XY Plotter using DC Servo Motor

Vibhore Jain, DESE, IISc

ABSTRACT

This experiment is aimed at designing and controlling XY plotter using DC motors with servo control loop. Servo control in DC motor provides precise position control of rotor position. Such precision control is necessary to achieve position tracking in XY plane. The system has two degrees of freedom and thus require control of two DC servo motors independently. This report addresses design and control of coupled configuration of XY plotter which can be controlled manually using buttons or be driven by commands from a PC to emulate drawing.

I INTRODUCTION

XY position control is a problem statement in 2D space is spanned by using two independent actuators on a gantry supported by smooth rods and usually pulled by pulleys and belt or lead screws. These actuators can be coupled or decoupled with each other. Example of coupled and decoupled topologies is shown in Figure 2 and Figure 1 respectively.

While decoupled mechanism allows X and Y position to be controlled independently with separate actuators, such topology have mechanical challenges as they are asymmetric in terms of loading and control. The controller for DC motor for the two actuators are not same as both actuators drive different load. The XY tracking on the other hand doesn't require any manipulation of variables because position of each actuator determines the physical position in the XY plane.

Coupled mechanisms span the XY plane in a coupled manner, that is the two actuators need to work together to reach a point in XY plane. Changing the position of any one actuator, keeping the other actuator stationary causes the point of interest to change both X and Y coordinates. This coupling thus requires intermediate transformation to convert the XY coordinate into the coupled mechanism coordinate system. The coupled mechanism implemented in this experiment requires coordinate system to be rotated clockwise by 45°.

II TASKS

(a) Coupled system design

(b) Motor selection and interface

(c) Driver and control circuit

(d) Printed circuit board design

(e) Embedded firmware design

(f) Python script for XY plotting

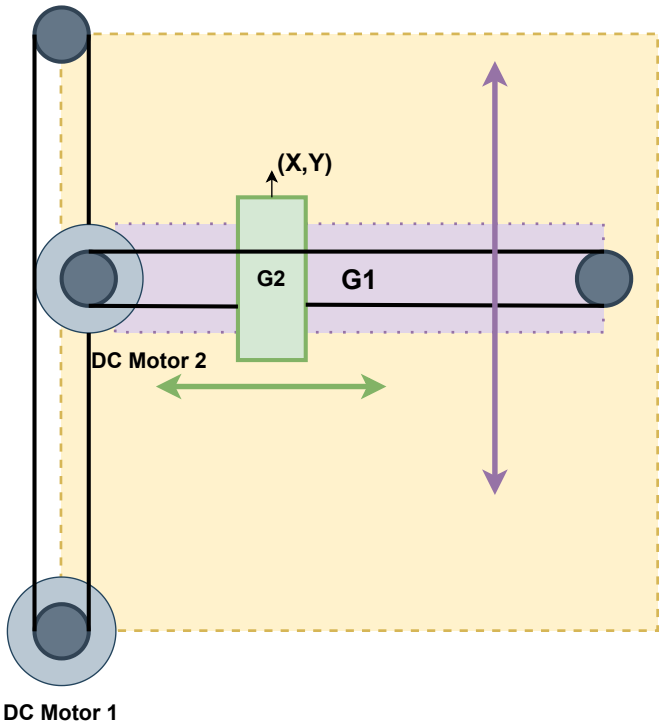

Figure 1: XY plotter with decoupled topology

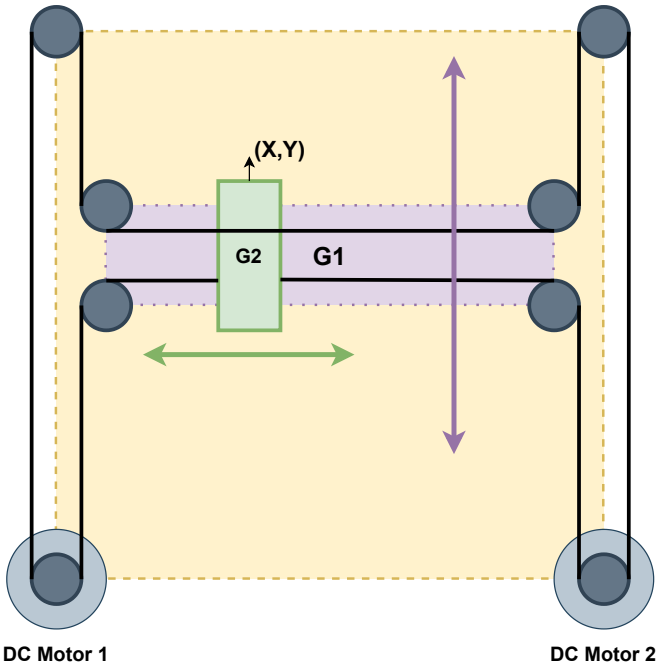

Figure 2: XY plotter with coupled topology

Task-0 ***Coupled system design***

The mechanical structure that was used to create the XY plotter is a de-coupled mechanism XY plotter sold by makeblocc. The original design used stepper motors in open loop configuration with belt drive to move X and Y gantry on smooth rods. While the original structure used two stepper motors to implement de-coupled mechanism, the general mechanical structure was suitable even for implementing coupled mechanism as well. That being said, the plotter was in pretty bad shape as all the parts including the smooth rods were rusted as shown in Figure 3. Two support beams were also missing from the original design which made the system prone to flexing. The pulley system was also not appropriate for implementing coupled configuration.

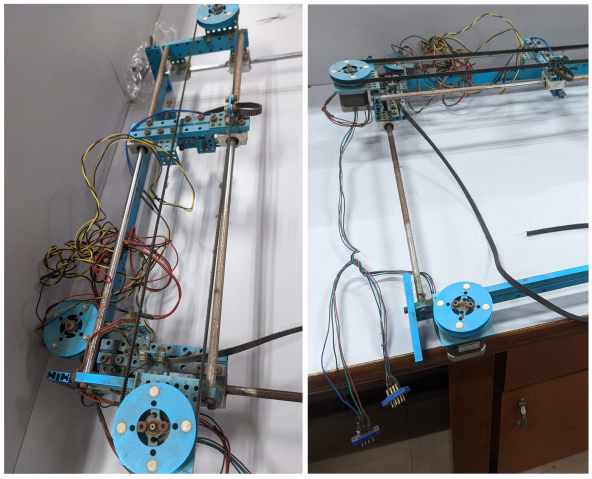

Figure 3: Original condition of XY plotter

The above mechanism was completely disassembled and parts were cleaned using soap water and sorted as depicted in Figure 4. The smooth steel rods were badly rusted and had to be treated with baking soda (shown in Figure 5 and sanded with fine grit sand paper. This step was necessary as over sanding or dry sanding could reduce the diameter of the steel rod and create wobble in the linear bearings.

Figure 4: Cleaning parts of XY plotter

Figure 5: Cleaning smooth rods using baking soda

After all the parts were cleaned and dry, they were assembled to make the stationary frame and gantry. Since support beams were missing from the original plotter, generic aluminium profiles were used to provide support and stiffen the plotter frame. Since there was no design guide to follow, the structure construction took a lot of trial and error making best use of available mechanical components. After the basic frame was ready, gantry G1 and G2 were constructed. Again generic support profiles were used wherever necessary to avoid flexing of the smooth rods. Electromagnet HCNE-0530 was used to actuate the pen assembly which was mounted on gantry 2. Finally gantry G1 and G2 were inserted in the main frame to complete the mechanical structure.

Coupled mechanism uses a single belt loop and required idler pulleys on gantry 1 and fixed frame. The idler pulleys on fixed frame are GT2 idler pulley with teeth where as the idler pulleys on gantry 1 are smooth. The endpoints of the belt terminate on gantry 2. Idler pulleys were thus installed at appropriate positions on gantry 1 and main frame to ensure that timing belt can slide smoothly without slipping/dislodging. All moving parts were treated with WD-40 to achieve smooth movements.

Task-1 ***Motor selection and interface***

Precision XY plotting requires low translation speeds (¡20mm/s) and high torque motors to prevent overshoots while still minimizing travel time. Position can be sensed using encoders so geared DC motors with encoders are suitable for this application. For this particular plotter GM25-370CA-12560

DC geared motor was chosen. The motor has maximum RPM of 370 at rated voltage of 12V. It includes a magnetic position encoder which can be interfaced with quadrature encoder peripherals of micro-controllers. The encoder doesn't come with reset/index pulse signal and hence homing has to be implemented separately using limit switches. Figure 6 shows the completed mechanical assembly of the coupled XY plotter.

