MotorPin2=0;
int MotorPin3=0;
int MotorPin4=0;
int motor1dir=1;
int motor2dir=1;

//dir is direction, 1 for counterclockwise, 2 for clockwise
//motornum selects the motor. motor 1 is connected on pins 1-4 and
motor 2 is connected on pins 5-8
//paustime changes the speed, higher is slower lower is faster

void inverse_kinematics(double x, double y, double *theta1, double
*theta2) {
```

```
    // Compute cos(theta2)
    double cosTheta2 = (x*x + y*y - L1*L1 - L2*L2) / (2 * L1 * L2);

    // Check if the target is reachable
    if (cosTheta2 < -1 || cosTheta2 > 1) {
        printf("Target (%.2f, %.2f) is out of reach.\n", x, y);
        return;
    }

    // Compute theta2 (two possible solutions: +sqrt or -sqrt)
    double sinTheta2 = sqrt(1 - cosTheta2 * cosTheta2);
    double theta2_a = atan2(sinTheta2, cosTheta2);  // Elbow down
    double theta2_b = atan2(-sinTheta2, cosTheta2); // Elbow up

    // Compute theta1 for both cases
    double k1 = L1 + L2 * cosTheta2;
    double k2_a = L2 * sinTheta2;
    double k2_b = -L2 * sinTheta2;

    double theta1_a = atan2(y, x) - atan2(k2_a, k1);
    double theta1_b = atan2(y, x) - atan2(k2_b, k1);

    // Convert to degrees
    *theta1 = theta1_a * (180.0 / PI);
    *theta2 = theta2_a * (180.0 / PI);

    //print("Solution 1: θ1 = %.2f°, θ2 = %.2f°\n", theta1_a * (180.0
/ PI), theta2_a * (180.0 / PI));
    //print("Solution 2: θ1 = %.2f°, θ2 = %.2f°\n", theta1_b * (180.0
/ PI), theta2_b * (180.0 / PI));
}

void motormove(int motornum, int dir, float pausetime){

  int stage;
  if(motornum==1){
    MotorPin1=M1p1;
    MotorPin2=M1p2;
    MotorPin3=M1p3;
    MotorPin4=M1p4;
    stage=motor1stage;
  } else {
    MotorPin1=M2p1;
    MotorPin2=M2p2;
    MotorPin3=M2p3;
```

```
  MotorPin4=M2p4;
  stage=motor2stage;
}
//print("%d",MotorPin1);
switch(stage){
  case 1:
    low(MotorPin1);
    low(MotorPin2);
    low(MotorPin3);
    high(MotorPin4);

    pause(pausetime);
    break;
  case 2:

    low(MotorPin1);
    low(MotorPin2);
    high(MotorPin3);
    high(MotorPin4);

    pause(pausetime);
  break;

  case 3:

    low(MotorPin1);
    low(MotorPin2);
    high(MotorPin3);
    low(MotorPin4);

    pause(pausetime);
  break;

  case 4:

    low(MotorPin1);
    high(MotorPin2);
    high(MotorPin3);
    low(MotorPin4);

    pause(pausetime);
  break;

  case 5:
```

```
      low(MotorPin1);
      high(MotorPin2);
      low(MotorPin3);
      low(MotorPin4);

      pause(pausetime);
    break;

    case 6:

      high(MotorPin1);
      high(MotorPin2);
      low(MotorPin3);
      low(MotorPin4);

      pause(pausetime);
    break;

    case 7:

      high(MotorPin1);
      low(MotorPin2);
      low(MotorPin3);
      low(MotorPin4);

      pause(pausetime);
    break;

    case 8:

      high(MotorPin1);
      low(MotorPin2);
      low(MotorPin3);
      high(MotorPin4);

      pause(pausetime);
    break;
  }
if(motornum==1){
 if(dir==0){

     motor1stage=1;

}
 else if(dir==1){
```

```
    motor1stage++;
    if(motor1stage>=9){
      motor1stage=1;
    }
 } else if(dir==2){
   motor1stage--;
    if(motor1stage<=0){
      motor1stage=8;
    }
 }
 }
 else if(motornum==2){
   if(dir==0){

      motor2stage=1;

