Garrick Jensen
Stevie Steinberg
E155, Professor Brake
11/23/21

MicroProcessor Systems and Design Final Project Check Off

Abstract:
Our project aims to create a robotic arm with three joints that can draw a predetermined set of shapes on a whiteboard that is a fixed distance away from the base of the arm. We aim to implement the system using a microprocessor to control the motor inputs and an FPGA to process user inputs like which shape the arm will draw.

Goals:
We aim for the arm to be able to draw a shape on a whiteboard, specified by the user from a given list of shapes. We plan on having the user be able to specify between at least two shapes: a circle and a star. We may have other shapes that it can draw, but the minimum shapes the robotic arm must be able to draw are the circle and star. The arm must be placed a certain distance from the white board. We are not expecting it to be able to draw on a surface from an unknown distance away. Additionally, the board we are drawing on must be a flat plane.

Motor Hardware Selection

We looked at multiple options for motors to control the arm. In selecting the type of motor, we prioritized objectives in this order: the angular position control precision, the torque of the motor, the size and weight of the motor, the complexity of the drive circuitry, and the cost.

Driving a DC motor (either brushed or brushless) would make the drive circuitry very simple. However, without closely understanding the specifications of the motor and performing real-time integration of the motor drive we would have trouble characterizing the position of each motor.

Between steppers and servos, it was a relatively straightforward decision to use steppers. Servos are best used in high-speed applications where closed-loop control is required and significant resources can go into tuning the control scheme. Steppers, meanwhile, thrive in low-speed applications (like this one), where they outperform servos in terms of maximum torque output. Additionally, the force of attraction between the stator and rotor provides a “holding torque” that prevents the motor from slipping, even when not actively being driven. This is important given the discontinuous nature of how we will be driving the motors.

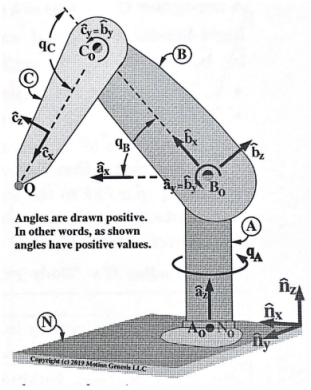
To complete the drive circuitry, we needed an H-bridge, a PWM generator, and (preferably) the ability to generate microsteps. The motors we selected each have a resolution of 1.8 degrees, which corresponds to an arc length of 0.47cm for an arm segment of 15cm. This is relatively low resolution, so finding an integrated driver circuit that could provide increased resolution was important. We selected the A4998 for quick shipping, sufficient current drive, and the ability to

easily microstep. With the 1/16 microstep setting, we can command a precision of 0.3mm, which is excellent. The resulting MCU control scheme is also very straightforward, as we will describe later.

Stepper motors derive their driving force from the current flowing through them, relatively independent of the voltage they are fed. There is a small drop across the H-bridge FETs, and a nominal drop across the windings of the stepper phase, but as a stepper motor increases in angular velocity a back emf is generated that serves as a voltage drop for the stepper. As such, it is important to have a sufficiently high voltage to overcome the first two effects with voltage to spare for the back emf. The returns are diminishing, but the manufacturers suggested a drive voltage in the realm of 30Vdc. As such, we purchased a 30V, 15A supply that should be ample to drive the system. Initially, we had concerns about the energy being shunted from a driven stepper motor onto the 30V bus, but the relatively low current and “mixed decay mode” (a scheme where, when the motor has just stopped being driven, each H-bridge FET opens briefly before the opposite FETs close) sufficiently protect the power supply from return current. The motors are sized according to simulations to ensure ample torque. We used a safety factor of 2 beyond the holding torque: so for a motor of holding torque 2Nm, we expected being able to move no more than 1Nm. This is due to the increased strain on the motor, resulting in a weaker drive and less acceleration when a step is commanded.

System Dynamics Characterization

Following the below general diagram, if we model the location of the motors and marker as points which depend on the angle between each arm and set lengths of each arm we get the below set of equations. Here the length of each arm is given by LA, LB and LC and the angles between each arm is given by qa, qb, and qc. We get the position of the marker in 3D space as x, y, and z where the distance of the marker from the board is the x distance We can then use these equations to solve for each of the angles over time and also solve for the first and second derivatives of each of these angles in terms of the arm lengths and values that are specified for the x, y, and z distances.