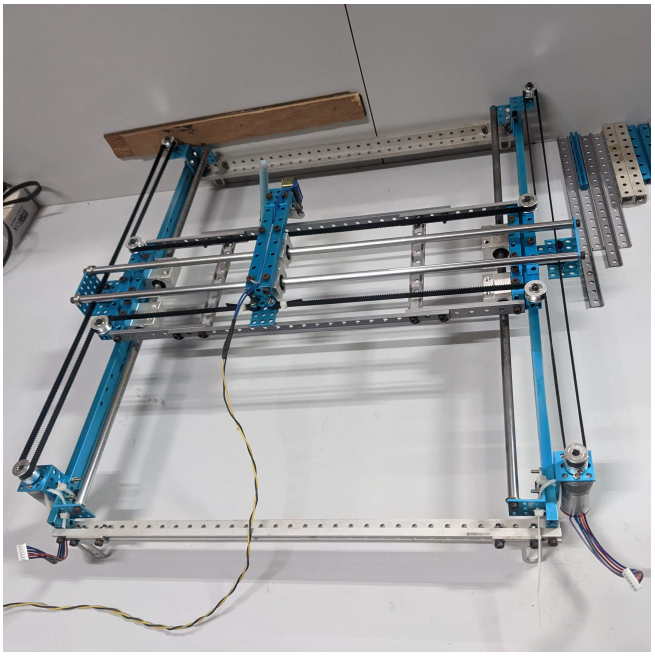

Figure 6: Completed mechanical assembly

Task-2 ***Driver and control circuit***

The control circuit for the XY plotter needs to interface with DC geared motor, their encoders, limit switch signals, solenoid actuation signal and PC for communication. Manual switches are a good add on to the control board for manually changing position of the pen. With all these IO requirements, control board was designed. The schematic of the control board is shown in Figure 10.

The headers for connecting the DC motor on the control board is in the same order as provided by the motor manufacturer to ensure out of the box compatibility. The current sense amplifier was implemented differently compared to previous implementations. The shunt resistor was placed in series with the motor and not in the tail of H bridge. Since the sense resistor is floating, a difference amplifier is required to sense voltage across the resistor which is then low pass filtered to remove switching noise. This allows bi-directional sensing of current flowing through the DC motor however since ADC on the TIVA board can only convert positive voltages, a DC offset was added to the output of sense circuit.

Since there are a lot of input-output signals on the control board, most signals are provided with indicator LEDs for quick debugging. The logic voltage for all signals is configurable between 3.3V and 5V to provide flexibility in terms of MCU selection.

Task-3 ***Printed circuit board design***

After verifying schematic, the board schematic was converted into layout for PCB design. All components used are through hole and hence a 2 layer PCB was sufficient to complete board layout. Signal traces are assigned a thickness of 20 mils and power traces 30 mils thick with 12 mil clearance. Top layer is poured with logic supply copper and bottom layer with ground net copper. Since the board required through hole copper plating which was difficult to be fabricated in department. The board dimensions at 19.3 cm x 12.4 cm which meant professionally manufacturing the board would be a costly affair, not to mention the long lead times.

It was decided that manufacturing the board would be done using traditional ferric chloride etching method for developing a double sided copper clad board. Electroplating vias is not possible in this method so all vias were increased in size (pad diameter to 1.6mm and drill diameter to 0.8mm) so that top and bottom traces can be stitched by passing copper wires through the holes and soldering on both sides. This also meant that all through hole components that usually required soldering on only bottom layer had to be soldered on both top and bottom layer doubling the number of solder points.

Figure 11 and 13 shows the top and bottom layer layout of the control board. Figure 12 shows the 3D render of the PCB with the components populated on the board. This view was crucial to give enough clearance for heat sinks for the voltage regulators and H bridge.

For PCB fabrication following steps were taken:

1. Copper clad is cut to required dimensions using craft cutter.
2. The clad piece is cleaned using soap water and abrasive dish cleaning sponge.
3. The clad is then painted black with matte black spray on both the sides and dried under 100W incandescent bulb.
4. The board is then aligned with the laser engraver axis and secured to the engraver bed.
5. Using LaserGRBL software, negative of PCB top layer is engraved on the paint coated copper clad.
6. The board is then flipped along y axis and secured back to the bed.
7. The negative of bottom layer is then flipped and engraved on the copper clad.
8. Once complete, the board is cleaned with toothbrush and toothpaste to remove all the engraved paint away while keeping tracks intact.
9. The copper clad board is then put in ferric chloride solution to etch away exposed copper on both sides of the PCB.
10. After etching completes, the board is washed with steel wool to remove paint coat and expose all the tracks.

11. Holes are then drilled in the board for through hole components and vias. Since this board required drilling 146 holes a mini bench press drill (Figure 7) was constructed to drill these holes quickly.

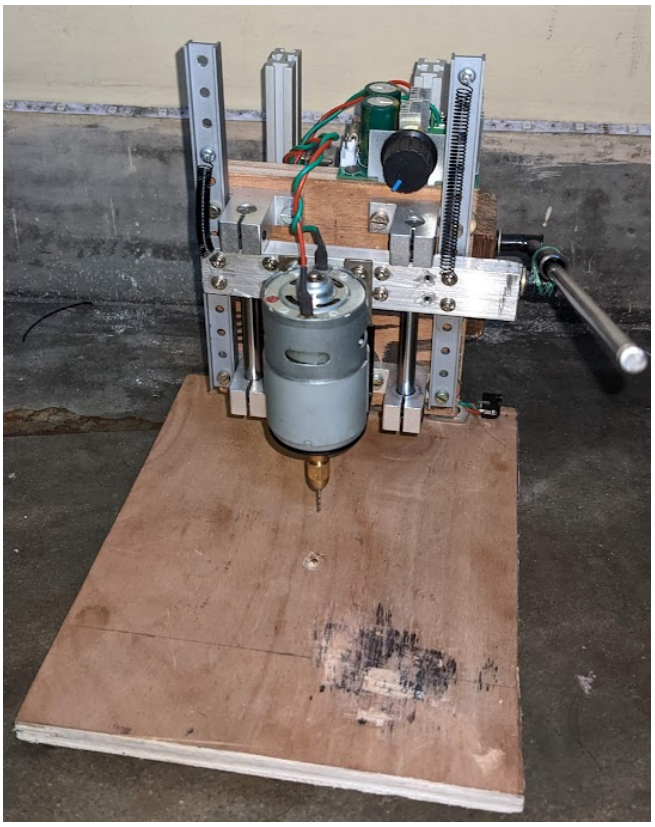

Figure 7: Drill Press for drilling holes and vias

Finally components were soldered on the board starting with power supply, IO circuits, headers and finally the IC sockets and L298N. The PCB was not very reliable as some traces required to be soldered on both top and bottom layer thus increasing debug time for the hardware. Also some components like the IC base did not allow two layer soldering and hence blue wiring had to be done. Although the PCB fabrication and assembly took over a week of time, it was much better in terms of reliability compared to jumper wire stitched circuits which often suffer from loose connections especially when kinetic parts are involved.

Task-4 ***Embedded firmware design***

The firmware on the TIVA board implements control system whose block diagram is shown in Figure 9. Since X and Y motors need to be controlled independently, each motor is controlled by its own PI control loop. This engages both ADC0 and ADC1 modules for sampling the current and QEI0 and QEI1 modules for sampling the angular position and velocity. Since the system is coupled, rotation of a single motor changes both X and Y axis. The truth table of rotation of the motor is summarised in Table 1. Anti-clockwise rotation is denoted by +1 and clock-wise rotation by -1. Stationary rotor is represented by 0.

	Left Motor	Right Motor	Pen Direction
1	0	0	Stationary
2	+1	0	North East
3	+1	+1	East
4	0	+1	South East
5	-1	+1	South
6	-1	0	South West
7	-1	-1	West
8	0	-1	North West
9	+1	-1	North