 }
  else if(dir==1){
  //print("HEHEHE");
    motor2stage++;
    if(motor2stage>=9){
      motor2stage=1;
    }
 } else if(dir==2){
   motor2stage--;
    if(motor2stage<=0){
      motor2stage=8;
    }
 }
 }


}



//64 cycles is 5.625 degrees
void motormovedegrees(int mnum, int direction, int ptime, int
numcycles){
  for(int i=0;i<numcycles;i++){
    motormove(mnum, direction, ptime);
  }
}
```

```
void motormoveduration(int mnum, int direction, int ptime, int
movetime){
  for(int i=0;i<movetime/ptime;i++){
    motormove(mnum, direction, ptime);
  }
}
void blendmove(int direction1, int direction2, int ptime1,  int
ptime2, int movetime){
  for(int i=0;i<movetime;i++){
    if(i%ptime1==0){
      motormove(1, direction1, 1);
    }
    if(i%ptime2==0){
      motormove(2, direction2, 1);
    }

  }
}

void movemotortoposition(int xp, int yp){
    inverse_kinematics(xp, yp, &theta1, &theta2);
    if((theta1-currenttheta1)<0){
      motor1dir=2;
    }else{
      motor1dir=1;
    }
    if((theta2-currenttheta2)<0){
      motor2dir=2;
    }else{
      motor2dir=1;
    }
    motormovedegrees(1, motor1dir, 2,
abs((int)((theta1-currenttheta1)*11.377777)));
    pause(1);
    motormovedegrees(2, motor2dir, 2,
abs((int)((theta2-currenttheta2)*11.377777)));
    pause(1);
    currenttheta1=theta1;
    currenttheta2=theta2;
    currentx=xp;
    currenty=yp;
}

void movemotorstraightline(double xtar, double ytar){
  double numstops=100;
```

```
  int numstopsint=numstops;
  double theta1vals[numstopsint];
  double theta2vals[numstopsint];
  int currstop=0;
  double initialx=currentx;
  double initialy=currenty;
  double nextx;
  double nexty;
  while(currstop<numstops){

    nextx=initialx+((xtar-initialx)*((currstop+1)/numstops));
    //print("%f\n",((currstop+1)/numstops));

    nexty=initialy+((ytar-initialy)*((currstop+1)/numstops));
    inverse_kinematics(nextx, nexty, &theta1, &theta2);
    theta1vals[currstop]=theta1;
    theta2vals[currstop]=theta2;

    currstop++;

  }
  print("donecalc\n");
  currstop=0;
  while(currstop<numstops){

    if((theta1vals[currstop]-currenttheta1)<0){
      motor1dir=2;
    }else{
      motor1dir=1;
    }
    if((theta2vals[currstop]-currenttheta2)<0){
      motor2dir=2;
    }else{
      motor2dir=1;
    }
    motormovedegrees(1, motor1dir, 2,
abs((int)((theta1vals[currstop]-currenttheta1)*11.377777)));
    pause(2);
    motormovedegrees(2, motor2dir, 2,
abs((int)((theta2vals[currstop]-currenttheta2)*11.377777)));
    pause(2);
    currenttheta1=theta1vals[currstop];
    currenttheta2=theta2vals[currstop];

    currstop++;
```

```
  }
  currentx=xtar;
  currenty=ytar;
}



int main()          // Main function
{
  pause(1000);
  adc_init(21, 20, 19, 18);  // Initialize ADC
  targetx=0;
  targety=10;

  for (int j = 0; j <= 15; j++){
    if(j%2==0){
      for (int i = 0; i <= 15; i++){
      LEDmapmatrix[i][j]=i+j*16;
    }
    } else if (j%2==1) {
      for (int i = 15; i >= 0; i--){
      LEDmapmatrix[i][j]=(15-i)+j*16;
    }
    }

  }




    // Direction of Power Pins
    DIRA |= (1 << OD_PIN);
    DIRA |= (1 << ED_PIN);
    // Enables buffer for data pin of the matrix
    OUTA &= ~(1 << ED_PIN);