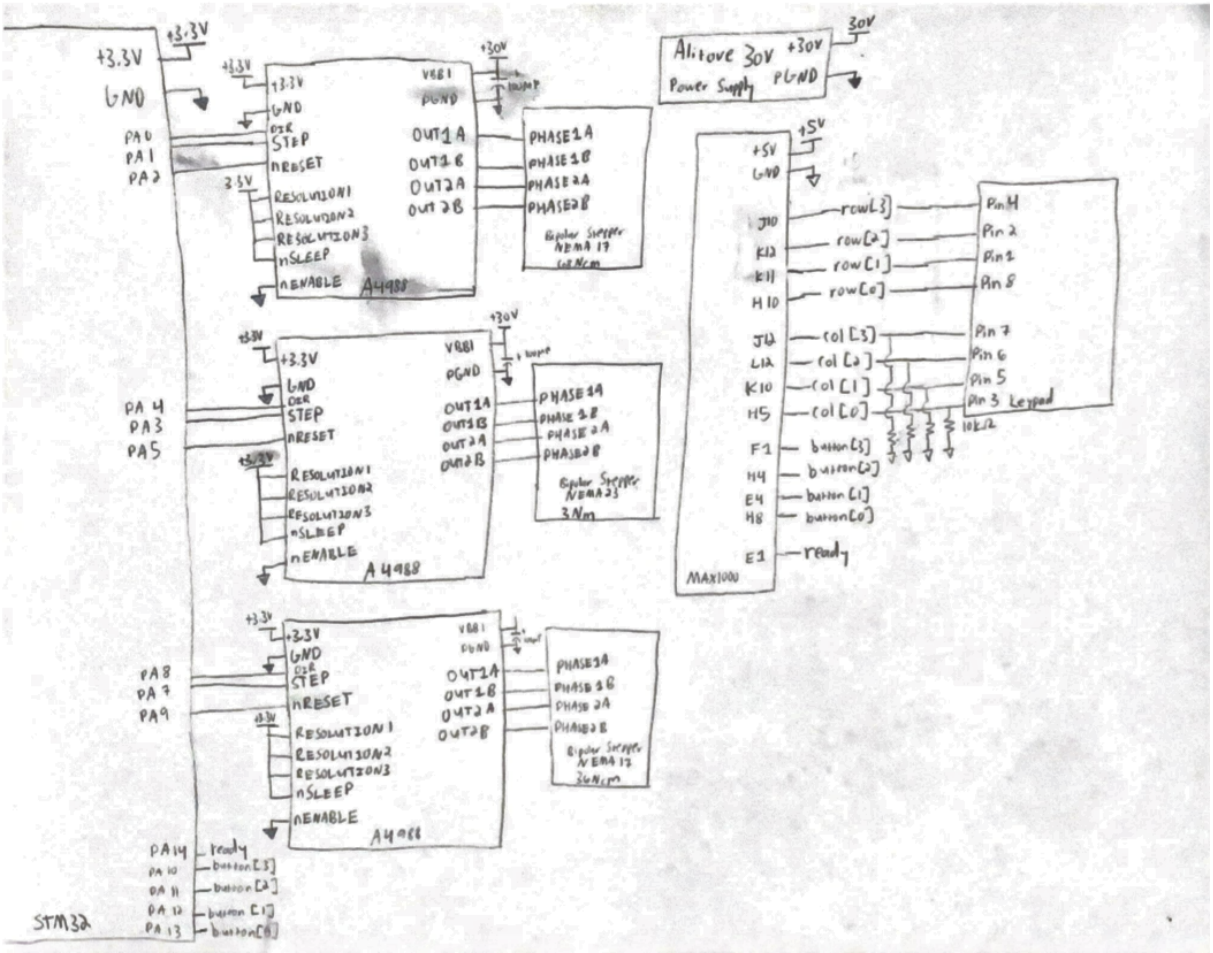


(Image credit to Paul Mitiguy)