Table 1: Relation between motor rotation direction and pen direction

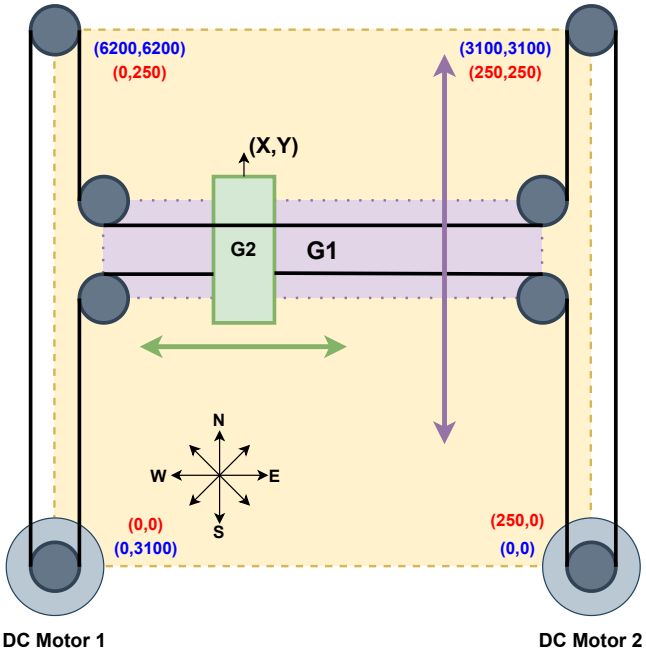

Figure 8: XY and LR coordinate system

Since the physical variables of interest that are finally manipulated to reach reference XY coordinates are the motor position which is given by the QEI position module, the QEI reference was set to 0 for left motor and 3100 for right motor. The value 3100 was obtained empirically by sweeping the pen position manually across the four end points of the XY plane and noting down the QEI position register. Figure 8 shows the QEI module values (in blue) and the XY values (in red) of the end points. It is easy to see that the motor plane (hence forth LR plane) is at an angle of 45° in anti clockwise direction with respect to XY plane. It is possible to decompose all vectors in XY plane to LR plane using simple trigonometry. Only $cos(45°)$ is required for this calculation. Thus all errors can be calculated in XY plane and then transformed into LR plane. It is also possible to transform all points in XY to LR plane and then calculate the errors. The latter approach was implemented

in the embedded firmware as the value of $cos(45^\circ)$ is also not required to be stored separately. It gets absorbed into scaling constants that convert the motor angle (position) into length. Equations 1 and 2 provide the relation of a point in XY plane (unit is millimeters) and its coordinates in LR plane.

$$Y = (L + R - 3100) \times \frac{250}{6200} \tag{1}$$

$$X = (L - R + 3100) \times \frac{250}{6200} \tag{2}$$

Above equations can be solved in terms of X and Y to get corresponding L and R values as shown in equations 3 and 4.

$$L = (Y + X) \times \frac{6200}{500} \tag{3}$$

$$R = (Y - X) \times \frac{6200}{500} + 3100 \tag{4}$$

To handle movement using physical buttons, each button changes the X or Y reference coordinate by 2mm and the control loop then tries to catch up with this perturbed value of X and Y reference coordinates. While manual control only allows for the pen to move in 8 directions, it is possible to move in arbitrary direction by changing the X and Y reference points. To achieve this, UART is used to update the XY reference coordinates which cascade down to LR reference coordinates. Once the plotter reaches the XY coordinate it sends an acknowledgment message back to the PC indicating that it is ready to receive a new coordinate position. The UART message handler also implements pen control so that every time the character "P" is received, the solenoid attached to pen is energised and pen is brought down. When the character "p" is received the solenoid is de-energised and the spring attached to the solenoid pulls the pen back. The firmware thus allows control of the XY plotter using serial terminal apps like putty, realterm, Arduino IDE or even the built in Serial terminal of CCS studio. For real-time sketching and plotting however, a custom python script was developed to track mouse clicks and movements and send the information to the TIVA board over serial. The embedded firmware is documented in code listing V.

Task-5 ***Python script for XY plotting***

The python script used for controlling XY plotter using mouse movements is documented in code listing VI. It begins by recording two diagonal points using two point calibration routine and maps any pixel within the range of the rectangle created by the diagonal into 250mm by 2500mm XY coordinate system. It uses a queue data structure to hold X and Y values temporarily along with pen up and down commands. Every time the mouse pointer is left clicked and moved in the valid XY range, the script records the corresponding XY pixel values and appends it to the queue. The while loop waits for acknowledgement from the XY plotter before sending new reference positions to the plotter thus ensuring that the pen follows the trajectory which is same as that of the cursor.

III CONCLUSIONS

The de-coupled XY plotter was more of a mechanical challenge to ensure that all the parts and mechanisms are smooth and free of flexing. Following observations were made in due course of completing this lab experiment.

- Apart from the joy of solving technical challenges, there is a philosophical satisfaction associated with restoration of old machines.
- Errors due to flexing and twisting of mechanical parts is not corrected by the XY plotter as the control loop closes from the motor position and doesn't include information of actual X and Y coordinates.
- Anti-windup provisions are absolutely essential for clamping output of every PI loop in order to prevent over shoots and instability problems.
- Coupled XY plotter are very simple to control as the motors require two identical PI loops with same parameters.
- As the whole system is driven by a single belt loop, the left and right motors experience the same belt tension which is not the case in decoupled systems.
- The errors in XY position that arise due to initial offset can be calibrated by using camera or equivalent optical sensors.
- Precision XY plotting ability can be used to manufacture PCBs (as described in section II)
- Since fabrication principles of PCB and semiconductor ICs are similar, it should be possible to fabricate ICs on silicon wafer using XY laser plotter instead of film based photo-plotting methods.

