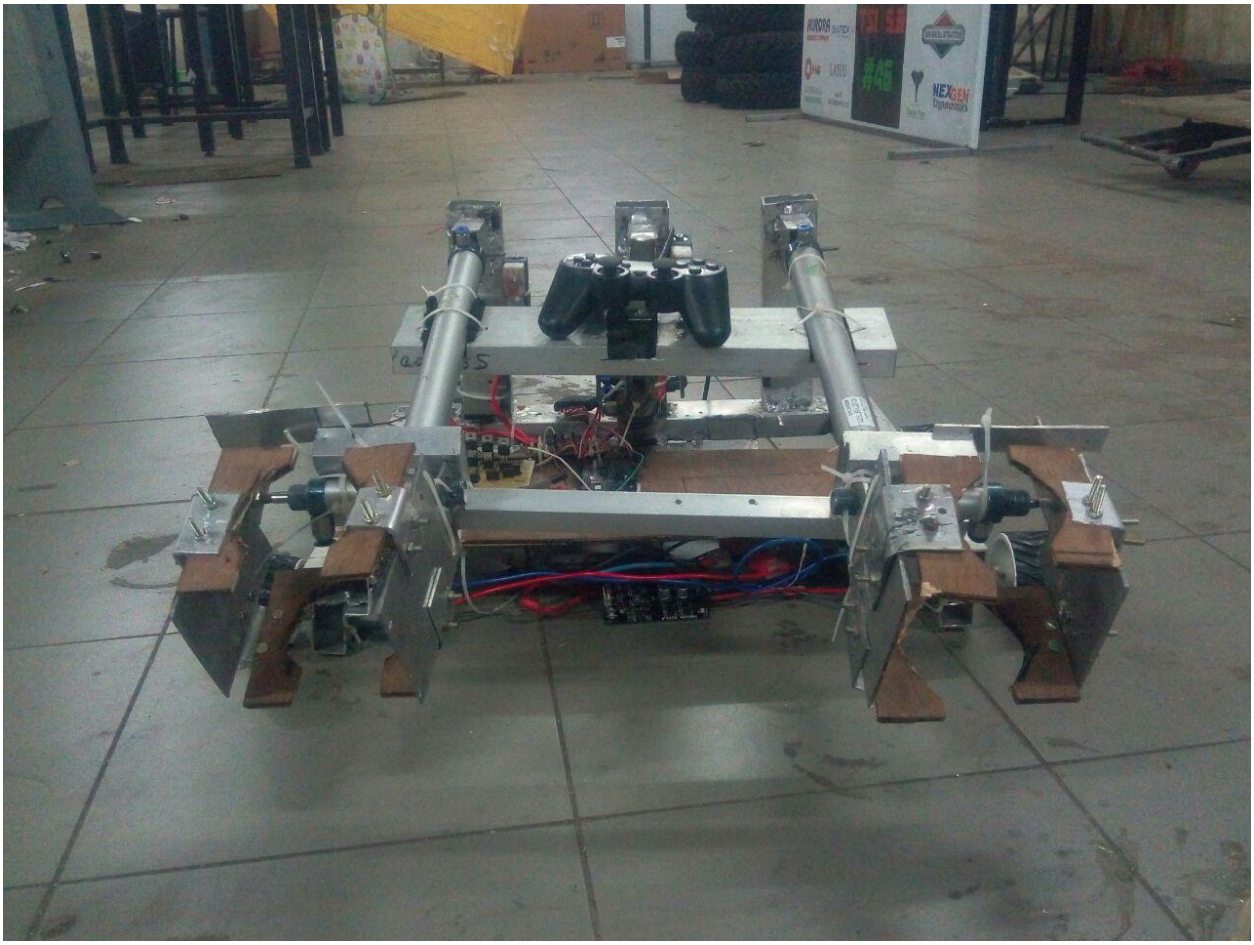
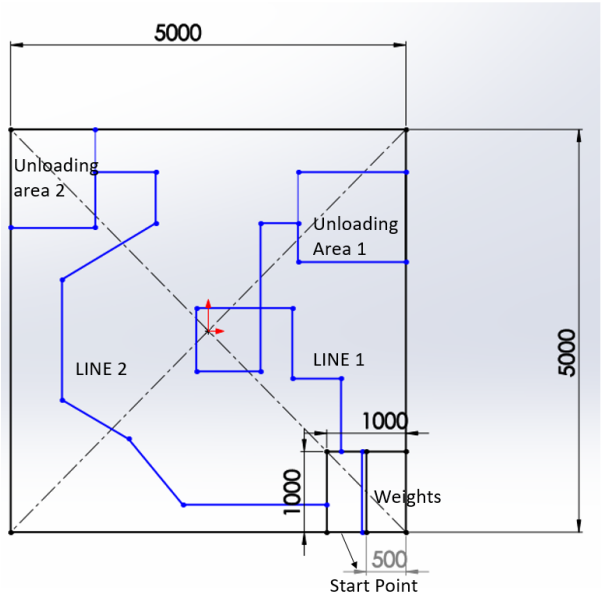
TASK – 2