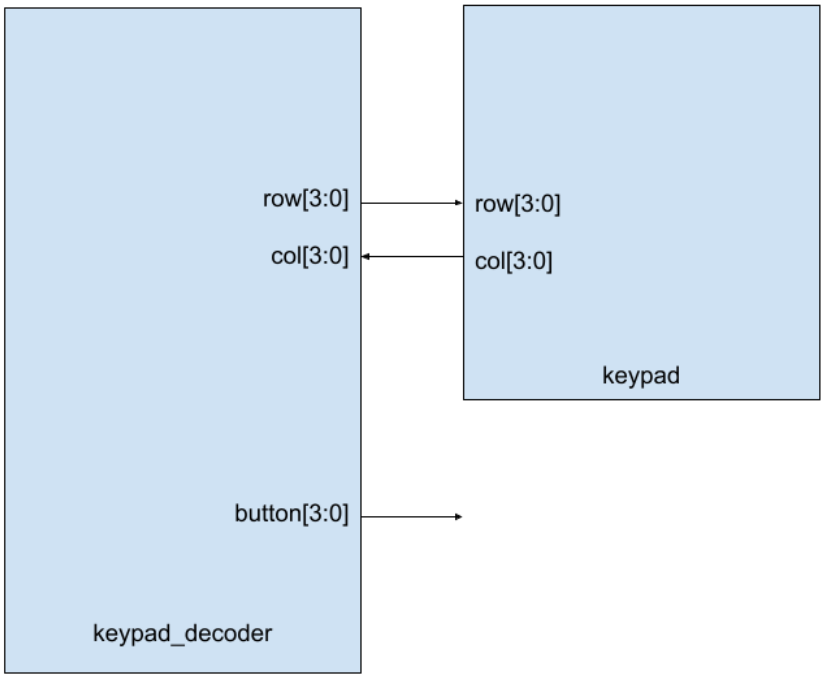
$$x = LB \cdot \cos(qa) \cdot \cos(qb) + LC \cdot \cos(qa) \cdot \cos(qb - qc)$$
$$y = LB \cdot \sin(qa) \cdot \cos(qb) + LC \cdot \sin(qa) \cdot \cos(qb - qc)$$
$$z = LA + LB \cdot \sin(qb) + LC \cdot \sin(qb - qc)$$

However, the primary goal of the robot is to be able to draw a shape. In this case, then our z and y values are no longer constant values but change over time. First, we parameterize y and z as functions of time. Then, we can then substitute these values into the scalar equations derived above and solve for the first and second angle derivatives. This means that we have our angle equations in terms of time and initial values of the system. When we increment time and graph the position vector from the base A0 to our marker Q, we see the specified curve traced out over time.

Breadboard Schematics



FPGA Logic Block Diagrams



MCU Routines

The Microprocessor code operates within an infinite while loop. It starts by waiting for user input from the FPGA which is passed data from the keypad. Once the button press is determined and the signal is passed to the microcontroller, the code exits its waiting state and then uses a case statement to choose which array of angles it will use to drive the motors. The array of angles that it will choose from for each shape is a Nx3 size array where N is the number of angles it takes to complete the shape. The 3 columns are for each of the motors. The array is terminated with a row of all 0's. A while loop iterates through the array until it reaches the last row of 0's. Before this occurs, the microprocessor calculates the max number of steps a single motor must take to get from the previous angle to the current angle. This max number of steps determines the iterations of a for loop that sends a pulse or single square wave to the motor drivers telling it to take a step. The other two motors take the modulus of the motor with the max number of necessary steps and so while on every iteration of the for loop the max motor takes a step, the other two will step on every other or 3rd or 4th iteration and so on depending on how many steps they need to get to their angle.

Once all three motors get to the current angle (which should happen at the same time) the while loop gets the next set of angles in the array. If there is ever a negative angle change, such that the motor needs to change the direction its spinning, then the direction pin is toggled for that

angle of the specific motor. This code was designed so that all three of the motors get to their new angles at the same time and rotate at the same time, just at different speeds. This is crucial to getting the correct shape we want to create. The code described above can be found below along with code that was used to test the functionality of the motors incrementally..