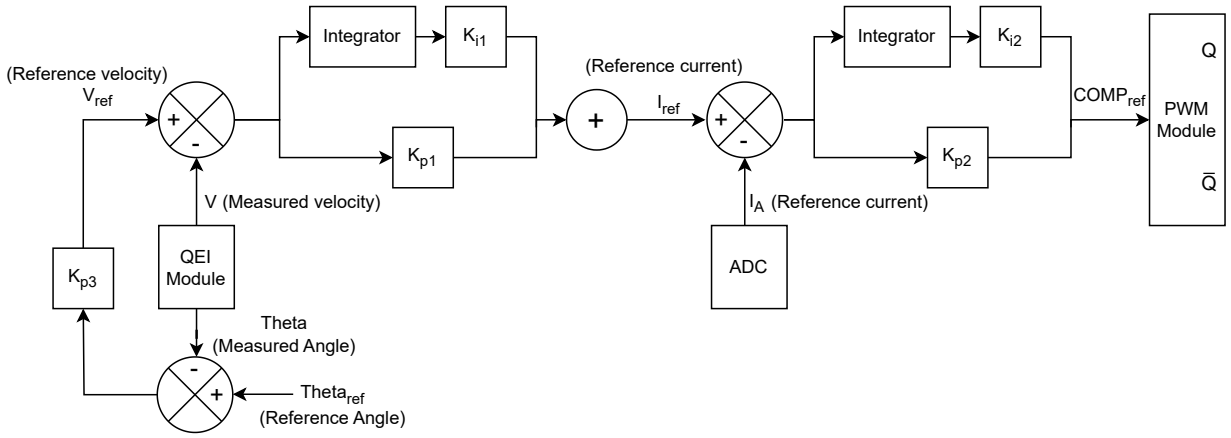

Figure 9: Closed loop controller for position control of DC motor

IV SCHEMATIC & LAYOUT

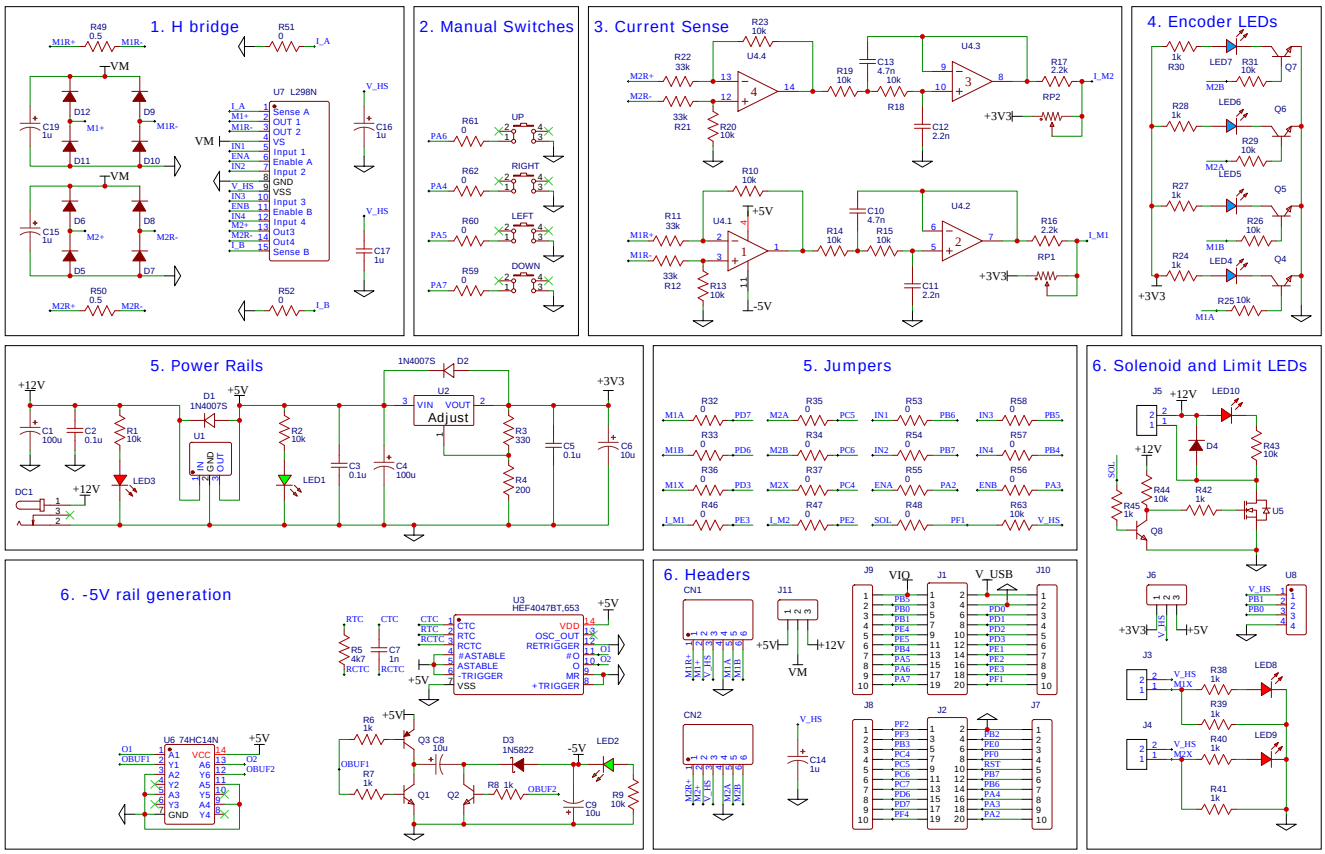

Figure 10: Schematic of control board for XY plotter

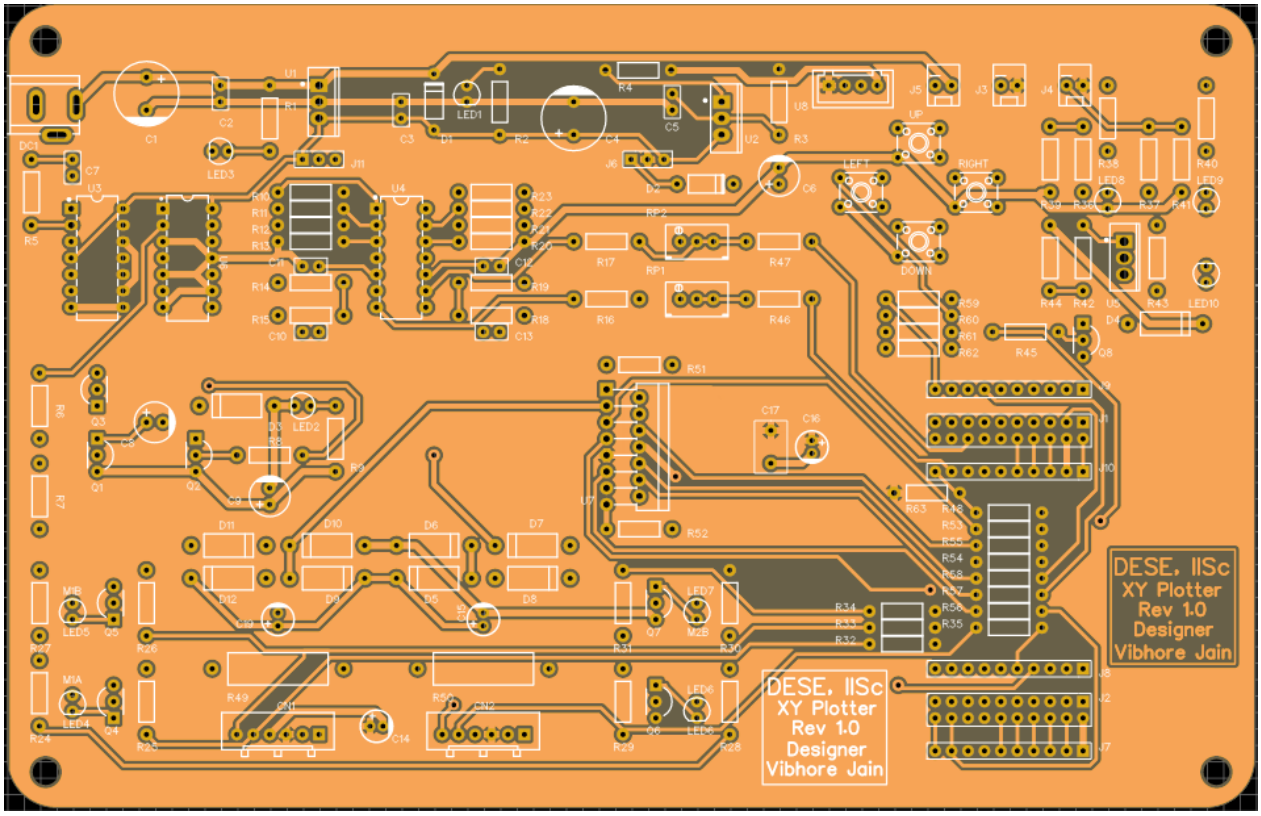

Figure 11: Top Layer layout of control board

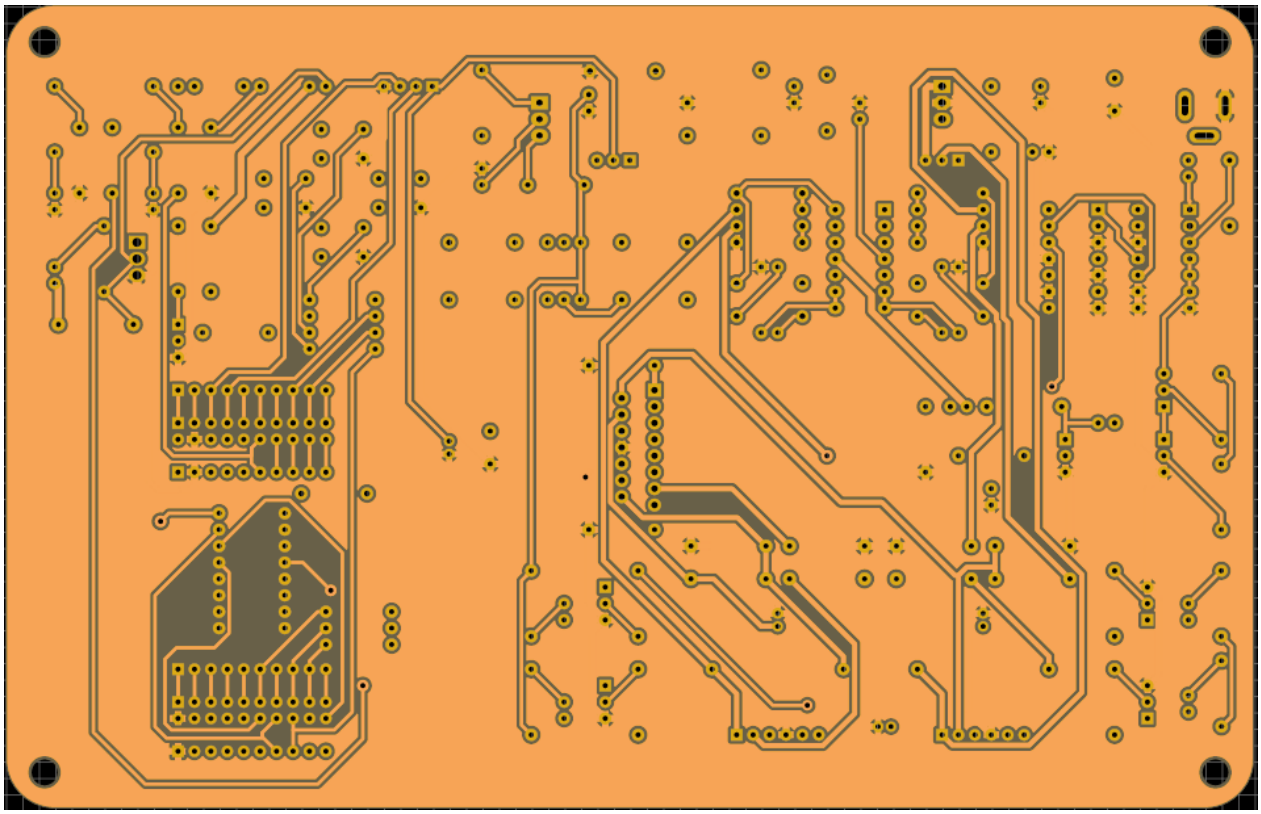

Figure 12: Bottom Layer layout of control board

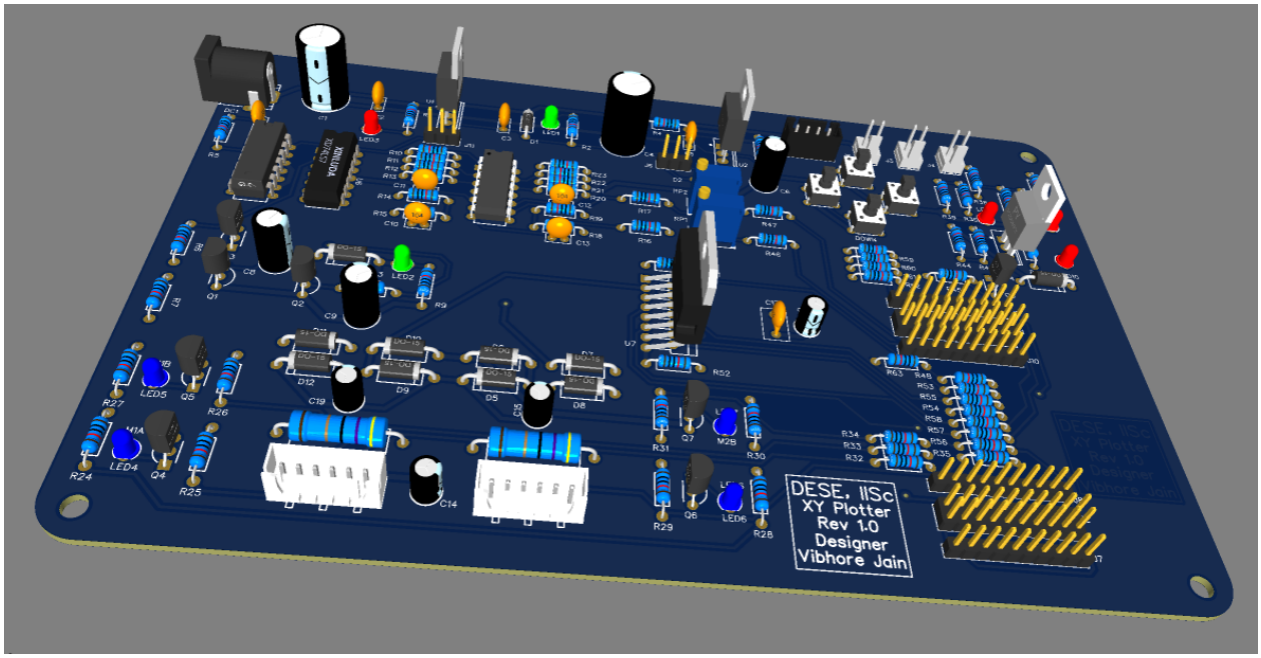

Figure 13: 3D render of control board

V C CODE

```c
#include <stdint.h>
#include <stdlib.h>
#include "inc/tm4c123gh6pm.h"

#include <stdint.h>
#include <stdbool.h>
#include <./inc/tm4c123gh6pm.h>
#include <inc/hw_memmap.h>
#include <inc/hw_types.h>
#include <driverlib/gpio.h>
#include <driverlib/pin_map.h>
#include <driverlib/sysctl.h>
#include <driverlib/uart.h>

#define pen         1 // Pen on PF1
#define green       3 // Green LED on PF2
#define blue        2 // Blue LED on PF3

#define m1a         7 // QEI motor1 A signal on PD7
#define m1b         6 // QEI motor1 B signal on PD6
#define x_lim       3 // X limit switch on PD3
#define m2a         5 // QEI motor2 A signal on PC5
#define m2b         6 // QEI motor2 B signal on PC6
#define y_lim       4 // Y limit switch on PC4

#define Im1         3 // Motor 1 current sense on PE3
#define Im2         2 // Motor 2 current sense on PE2

#define IN1         6 // IN1 on PB6 for Motor 1
#define IN2         7 // IN2 on PB7 for Motor 1
#define ENA         2 // Enable for motor 1 on PA2
#define IN3         5 // IN3 on PB5 for Motor 1
#define IN4         4 // IN4 on PB4 for Motor 1
#define ENB         3 // Enable for motor 2 on PA3

#define sw_up       6 // Up button on PA6
#define sw_right    4 // Right button on PA4
#define sw_down     7 // Down button on PA7
#define sw_left     5 // Left button on PA5

#define abs_tol     0.2   // Tolerance allowed by the CNC in mm
#define bed_range   250.0 // X and Y maximum travel is 250mm
#define QEI_LIM     6200  // Max Limit of encoder starting from 0
#define PPR         116   // Pulse per revolution for encoder
#define MAX_POS     0xFFFFFFFF // No reset of QEI position reg
#define fcpu        16000000
#define f_speed     100
#define edges       4   // Fouredges per pulse of A and B
#define pwm_top     1600

#define Kp_3 5      // Proportional Gain for QEI code error to get omega ref
#define Kp_2 10      // Proportional Gain for Omega error
#define Kp_1 1      // Proportional Gain for Current error

#define Ki_2 5      // Integral gain for Omega error
#define Ki_1 1      // Integral gain for Current error
#define Ts   0.001  // 1 milli-second (1kHz sampling)

float x_pos,y_pos;  // Current position of pen in XY basis
float x_ref,y_ref;  // Reference position to track in XY basis
float omega_0, omega_1; // Current angular speed of motors
int qei0_ref,qei1_ref;  // Reference QEI positions to track

int adc_offset0, adc_offset1; // Offset values to subtract from ADC results

float int_I0,int_I1;    // Current error integrator for motors
float int_RPM_0, int_RPM_1; // Angular error velocity integrator for motors