FINAL REPORT



PROBLEM STATEMENT- TASK 2

1) At the start point the robot should stand still. There should be a button to start the robot. The robot has to pick up given objects(cylindrical in shape)(pepsi bottle small 750 ml) either manually or autonomously.
2) When the weight picked up is 500g, the robot should choose line 1 autonomously and go along the given path and unload the weight at given area at ground level.
3) After unloading the robot should return to the starting point without line following.
4) Then it has to pick up objects again. When the weight is 1kg, it has to autonomously choose line 2 and go along the path and unload the object or objects at a height of 500mm and return to start point.
5) Robot dimension: 500mmX500mmX1000mm
6) Total time for completion is 3 minutes.
7) NOTE: The unloading area 2 is at height 500mm.



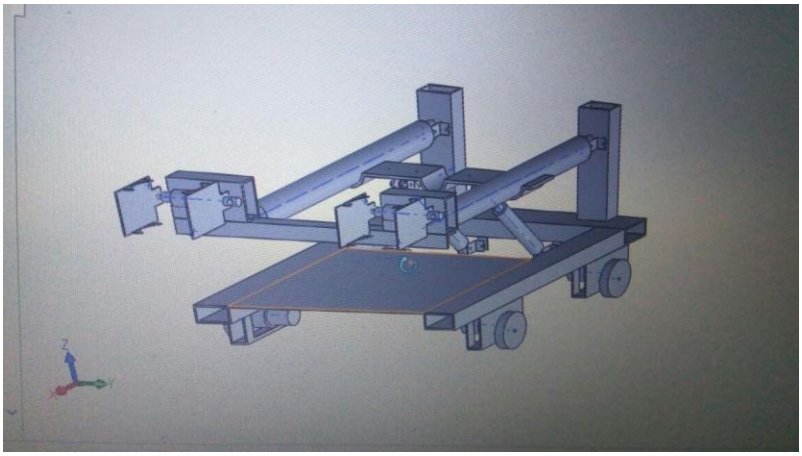
MECHANICAL REPORT:

Base:

The base is manufactured using rectangular aluminium channels that are welded together, using TIG welding. The base dimensions are 40*40 centimeters. It is a 2 wheel drive, having both the rear wheels powered up.

The base is U shaped, with pneumatic support mounted on its back channel.

4 motor mounts have been used in the base.



Gripper:

Two single acting pneumatics have been used as grippers with a wooden claw attached on each end of the gripper, shaped according to the bottle.

The gripper are linearly actuated with the help of pneumatics, having a bigger stroke length. The two pneumatics are mounted on a rectangular aluminium channel that is again mounted on a piston end of a heavy load pneumatic that is attached at the back of the base with the help of welding and press fitting.

Movement in Z direction is provided by the heavy load pneumatic, whereas in the X direction it is provided by the pneumatic having longer stroke length.

The grippers are mounted on the end of pneumatic having longer stroke length.

The gripper are press fitted and fixed with the help of zip tags.

End Actuators:

The end actuators are made U shaped wood that is attached with a metal plate at the either end of single acting pneumatic cylinder.

Base for electrical stuff:

A wooden piece is used as base for electrical stuff. It inserted between the between the bot and is kept fixed and hanging with the help of zip tags.

Challenges faced during fabrication:

1) Wheel alignment was solved.
2) The motor mounts manufactured earlier were not proper. So it was manufactured again for welding.
3) Initially it was planned to use 3 pneumatics, but the 3rd pneumatic was unfit, so it was removed and now only two pneumatics are in use.
4) Lack of components
5) The motor shafts did have holes, so it was drilled into the shafts

ELECTRICAL DEPARTMENT:

Drive:-

We had decided to go for X wheel drive as we needed good mobility due to many turns of the tracks in the arena. For X wheel drive we needed omni- wheels. When the design was made we found out that we couldn't go for omni-wheels as the mount for those wheels was not a suitable option for our design. So we decided to go for standard wheels with 7cm diameter and 4cm thickness with a hole in the shaft to put screws in.

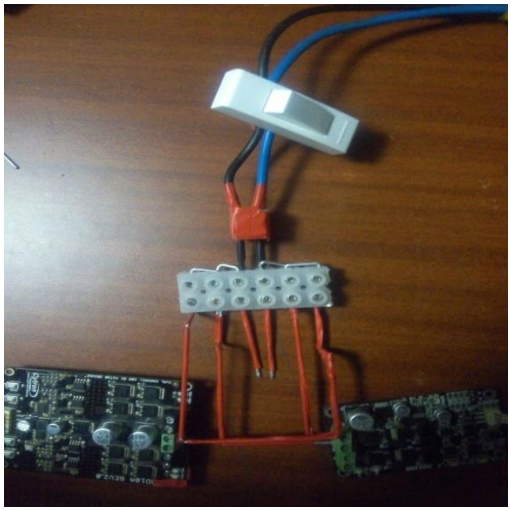
Motor drivers:-

According to the weight of the bot(20kg) and the motor calculations, we had to go for 10kgcm and torque 300rpm motors, but in our inventory there weren't 4 motors with the same specs(ie. Torque and rpm). So we decided to use the motors with the same torque and to control the rpm with pulse width modulation(PWM) of the motor drivers(MD10A).