Microprocessor C code:

```
#include <stdio.h>
#include <math.h>
#include "STM32F401RE.h"


//define the pins for the three drivers
//driverA
#define STEPPINA GPIO_PA1
#define DIRPINA GPIO_PA0
//driverB
#define STEPPINB GPIO_PA3
#define DIRPINB GPIO_PA4
//driverC
#define STEPPINC GPIO_PA7
#define DIRPINC GPIO_PA8
//the reset pins for the motor drivers
#define RESETA GPIO_PA2
#define RESETB GPIO_PA5
#define RESETC GPIO_PA9

//pins for the FPGA
#define FPGAPIN GPIO_PA14

//define the angle vectors
//{motor A, motor B, motor C}
const float circleAngles[][3] = {
{3.14, 3.14, 3.14},
{0.0, 0.0, 0.0}};

const float starAngles[][3] = {
{3.14, 3.14, 3.14},
{0.0, 0.0, 0.0}};

const float squareAngles[][3] = {
{3.14, 3.14, 3.14},
{0.0, 0.0, 0.0}};
```

```
#define STEPANGLE 0.00196349 //radians, in degrees = 0.1125


int main(void){ //testing for motor control for each of the different motors different functionality
    RCC->APB1ENR.TIM2EN = 1;
    // Configure flash latency and set clock to run at 84 MHz
    configureFlash();
    configureClock();
    //configure the timer for the pulses
    initTIM(TIM2);
    // Enable GPIOA
    RCC->AHB1ENR.GPIOAEN = 1;

    //testing motors A, B, and C separately
    // direction and step pins
    pinMode(GPIOA, STEPPINA, GPIO_OUTPUT);
    pinMode(GPIOA, DIRPINA, GPIO_OUTPUT);
    pinMode(GPIOA, RESETA, GPIO_OUTPUT);
    pinMode(GPIOA, STEPPINB, GPIO_OUTPUT);
    pinMode(GPIOA, DIRPINB, GPIO_OUTPUT);
    pinMode(GPIOA, RESETB, GPIO_OUTPUT);

    pinMode(GPIOA, STEPPINC, GPIO_OUTPUT);
    pinMode(GPIOA, DIRPINC, GPIO_OUTPUT);
    pinMode(GPIOA, RESETC, GPIO_OUTPUT);

    digitalWrite(GPIOA, STEPPINA, GPIO_LOW);
    digitalWrite(GPIOA, DIRPINA, GPIO_HIGH);
    digitalWrite(GPIOA, RESETA, GPIO_HIGH);

    digitalWrite(GPIOA, STEPPINB, GPIO_LOW);
    digitalWrite(GPIOA, DIRPINB, GPIO_HIGH);
    digitalWrite(GPIOA, RESETB, GPIO_HIGH);

    digitalWrite(GPIOA, STEPPINC, GPIO_LOW);
    digitalWrite(GPIOA, DIRPINC, GPIO_HIGH);
    digitalWrite(GPIOA, RESETC, GPIO_HIGH);

    float test180  = 3.14;
    int pulseTest180 = floor(test180/STEPANGLE);
    volatile int k = 0;
    volatile int j = 0;
    while(1) {
```

```
while(j<2) {
   while(k<2) {
   digitalWrite(GPIOA, DIRPINA, GPIO_HIGH);
   for(int i = 0; i< pulseTest180; i++){
      pulse(GPIOA, STEPPINA);
   }
   digitalWrite(GPIOA, DIRPINA, GPIO_LOW);
   for(int i = 0; i< pulseTest180; i++){
      pulse(GPIOA, STEPPINA);
   }
   k++;
   }
   digitalWrite(GPIOA, RESETA, GPIO_LOW);
   delay_millis(TIM2, 1000);
   digitalWrite(GPIOA, RESETA, GPIO_HIGH);
   k =0;
   j++;
}
j = 0;
while(j<2) {
   while(k<2) {
   digitalWrite(GPIOA, DIRPINB, GPIO_HIGH);
   for(int i = 0; i< pulseTest180; i++){
      pulse(GPIOA, STEPPINB);
   }
   digitalWrite(GPIOA, DIRPINB, GPIO_LOW);
   for(int i = 0; i< pulseTest180; i++){
      pulse(GPIOA, STEPPINB);
   }
   k++;
   }
   digitalWrite(GPIOA, RESETB, GPIO_LOW);
   delay_millis(TIM2, 1000);
   digitalWrite(GPIOA, RESETB, GPIO_HIGH);
   k =0;
   j++;
}
j =0;
while(j<2) {
   while(k<2) {
   digitalWrite(GPIOA, DIRPINC, GPIO_HIGH);
   for(int i = 0; i< pulseTest180; i++){
      pulse(GPIOA, STEPPINC);
   }
```

```
        digitalWrite(GPIOA, DIRPINC, GPIO_LOW);
        for(int i = 0; i< pulseTest180; i++){
            pulse(GPIOA, STEPPINC);
        }
        k++;
        }
        digitalWrite(GPIOA, RESETC, GPIO_LOW);