```

```
70
71 float v;          // Current speed in rotations per minute
72 float v_ref;      // Reference velocity in rotations per minute
73 float int_v;      // Velocity error accumulator
74 float pwm_val;    // Variable to store PWM value
75 float Ia;         // Measured current in mA
76 float Ia_ref;     // Reference current in mA to be supplied to the motor
77 float int_i;      // Current error accumulator
78 int uart_flag;    // Flag to indicate UART buffer ready to be processed
79 float theta;      // Current angle of motor
80 float theta_ref;  // rotation angle in milli-degrees
81
82 int lock_flag;
83 int dir;
84 char arr[20];               // UART buffer
85 int buff_index;             // UART buffer index
86 int uart_flag;              // Flag to indicate if UART buffer is ready for processing
87 int print_flag;             // Flag to indicate if new values are to be printed
88
89
90 static inline void enable_irq(void);
91 static inline void disable_irq(void);
92
93 void delayMs(int n);
94 void process_command(void);
95 //Send an ACK number between 0 and 9 over UART
96 void uart_send_ack(int x)
97 {
98     UARTCharPut(UART0_BASE,'0'+x);
99     UARTCharPut(UART0_BASE,'\n');
100 }
101
102 void configure_uart()
103 {
104         SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
105         SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
106         GPIOPinConfigure(GPIO_PA0_U0RX);
107         GPIOPinConfigure(GPIO_PA1_U0TX);
108         GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
109         UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,\
110         (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
111         UART0_IM_R |= (1<<4)|(1<<6) ;
112         UART0_IFLS_R = 0;
113         NVIC_EN0_R |= (1<<5);
114 }
115
116 void configure_systick()
117 {
118     NVIC_ST_CTRL_R      = 0;              // Disable timer during setup
119     NVIC_ST_RELOAD_R    = 320000-1;       // This should expire once every 20ms
120     NVIC_ST_CTRL_R      = 0x00000007;     // Clock src set to system clock and enable timer and interrupt
121 }
122 //Configure ADC0 and ADC1 to sample motor currents
123 //Triggered by Timer1
124 void configure_adc()
125 {
126         //ADC0 Configuration
127         SYSCTL_RCGCADC_R    |= 1;         // Enable clock to ADC0
128         delayMs(5);
129         ADC0_ACTSS_R    &= ~8;            // Disable SS3 during configuration
130         ADC0_EMUX_R     |= 0x5000;        // GPTM triggers the sequencer
131         ADC0_SSMUX3_R    = 0;             // Get input from channel 0
132         ADC0_SSCTL3_R   |= 6;             // Take one sample at a time, set flag at 1st sample
133         ADC0_ACTSS_R    |= 8;             // Enable ADC0 sequencer 3
134         //ADC0_IM_R     |= (1<<3);        // Enable ADC0 sequencer 3 interrupt
135         ADC0_SAC_R       = 6;             // Enable 64 point hardware averaging :)
136         //NVIC_EN0_R      |= ((1<<17));     // Enable ADC0 sequencer 3 interrupt
137
138         //ADC1 Configuration
139         SYSCTL_RCGCADC_R    |= 2;         // Enable clock to ADC1
140         delayMs(5);
```

```
        ADC1_ACTSS_R    &= ~8;              // Disable SS3 during configuration
        ADC1_EMUX_R     |= 0x5000;         // GPTM triggers the sequencer
        ADC1_SSMUX3_R    = 1;              // Get input from channel 1
        ADC1_SSCTL3_R   |= 6;              // Take one sample at a time, set flag at 1st sample
        ADC1_ACTSS_R    |= 8;              // Enable ADC1 sequencer 3
       //ADC1_IM_R      |= (1<<3);        // Enable ADC1sequencer 3 interrupt
        ADC1_SAC_R       = 6;              // Enable 64 point hardware averaging :)
}

// Configure Timer1 to generate sampling signals for ADC and interrupt for
// Servicing PI loops
void configure_timer(void)
{
    SYSCTL_RCGCTIMER_R |= (1<<1);   // Enable clock Timer1 sub-timer A in run mode
    TIMER1_CTL_R        = 0;        // Disable timer1 output
    TIMER1_CFG_R        = 0x4;      // Select 16-bit configuration option
    TIMER1_TAMR_R       = ((1<<1)|\
                           (1<<10)|\
                           (1<<5));  // select down count mode
    TIMER1_TAILR_R      = fcpu*Ts-1;// TimerA counter starting count down value
    TIMER1_ICR_R       |= 0x01;     // TimerA timeout flag bit clears
    TIMER1_IMR_R        =((1<<0));  // Enable TimerA time-out interrupt mask
    TIMER1_CTL_R       |= (1<<5);   // Enable TimerA trigger to ADC
    NVIC_EN0_R         |= ((1<<21));// Enable Timer1 sub-timer A interrupt
    TIMER1_CTL_R       |= 0x01;     // Enable TimerA
}

void configure_pen_gpio()
{
    SYSCTL_RCGC2_R      |= 0x00000020;                    // enable clock to GPIOF
    GPIO_PORTF_LOCK_R    = 0x4C4F434B;                    // unlock commit register
    GPIO_PORTF_DIR_R    |= (1<<pen);                      // set pen pin as output pin
    GPIO_PORTF_DEN_R    |= (1<<pen);                      // set pen pin as digital pin
    GPIO_PORTF_DIR_R    |= ((1<<green)|(1<<blue));        // set pen pin as output pin
    GPIO_PORTF_DEN_R    |= ((1<<green)|(1<<blue));        // set pen pin as digital pin
}
// Configure QEI module 0 and 1 to work with the motors
void configure_qei_gpio()
{
//  QEI0 GPIO Configuration
    SYSCTL_RCGC2_R      |=  ((1<<3));                   // Enable clock to port D
    GPIO_PORTD_LOCK_R    =  0x4c4f434B;                 // Unlock port D commit register
    GPIO_PORTD_CR_R     |=  ((1<<7));                   // Make PD7 configurable
    GPIO_PORTD_AFSEL_R  |=  ((1<<m1a)|(1<<m1b));        // Alternate function select for PD7 and PD6
    GPIO_PORTD_PCTL_R   |=  ((6<<28)|(6<<24));          // Select QEI0 signals on PD7 and PD6
    GPIO_PORTD_DIR_R    &= ~((1<<m1a)|(1<<m1b));        // Set PD7 and PD6 as inputs
    GPIO_PORTD_DEN_R    |=  ((1<<m1a)|(1<<m1b));
//  QEI1 GPIO Configuration
    SYSCTL_RCGC2_R      |=  ((1<<2));                   // Enable clock to port C
    GPIO_PORTC_LOCK_R    =  0x4c4f434B;                 // Unlock port C commit register
    GPIO_PORTC_AFSEL_R  |=  ((1<<m2a)|(1<<m2b));        // Alternate function select for PC5 and PC6
    GPIO_PORTC_PCTL_R   |=  ((6<<20)|(6<<24));          // Select QEI1 signals on PC5 and PC6
    GPIO_PORTC_DIR_R    &= ~((1<<m2a)|(1<<m2b));        // Set PC5 and PC6 as inputs
    GPIO_PORTC_DEN_R    |=  ((1<<m2a)|(1<<m2b));
}

void configure_adc_gpio()
{
    SYSCTL_RCGCGPIO_R   |= 0x10;      // Enable clock to PE (AIN0 is on PE3)
    // Initialize PE3 for AIN0 input
    GPIO_PORTE_AFSEL_R  |= 0x0F;      // Enable alternate function
    GPIO_PORTE_DEN_R    &= (~0x0F);   // Disable digital function
    GPIO_PORTE_AMSEL_R  |= 0x0F;      // Enable analog function
}
// Configure pins for controlling dual H bridge (L298N)
void configure_DHB_gpio()
{
    SYSCTL_RCGC2_R      |= 0x03;                // Enable clock to port A and port B
    GPIO_PORTA_DIR_R    |= (1<<ENA)|(1<<ENB);   // Set ENA, ENB to output
    GPIO_PORTA_DEN_R    |= (1<<ENA)|(1<<ENB);   // set ENA and ENB as digital pins
    GPIO_PORTB_AFSEL_R  |= (1<<IN1)|(1<<IN2);   // IN1 and IN2 alternate function
```

```
    GPIO_PORTB_AFSEL_R  |= (1<<IN3)|(1<<IN4);   // IN3 and IN4 alternate function
    GPIO_PORTB_PCTL_R   |= 0x44000000;          // make IN1 and IN2 PWM output pin
    GPIO_PORTB_PCTL_R   |= 0x00440000;          // make IN3 and IN4 PWM output pin
    GPIO_PORTB_DEN_R    |= (1<<IN1)|(1<<IN2);   // set IN1 and IN2 as digital pins
    GPIO_PORTB_DEN_R    |= (1<<IN3)|(1<<IN4);   // set IN1 and IN2 as digital pins
    GPIO_PORTA_DATA_R   &= ~(1<<ENA);           // ENA = LOW
    GPIO_PORTA_DATA_R   &= ~(1<<ENB);           // ENB = LOW
}
// Configure GPIO for X and Y Limit switches
void configure_xys_gpio()
{
    SYSCTL_RCGC2_R      |= 0x00000004;       // enable clock to GPIOC
    GPIO_PORTC_LOCK_R    = 0x4C4F434B;       // Unlock PORTC to configure the GPIO
    GPIO_PORTC_DEN_R    |= (1<<y_lim);       // enable digital function on y limit pin
    SYSCTL_RCGC2_R      |= 0x00000008;       // enable clock to GPIOD
    GPIO_PORTD_DEN_R    |= (1<<x_lim);       // enable digital function on x limit pin
}
// Configure GPIOs connected to manual control buttons
void configure_btn_gpio()
{
    SYSCTL_RCGC2_R      |= 0x00000001;       // enable clock to GPIOA
    GPIO_PORTA_LOCK_R    = 0x4C4F434B;       // Unlock PORTA to configure the GPIO
    GPIO_PORTA_DEN_R    |= (1<<sw_up)\
                           |(1<<sw_down)\
                           |(1<<sw_left)\
                           |(1<<sw_right);   // enable digital function on push button gpios
    GPIO_PORTA_PUR_R    |= (1<<sw_up)\
                           |(1<<sw_down)\
                           |(1<<sw_left)\
                           |(1<<sw_right);   // enable internal pull ups on push button gpios
}
// Master function for GPIO configuration
void configure_gpio()
{
    configure_pen_gpio();
    configure_qei_gpio();
    configure_adc_gpio();
    configure_DHB_gpio();
    configure_xys_gpio();
    configure_btn_gpio();
}
// Energize solenoid to touch down the pen