    // Create an array of colors
    uint32_t leds[NUM_LEDS];

    for (int i = 0; i < NUM_LEDS; i++) {
        leds[i] = ws2812_wheel_dim(1, 10);
    }
```

```
    for (int i = 0; i < 16; i++) {
      for (int j = 0; j < 16; j++) {
        if(((i-7.5)*(i-7.5)+(j-7.5)*(j-7.5))>63){
          board[i][j]=ws2812_wheel_dim(30,20);
        } else if (((i-7.5)*(i-7.5)+(j-7.5)*(j-7.5))>53) {
          board[i][j]=ws2812_wheel_dim(1,20);
        } else {
          board[i][j]=0x080808;
        }
      }
    }
    for (int i = 0; i < 16; i++) {
      for (int j = 0; j < 16; j++) {
        leds[LEDmapmatrix[i][j]]=board[i][j];
      }
    }
    ws2812_t *led_driver = ws2812b_open(); // Assuming WS2812B LEDs
    ws2812b_start(led_driver);
    pause(100);
    ws2812_set(led_driver, OD_PIN, leds, NUM_LEDS);



    while(1){
    switch(Current_State){
      case StraightSelection:
        updown = adc_in(0);      // Read raw 12-bit value
        leftright = adc_in(1); // Convert to voltage

        if(updown<1000){
          selectorchecky=selectorcentery-1;
        } else if (updown>3000){
          selectorchecky=selectorcentery+1;
        }
        if(leftright<1000){
          selectorcheckx=selectorcenterx-1;
      } else if (leftright>3000) {
          selectorcheckx=selectorcenterx+1;
        }

if(((selectorcheckx-7.5)*(selectorcheckx-7.5)+(selectorchecky-7.5)*(s
electorchecky-7.5))<53.5){
          selectorcentery=selectorchecky;
          selectorcenterx=selectorcheckx;
        }  else {
```

```
          selectorchecky=selectorcentery;
          selectorcheckx=selectorcenterx;
        }
        for (int i = 0; i < 16; i++) {
          for (int j = 0; j < 16; j++) {
            if(((i-7.5)*(i-7.5)+(j-7.5)*(j-7.5))>63){
              board[i][j]=ws2812_wheel_dim(30,20);
            } else if (((i-7.5)*(i-7.5)+(j-7.5)*(j-7.5))>53) {
              board[i][j]=ws2812_wheel_dim(1,20);
            } else {
              board[i][j]=0x080808;
            }
          }
        }

board[(int)((currentx+15)/2)+1][(int)((currenty+15)/2)]=ws2812_wheel_
dim(200,100);

board[(int)((currentx+15)/2)+1][(int)((currenty+15)/2)+1]=ws2812_whee
l_dim(200,100);

board[(int)((currentx+15)/2)][(int)((currenty+15)/2)]=ws2812_wheel_di
m(200,100);

board[(int)((currentx+15)/2)][(int)((currenty+15)/2)+1]=ws2812_wheel_
dim(200,100);


board[(int)(selectorcenterx-0.5)][(int)(selectorcentery-0.5)]=ws2812_
wheel_dim(100,100);

board[(int)(selectorcenterx-0.5)][(int)(selectorcentery+0.5)]=ws2812_
wheel_dim(100,100);

board[(int)(selectorcenterx+0.5)][(int)(selectorcentery-0.5)]=ws2812_
wheel_dim(100,100);

board[(int)(selectorcenterx+0.5)][(int)(selectorcentery+0.5)]=ws2812_
wheel_dim(100,100);




        for (int i = 0; i < 16; i++) {
```

```
    for (int j = 0; j < 16; j++) {
      leds[LEDmapmatrix[i][j]]=board[i][j];
    }
  }


  ws2812_set(led_driver, OD_PIN, leds, NUM_LEDS);