        delay_millis(TIM2, 1000);
        digitalWrite(GPIOA, RESETC, GPIO_HIGH);
        k =0;
        j++;
    }
}

 //testing moving 2 motors to different angles at the same time
float testMotorA180 = 3.14;
float testMotorC360 = 6.28;
volatile int pulsesA = floor((testMotorA180)/STEPANGLE);
volatile int pulsesC = floor((testMotorC360)/STEPANGLE);
volatile int firstModA = 0;
volatile int firstModC = 0;
volatile int secondModA = 0;
volatile int secondModC = 0;

while(1) {
    int maxPulse = testMotorC360;
    firstModA = floor(maxPulse/pulsesA);
    if (maxPulse - firstModA*pulsesA != 0) {
        secondModA = floor(maxPulse/(maxPulse - firstModA*pulsesA));
    }
    firstModC = floor(maxPulse/pulsesC);
    if (maxPulse - firstModC*pulsesC != 0) {
        secondModC = floor(maxPulse/(maxPulse - firstModC*pulsesA));
    }

    for(volatile int i = 0; i<maxPulse; i++){ //should i be adding a wait after all of these? how
long?
        if((firstModA != 0) && (i%firstModA == 0)){
            pulse(GPIOA, STEPPINA);
        }
        if ((secondModA != 0) && (i%secondModA == 0)){
            pulse(GPIOA, STEPPINA);
        }
        if((firstModC != 0) && (i%firstModC == 0)){
```

```
            pulse(GPIOA, STEPPINC);
        }
        if ((secondModC != 0) && (i%secondModC == 0)){
            pulse(GPIOA, STEPPINC);
        }

    }
  }


  return 0;
}


int main(void) { //running the actual code for a given array of angles and user input

  RCC->APB1ENR.TIM2EN = 1;
  // Configure flash latency and set clock to run at 84 MHz
  configureFlash();
  configureClock();


  //configure the timer for the pulses
  initTIM(TIM2);
  // Enable GPIOA
  RCC->AHB1ENR.GPIOAEN = 1;
  RCC->AHB1ENR.GPIOBEN = 1;
  RCC->AHB1ENR.GPIOCEN = 1;



  // direction and step pins
  pinMode(GPIOA, STEPPINA, GPIO_OUTPUT);
  pinMode(GPIOA, STEPPINB, GPIO_OUTPUT);
  pinMode(GPIOA, STEPPINC, GPIO_OUTPUT);

  pinMode(GPIOA, DIRPINA, GPIO_OUTPUT);
  pinMode(GPIOA, DIRPINB, GPIO_OUTPUT);
  pinMode(GPIOA, DIRPINC, GPIO_OUTPUT);

  pinMode(GPIOA, RESETA, GPIO_OUTPUT);
  pinMode(GPIOA, RESETB, GPIO_OUTPUT);
  pinMode(GPIOA, RESETC, GPIO_OUTPUT);
```

```
    pinMode(GPIOA, FPGAPIN, GPIO_INPUT);

    digitalWrite(GPIOA, STEPPINA, GPIO_LOW);
    digitalWrite(GPIOA, STEPPINB, GPIO_LOW);
    digitalWrite(GPIOA, STEPPINC, GPIO_LOW);

    digitalWrite(GPIOA, DIRPINA, GPIO_HIGH);
    digitalWrite(GPIOA, DIRPINB, GPIO_HIGH);
    digitalWrite(GPIOA, DIRPINC, GPIO_HIGH);

    while(1) {
        //want to set the reset pin high so the motors can take in angle changes
        digitalWrite(GPIOA, RESETA, GPIO_HIGH);
        digitalWrite(GPIOA, RESETB, GPIO_HIGH);
        digitalWrite(GPIOA, RESETC, GPIO_HIGH);
        //wait for user input for the next shape




        //need a case statement to choose the shape from the user input

        //initialize the variables for the angles
        volatile int i = 0;
        volatile float newAngleA = circleAngles[i][0];
        volatile float newAngleB = circleAngles[i][1];
        volatile float newAngleC = circleAngles[i][2];

        volatile float extraAngleA = 0;
        volatile float extraAngleB = 0;
        volatile float extraAngleC = 0;

        digitalWrite(GPIOA, DIRPINA, GPIO_HIGH);
        digitalWrite(GPIOA, DIRPINB, GPIO_HIGH);
        digitalWrite(GPIOA, DIRPINC, GPIO_HIGH);