static inline void pen_down()
{
    GPIO_PORTF_DATA_R &= ~(1<<pen);
}
// De-energize solenoid to lift up the pen
static inline void pen_up()
{
    GPIO_PORTF_DATA_R |= (1<<pen);
}
// PWM0 GEN0 and GEN1 configuration for controlling motors
void configure_pwm()
{
    SYSCTL_RCGCPWM_R    |= SYSCTL_RCGCPWM_R0;        // Enable clock to PWM0 module
    SYSCTL_RCC_R        &= ~(SYSCTL_RCC_USEPWMDIV);  // Use system clock for PWM modules
////////////////////////////////////////////////////////////////////////////////////////////
//  Configure PWM0 generator 0
    PWM0_0_CTL_R        &= ~1;                       // Disable PWM0 GEN0 module
    PWM0_0_CTL_R        |= ((1<<2)|(1<<1));          // PWM0 GEN0 module in up-down mode
    //  PWM0_0A is IN1 signal
    PWM0_0_GENA_R       |= (0x2<<10);                // PWMA goes low on comp B falling match
    PWM0_0_GENA_R       |= (0x3<<8);                 // PWMA goes high on comp B rising match
    //  PWM0_0B is IN2 signal
    PWM0_0_GENB_R       |= (0x3<<10);                // PWMB goes high on comp B falling match
    PWM0_0_GENB_R       |= (0x2<<8);                 // PWMB goes low on comp B rising match
    PWM0_0_LOAD_R       = pwm_top-1;                 // Load counter top value to pwm_top
    PWM0_0_CMPB_R       = pwm_top/2 - 1;
////////////////////////////////////////////////////////////////////////////////////////////
//  Configure PWM0 generator 1
    PWM0_1_CTL_R        &= ~1;                       // Disable PWM0 GEN1 module
```

```
    PWM0_1_CTL_R        |= ((1<<2)|(1<<1));          // PWM0 GEN1 module in up-down mode
    //  PWM0_0A is IN4 signal
    PWM0_1_GENA_R       |= (0x3<<10);               // PWMA goes high on comp B falling match
    PWM0_1_GENA_R       |= (0x2<<8);                // PWMA goes low on comp B rising match
    //  PWM0_0B is IN3 signal
    PWM0_1_GENB_R       |= (0x2<<10);               // PWMB goes low on comp B falling match
    PWM0_1_GENB_R       |= (0x3<<8);                // PWMB goes high on comp B rising match
    PWM0_1_LOAD_R        = pwm_top-1;               // Load counter top value to pwm_top
    PWM0_1_CMPB_R        = pwm_top/2 - 1;
////////////////////////////////////////////////////////////////////////////////////////////

// Configure output channels and enable PWM0 GEN0 and GEN1 modules
    PWM0_ENABLE_R       |= 0x0F;                    // Enable PWM0 channel 0,1,2 and 3 output
    PWM0_0_CTL_R        |= 1;                       // Enable PWM0_0
    PWM0_1_CTL_R        |= 1;                       // Enable PWM0_0

}

void configure_qei()
{
//QEI Module 0 configuration
    SYSCTL_RCGCQEI_R    |= ((1<<0));                    // Enable clock to QEI module 0
    QEI0_CTL_R          |= ((1<<5)|(1<<3));             // Enable velocity capture, reset based on MAX_POS
                                                        // Capture edges on both PhA and PhB
    QEI0_MAXPOS_R       |= MAX_POS;
    QEI0_LOAD_R          = (fcpu/f_speed);                 // Update speed once every 10ms
    QEI0_CTL_R          |= ((1<<0));                    // Enable QEI module 0

//QEI Module 1 configuration
    SYSCTL_RCGCQEI_R    |= ((1<<1));                    // Enable clock to QEI module 1
    QEI1_CTL_R          |= ((1<<5)|(1<<3));             // Enable velocity capture, reset based on MAX_POS
                                                        // Capture edges on both PhA and PhB
    QEI1_MAXPOS_R       |= MAX_POS;
    QEI1_LOAD_R          = (fcpu/f_speed);              // Update speed once every 20ms (50Hz)
    QEI1_CTL_R          |= ((1<<0)|(1<<1));             // Enable QEI module 1 swap A and B
}
// Move in increaments of 2mm in the direction of
// manual button press.
void move()
{
    float x_temp = x_pos;
    float y_temp = y_pos;
    int bt_flag = 0;
    int button_read = GPIO_PORTA_DATA_R;
    GPIO_PORTA_DATA_R   |= (1<<ENA);          // ENA = LOW
    GPIO_PORTA_DATA_R   |= (1<<ENB);          // ENB = LOW
    if(!(button_read&(1<<sw_up)))
    {
        x_temp += 2;
        bt_flag |= 1;
    }
    if(!(button_read&(1<<sw_right)))
    {
        y_temp += 2;
        bt_flag |= 2;
    }
    if(!(button_read&(1<<sw_down)))
    {
        x_temp -= 2;
        bt_flag |= 1;
    }
    if(!(button_read&(1<<sw_left)))
    {
        y_temp -= 2;
        bt_flag |= 2;
    }
    if(bt_flag&1)
        if(x_temp < bed_range && x_temp > 1)
            x_ref = x_temp;
    if(bt_flag&2)
        if(y_temp < bed_range && y_temp > 1)
```

```
            y_ref = y_temp;
}
// Function to home the pen to 0,0 in XY xoordinate system as the
// encoders on motor are relative and not absolute. At the end of
// Homing QEI module position registers are initialised and any
// DC offset in current is recorded and subtracted for subsequent
// Calculations
void homing()
{
          GPIO_PORTA_DATA_R   |= (1<<ENA);           // ENA = HIGH
          GPIO_PORTA_DATA_R   |= (1<<ENB);           // ENB = HIGH
          while(!(GPIO_PORTC_DATA_R&(1<<y_lim))) // Move down until y limit switch is hit
          {
              PWM0_0_CMPB_R = pwm_clk;
              PWM0_1_CMPB_R = pwm_aclk;
          }
          while(!(GPIO_PORTD_DATA_R&(1<<x_lim))) // Move left until x limit switch is hit
          {
              PWM0_0_CMPB_R = pwm_clk;
              PWM0_1_CMPB_R = pwm_clk;
          }
          while(!(GPIO_PORTC_DATA_R&(1<<y_lim))) // Move down until y limit switch is hit
          {
              PWM0_0_CMPB_R = pwm_clk;
              PWM0_1_CMPB_R = pwm_aclk;
          }
          while(!(GPIO_PORTD_DATA_R&(1<<x_lim))) // Move left until x limit switch is hit
          {
              PWM0_0_CMPB_R = pwm_clk;
              PWM0_1_CMPB_R = pwm_clk;
          }
          PWM0_0_CMPB_R = pwm_stop;
          PWM0_1_CMPB_R = pwm_stop;
          delayMs(1000);
          QEI0_POS_R = 0;                            // Zero QEI0 position register
          QEI1_POS_R = 3100;                         // Mid  QEI1 position register
          x_pos = 0;                                 // Initialize x = 0
          y_pos = 0;                                 // Initialize y = 0
          adc_offset0 = ADC0_SSFIFO3_R;              // Zero current reading from ADC0
          adc_offset1 = ADC1_SSFIFO3_R;              // Zero current reading from ADC0
          delayMs(1000);
          adc_offset0 = ADC0_SSFIFO3_R;              // Zero current reading from ADC0
          adc_offset1 = ADC1_SSFIFO3_R;              // Zero current reading from ADC0
          uart_send_ack(0);                          // Send Homing Complete flag to PC
}
// Function to transform a point in LR coordinate to XY coordinate
static inline void update_coordinate()
{
    x_pos = (((int)(QEI0_POS_R-QEI1_POS_R+(QEI_LIM/2)))*bed_range)/QEI_LIM;
    y_pos = (((int)(QEI0_POS_R+QEI1_POS_R-(QEI_LIM/2)))*bed_range)/QEI_LIM;
}
// Function to transform a point in XY coordinate to LR coordinate
static inline void update_qei_pos()
{
    qei0_ref = (QEI_LIM*(x_ref+y_ref))/(2*250.0);
    qei1_ref = (QEI_LIM*(y_ref-x_ref))/(2*250.0)+(QEI_LIM/2);
}

void main_loop()
{
   // While loop keeps checking if absolute error in X and Y position
   // is less than absolute tolerance (0.2mm here) and sends an ACK
   // if that is the case. Also calls the move function to service
   // Movements reqeusted by push buttons
   while(1)
   {
      update_coordinate();
      if((abs(x_pos-x_ref) < abs_tol) && (abs(y_pos-y_ref) < abs_tol))
      {
          if(lock_flag == 0)
          {
```

```
                uart_send_ack(1);
                lock_flag = 1;
            }
        }
        else
          lock_flag = 0;


        if(uart_flag == 1)
            process_command();
        move();
    }
}

int main(void)
{
        disable_irq();
        configure_gpio();
        configure_adc();
        configure_qei();
        configure_timer();
        configure_uart();
        configure_pwm();
        pen_up();
        homing();
        x_ref = 125.0;
        y_ref = 125.0;
        int_RPM_0 = 0;
        int_RPM_1 = 0;
        enable_irq();
        main_loop();
}

static inline void enable_irq()
{
    __asm ("    CPSIE  I\n");
}

static inline void disable_irq()
{
    __asm ("    CPSID  I\n");
}

void SysTick_Handler(void)
{
  // Not used here, can be removed from code
}

// While I can calculate theta and then the position why not just
// skip that middle calculation and directly calculate x,y coordinates
// from QEI0 pos and QEI1 pos. As final errors are vectors in Cartesian plane.
// ISR for servicing PI loop
void GPTM1_Int_Handler()
{
    float omega_ref_0, omega_ref_1;
    float error_rpm_0, error_rpm_1;
    float loc_int_RPM0 = int_RPM_0;
    float loc_int_RPM1 = int_RPM_1;

    float I0_ref, I1_ref;
    float I0_val, I1_val;
    float error_I0, error_I1;
    float loc_int_I0 = int_I0;
    float loc_int_I1 = int_I1;

    int PWM_val_0, PWM_val_1;
    // RPM reference generation using Theta error
    update_qei_pos();
    omega_ref_0 = (float)(Kp_3*((int)((qei0_ref - QEI0_POS_R))));
    omega_ref_1 = (float)(Kp_3*((int)((qei1_ref - QEI1_POS_R))));