  pause(10);
  if(recoup_pause){
    pause(1000);
    recoup_pause=0;
  }
  if(!input(4)){
    pause(1000);
    if(!input(4)){
      Current_State=PatternSelection;
      recoup_pause=1;
    } else {
    targetx=(selectorcenterx*2)-15;
    print("%f \n" , targetx);
    targety=(selectorcentery*2)-15;
    print("%f \n" , targety);
    Current_State=StraightLines;
  }
  }
  break;
case PatternSelection:
updown = adc_in(0);      // Read raw 12-bit value
leftright = adc_in(1); // Convert to voltage

if(updown<1000 && patternM1speed>1){
    patternM1speed--;
  } else if (updown>3000 && patternM1speed<10){
    patternM1speed++;
  }
  if(leftright<1000 && patternM2speed>1){
    patternM2speed--;
} else if (leftright>3000 && patternM2speed<10) {
    patternM2speed++;
  }
  for (int i = 0; i < 16; i++) {
    for (int j = 0; j < 16; j++) {
      if(((i-7.5)*(i-7.5)+(j-7.5)*(j-7.5))>63){
```

```
        board[i][j]=ws2812_wheel_dim(30,20);
      } else if (((i-7.5)*(i-7.5)+(j-7.5)*(j-7.5))>53) {
        board[i][j]=ws2812_wheel_dim(1,20);
      } else {
        board[i][j]=0x080808;
      }
    }
  }

  for(int g=1; g<=patternM1speed; g++){
    board[4][2+g]=ws2812_wheel_dim(1,20);
  }
  for(int h=1; h<=patternM2speed; h++){
    board[11][2+h]=ws2812_wheel_dim(1,20);
  }

  for (int i = 0; i < 16; i++) {
    for (int j = 0; j < 16; j++) {
      leds[LEDmapmatrix[i][j]]=board[i][j];
    }
  }


  ws2812_set(led_driver, OD_PIN, leds, NUM_LEDS);

  if(recoup_pause){
    pause(1000);
    recoup_pause=0;
  }
  if(!input(4)){
    pause(1000);
    if(!input(4)){
      Current_State=ManualThetas;
      recoup_pause=1;
    } else {
      Current_State=CircularPatterns;
    }

  }


break;
case ManualThetas:
  updown = adc_in(0);      // Read raw 12-bit value
  leftright = adc_in(1); // Convert to voltage
```

```
  //printf("Raw: %d, asdf %d \n", updown, leftright);
  for(int coasttime=0;coasttime<8;coasttime++){
    //print("hello\n");
  if(updown<500){
    motormove(1, 1, 2);
  } else if (updown>3500){
    motormove(1, 2, 2);
  }
  if(leftright<500){
    motormove(2, 1, 2);
} else if (leftright>3500) {
    motormove(2, 2, 2);
  }
}
```

```
if(recoup_pause){
  for (int i = 0; i < 16; i++) {
    for (int j = 0; j < 16; j++) {
      if(((i-7.5)*(i-7.5)+(j-7.5)*(j-7.5))>63){
        board[i][j]=ws2812_wheel_dim(30,20);
      } else if (((i-7.5)*(i-7.5)+(j-7.5)*(j-7.5))>53) {
        board[i][j]=ws2812_wheel_dim(1,20);
      } else {
        board[i][j]=0x080808;
      }
    }
  }

  for (int i = 0; i < 16; i++) {
    for (int j = 0; j < 16; j++) {
      leds[LEDmapmatrix[i][j]]=board[i][j];
    }
  }

  ws2812_set(led_driver, OD_PIN, leds, NUM_LEDS);



    pause(1000);
    recoup_pause=0;
```

```
        }

      if(!input(4)){
          pause(1000);
          if(!input(4)){
            Current_State=StraightSelection;
            recoup_pause=1;
          } else {
            currentx=15;
            currenty=0;


          }


        }
        break;
      case StraightLines:
      movemotorstraightline(targetx,targety);

      Current_State=StraightSelection;
        break;
      case CircularPatterns:
      blendmove(1 ,1 ,patternM1speed ,patternM2speed
,patternM1speed*4096);
      Current_State=PatternSelection;
      break;

      };

  }
  ws2812_stop(led_driver);
  ws2812_close(led_driver);
  return 0;
}
```