        while(newAngleA != 0.0 || newAngleB != 0.0 || newAngleC != 0.0) {
            //calculate the number of pulses that each angle needs and tacking on the extra to the
next angle
            volatile int pulsesA = floor((newAngleA + extraAngleA)/STEPANGLE);
            extraAngleA = (newAngleA + extraAngleA) - ((newAngleA +
extraAngleA)/STEPANGLE)*STEPANGLE;
            volatile int pulsesB = floor((newAngleB + extraAngleB)/STEPANGLE);
```

```
        extraAngleB = (newAngleB + extraAngleB) - ((newAngleB +
extraAngleB)/STEPANGLE)*STEPANGLE;
        volatile int pulsesC = floor((newAngleC + extraAngleC)/STEPANGLE);
        extraAngleC = (newAngleC + extraAngleC) - ((newAngleC +
extraAngleC)/STEPANGLE)*STEPANGLE;

        volatile int firstModA = 0;
        volatile int firstModB = 0;
        volatile int firstModC = 0;
        volatile int secondModA = 0;
        volatile int secondModB = 0;
        volatile int secondModC = 0;


        volatile int maxPulse = (pulsesA > pulsesB ) ? (pulsesA > pulsesC ) ? pulsesA : pulsesC
: (pulsesB > pulsesC ) ? pulsesB : pulsesC;
        if (pulsesA != 0) {
           digitalWrite(GPIOA, DIRPINA, GPIO_HIGH);
           if (pulsesA < 0){
              digitalWrite(GPIOA, DIRPINA, GPIO_LOW);
              pulsesA = pulsesA*-1;
           }
           firstModA = floor(maxPulse/pulsesA);
           if (maxPulse - firstModA*pulsesA != 0) {
              secondModA = floor(maxPulse/(maxPulse - firstModA*pulsesA));
              extraAngleA += maxPulse - (maxPulse - firstModA*pulsesA)*secondModA;
           }
        }
        if(pulsesB != 0) {
           digitalWrite(GPIOA, DIRPINB, GPIO_HIGH);
           if (pulsesB < 0){
              digitalWrite(GPIOA, DIRPINB, GPIO_LOW);
              pulsesB = pulsesB*-1;
           }
           firstModB = floor(maxPulse/pulsesB);
           if (maxPulse - firstModB*pulsesB != 0) {
              secondModB = floor(maxPulse/(maxPulse - firstModB*pulsesB));
              extraAngleB += maxPulse - (maxPulse - firstModB*pulsesB)*secondModB;
           }
        }
        if (pulsesC != 0) {
           digitalWrite(GPIOA, DIRPINC, GPIO_HIGH);
           if (pulsesC < 0){
              digitalWrite(GPIOA, DIRPINC, GPIO_LOW);
```

```
            pulsesC = pulsesC*-1;
        }
        firstModC = floor(maxPulse/pulsesC);
        if (maxPulse - firstModC*pulsesC != 0) {
            secondModC = floor(maxPulse/(maxPulse - firstModC*pulsesC));
            extraAngleC += maxPulse - (maxPulse - firstModC*pulsesC)*secondModC;
        }
    }



    for(volatile int i = 0; i<maxPulse; i++){
        if((firstModA != 0) && (i%firstModA == 0)){
            pulse(GPIOA, STEPPINA);
        }
        if ((secondModA != 0) && (i%secondModA == 0)){
            pulse(GPIOA, STEPPINA);
        }

        if((firstModB != 0) && (i%firstModB == 0)){
            pulse(GPIOA, STEPPINB);
        }
        if ((secondModB != 0) && (i%secondModB == 0)){
            pulse(GPIOA, STEPPINB);
        }

        if((firstModC != 0) && (i%firstModC == 0)){
            pulse(GPIOA, STEPPINC);
        }
        if ((secondModC != 0) && (i%secondModC == 0)){
            pulse(GPIOA, STEPPINC);
        }

    }
    i++;
    newAngleA = circleAngles[i][0];
    newAngleB = circleAngles[i][1];
    newAngleC = circleAngles[i][2];
  }
 }
 return 0;
}
```