```

```
    // Calculating RPM and its error from QEI Speed register
    omega_0  = (float)((f_speed*QEI0_SPEED_R*60)/(PPR*edges));
    omega_1  = (float)((f_speed*QEI1_SPEED_R*60)/(PPR*edges));
    if(QEI0_STAT_R & 0x02)
        omega_0 = -1*omega_0;

    if(QEI1_STAT_R & 0x02)
        omega_1 = -1*omega_1;


    error_rpm_0 = omega_ref_0 - omega_0;
    error_rpm_1 = omega_ref_1 - omega_1;

    int_RPM_0 = int_RPM_0 + error_rpm_0*Ts;
    int_RPM_1 = int_RPM_1 + error_rpm_1*Ts;

    // Calculating reference current value from omega

    I0_ref =  Kp_2*error_rpm_0 + Ki_2*int_RPM_0;
    I1_ref =  Kp_2*error_rpm_1 + Ki_2*int_RPM_1;

    // Calculating current sense and its error using ADC value

    I0_val = ((int)(ADC0_SSFIFO3_R-adc_offset0))*(3300*22)/(4095*13);
    I1_val = ((int)(ADC1_SSFIFO3_R-adc_offset1))*(3300*22)/(4095*13);

    error_I0 = I0_ref-I0_val;
    error_I1 = I1_ref-I1_val;

    int_I0 = int_I0 + error_I0*Ts;
    int_I1 = int_I1 + error_I1*Ts;

    // Calculating PWM from current error
    PWM_val_0 = (pwm_top/2) + Kp_1*error_I0 + Ki_1*int_I0;
    PWM_val_1 = (pwm_top/2) + Kp_1*error_I1 + Ki_1*int_I1;

  // Logic for anti-windup of current and velocity PI loop
    if(PWM_val_0 > pwm_top)
    {
        PWM_val_0 = pwm_top-1;
        if(int_RPM_0 > loc_int_RPM0)
            int_RPM_0 = loc_int_RPM0;
        if(int_I0 > loc_int_I0)
            int_I0 = loc_int_I0;
    }
    if(PWM_val_0 < 0)
    {
        PWM_val_0 = 1;
        if(int_RPM_0 < loc_int_RPM0)
             int_RPM_0 = loc_int_RPM0;
        if(int_I0 < loc_int_I0)
             int_I0 = loc_int_I0;
    }

    if(PWM_val_1 > pwm_top)
    {
        PWM_val_1 = pwm_top-1;
        if(int_RPM_1 > loc_int_RPM1)
             int_RPM_1 = loc_int_RPM1;
        if(int_I1 > loc_int_I1)
             int_I1 = loc_int_I1;
    }

    if(PWM_val_1 < 0)
    {
        PWM_val_1 = 1;
        if(int_RPM_1 < loc_int_RPM1)
             int_RPM_1 = loc_int_RPM1;
        if(int_I1 < loc_int_I1)
             int_I1 = loc_int_I1;
    }
```

```
  // Update PWM values
    PWM0_0_CMPB_R = PWM_val_0;
    PWM0_1_CMPB_R = PWM_val_1;

    TIMER1_ICR_R |= 1<<0; //clear interrupt
}

void UART_Int_Handler(void)
{
    arr[buff_index] = UARTCharGet(UART0_BASE); //Store UART Char in buffer
    if(arr[buff_index]=='\n')
    uart_flag = 1;                             //Flag new input complete
    UART0_ICR_R |= (1<<4)|(1<<6);              //Clear Interrupt Flag
    buff_index++;
}

// If UART receives p, pen is lifted up
// If UART receives P, pen is touched down
// Else try converting the command to XY
// coordinate and store in reference variables
void process_command()
{
    int j = 0;
    float mul = 0.1;
    float xtemp = 0;
    float ytemp = 0;
    if(arr[buff_index-2] == 'p')
    {
        pen_up();
        uart_send_ack(1);
    }
    else if(arr[buff_index-2] == 'P')
    {
        pen_down();
        uart_send_ack(1);
    }
    else
    {
        for (j = (buff_index-2); arr[j] != ','; j--)
        {
            if(arr[j] == '.')
                continue;
            ytemp += (arr[j]-'0')*mul;
            mul = mul*10;
        }
        mul = 0.1;
        j--;
        for (; j >= 0; j--)
        {
            if(arr[j] == '.')
                 continue;
            xtemp += (arr[j]-'0')*mul;
                 mul = mul*10;
        }
        if(xtemp < bed_range && ytemp < bed_range)
        {
            x_ref = xtemp;
            y_ref = ytemp;
            lock_flag = 0;
        }
    }
    buff_index = 0;
    uart_flag = 0;
}
// Software Delay Function
void delayMs(int n)
{
    int i, j;
    for(i =0; i < n; i++)
        for(j =0; j <3180; j++){}/* do nothing for 1 ms */
}
```

VI PYTHON CODE

```python
from pynput.mouse import Listener
import serial.tools.list_ports
import serial
from collections import deque

cmd_que  = deque([]) # Queue to store XY points temporarily
click_count = 0      # On 4 clicks at same pixel, register pixel
clicked     = 0      # Flag to indicate left click status
x_list =    []       # Temporary storage for X pixel value
y_list =    []       # Temporary storage for Y pixel value
range = 250.0        # Range in mm to span
(x_last,y_last) = (0,0)
xmin = 0
xmax = 0
ymin = 0
ymax = 0
ser = []
x_range = 250.0 # X axis movement in mm
y_range = 250.0 # Y axis movement in mm

bound_flag = 0

def between(num, min, max):
    if min < num and num < max:
        return True
    else:
        return False
# Handler to push XY coordinates to the queue if mouse is left clicked in valid range
# Also push pen up and down commands according to mouse click status
def on_move(x, y):
    global bound_flag
    global xmin, ymin, xmax, ymax
    global ser
    global clicked
    global cmd_que
    global range
    if(bound_flag == 1):
        if(between(x,xmin,xmax) and between(y,ymin,ymax)):
            if(clicked == 1):
                x_mm = range*(x-xmin)/(xmax-xmin)
                x_str = "{:.1f}".format(x_mm)
                y_mm = range*(ymax-y)/(ymax-ymin)
                y_str = "{:.1f}".format(y_mm)
                cmd_que.appendleft(bytes(y_str+','+x_str+'\n',encoding="utf"))
                clicked = 2
                cmd_que.appendleft(bytes("P\n", encoding="utf"))

            elif(clicked == 2):
                x_mm = range*(x-xmin)/(xmax-xmin)
                x_str = "{:.1f}".format(x_mm)
                y_mm = range*(ymax-y)/(ymax-ymin)
                y_str = "{:.1f}".format(y_mm)
                cmd_que.appendleft(bytes(y_str+','+x_str+'\n',encoding="utf"))

            elif(clicked == -1):
                cmd_que.appendleft(bytes("p\n", encoding="utf"))
                clicked = 0

# Handle registration of end points and mouse left clicks after registration is done
def on_click(x, y, button, pressed):
    global bound_flag
    global click_count
    global clicked
    global x_list
    global y_list
    global x_last
    global y_last
    global xmin, ymin, xmax, ymax

```

```
    if pressed:
        clicked = 1
        if(bound_flag == 0):

            if(x_last == x and y_last == y):
                click_count = click_count+1
            else:
                click_count = 1
                x_last = x
                y_last = y

            if(click_count > 3):
                x_list.append(x)
                y_list.append(y)
                print(str(x)+", "+str(y)+" Point registered")
                click_count = 0

            if(len(x_list) >= 2):
                xmin = min(x_list)
                ymin = min(y_list)
                xmax = max(x_list)
                ymax = max(y_list)
                bound_flag = 1
                print("Xmin = "+str(xmin))
                print("Ymin = "+str(ymin))
                print("Xmax = "+str(xmax))
                print("Ymax = "+str(ymax))
                print("X range = "+str(xmax-xmin))
                print("Y range = "+str(ymax-ymin))
                x_list = []
                y_list = []
    else:
        clicked = -1
# Dummy function as scroll is not used
def on_scroll(x, y, dx, dy):
    pass


# Input Serial port number and open it @ 115200 baud
i = 1
com_list = []
for item in list(serial.tools.list_ports.comports()):
    print(str(i)+". "+str(item))
    com_list.append(str(item))
    i = i + 1
choice  = int(input("Input the number corresponding to Output COM Port\n"))
ser = serial.Serial(com_list[choice-1].split(" ")[0], timeout=1)
ser.baudrate = 115200

print("Opening "+com_list[choice-1])

# Create mouse listener to capture movement and clicks
listener = Listener(
    on_move=on_move,
    on_click=on_click,
    on_scroll=on_scroll)
listener.start()

print("Proof stuff after listener also executes")

# While loop to send XY/Pen commands to plotter and wait for
# Acknowldegement until timeout. If timed out, resend same command
while(1):
    if(len(cmd_que) != 0):
        cmd = cmd_que.pop()
        ser.write(cmd)
        print(str(cmd))
        str11 = ser.readline()
        while(str11 != b'1\n'):
            ser.write(cmd)
            str11 = ser.readline()
```