10kgcm motors required a high current input and a 12V supply to work, so we couldn't go for L298 or L293D as they wouldn't be able to provide the required amount of current and also it had the risk of burning due to the high discharge current of LiPo battery .

Since the new recruits were divided into 2 teams, we had a shortage of one dual channel MD10A motor drivers, hence we had to use 1 dual channel and 2 single channel motor drivers for all the 4 motors.

On the day of finalizing the motor wirings, few events really delayed our work. Since the motor drivers were old, the bases of the screw connectors were rusted and hence they were breaking from the soldered part of the board. So we had to desolder the remaining part of the pins and then solder the screw connectors to the board. For this we had almost lost 4 hours it was really difficult to desolder the pins. At last we decided to directly solder the battery and motors wires to the back of the motor driver. Since there was no tether in the board, we had to put M-seal on the soldered wires and let it dry for 12 hours. So our work was delayed by a whole day.We were using 3 motor drivers so we had to connect all of them in parallel so that each of them get the same power supply. We decided to use an array of 6 terminal blocks(bus bars) and connect the battery in parallel to the motor drivers. We had a shortage of thick wires , so we used thin filler rods to connect the supply and the motor drivers.



MOTOR CALCULATIONS

Thumb rule- **maximum distance in minimum time**

Weight of bot taken = 20kg

Radius of wheel= 4.25cm

Approximating the straight maximum distance in our arena… it came 8m.

Assuming it as to be covered in 8seconds

We get

V=1m/s,

Angular velocity, w= 100/4.25= 23.52 rad/s

So, n=60w/2*pi

This came = **224.68 RPM**

For torque,

F=ma=20*(1-0)/1

F=20 N

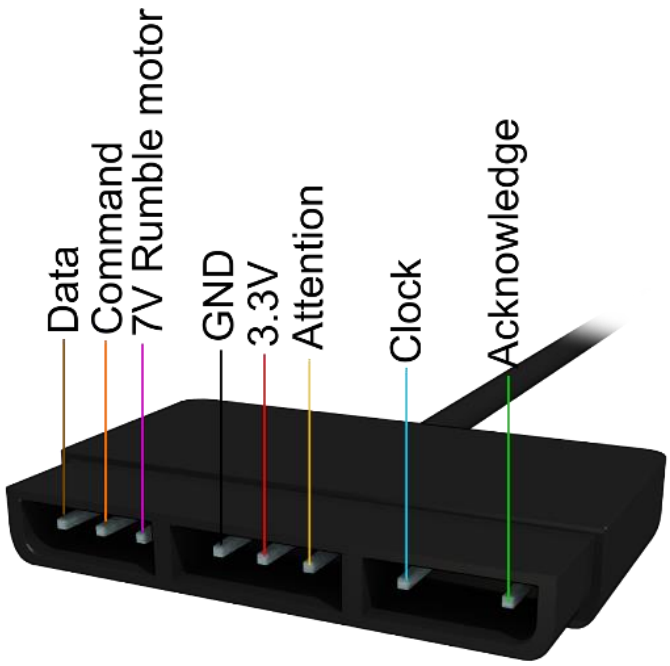
T=F*r

T=20*4.25/100

This comes**, 0.85 N-m** or **8.67 kgcm**

The motors available close to both values were used.

PS2 PIN CONNECTIONS:



Ps2 test code:

```
#include <PS2X_lib.h>
PS2X ps2x;
#define PS2_DAT       13
#define PS2_CMD       11
#define PS2_SEL       10
#define PS2_CLK       12

int green=0; int red=0; int blue=0; int pink=0;
int L1=0; int L2=0; int R1=0; int R2=0;
int UP=0; int DOWN=0; int LEFT=0; int RIGHT=0;
int SELECT=0; int START=0;

void setup() {
  ps2x.config_gamepad(PS2_CLK, PS2_CMD, PS2_SEL, PS2_DAT, false, false);
  Serial.begin(9600);
```

```
}

void loop() {
  ps2x.read_gamepad();

  green=ps2x.ButtonPressed(PSB_GREEN);
  red=ps2x.ButtonPressed(PSB_RED);
  blue=ps2x.ButtonPressed(PSB_BLUE);
  pink=ps2x.ButtonPressed(PSB_PINK);
  L1=ps2x.ButtonPressed(PSB_L1);
  L2=ps2x.ButtonPressed(PSB_L2);
  R1=ps2x.ButtonPressed(PSB_R1);
  R2=ps2x.ButtonPressed(PSB_R2);
  UP=ps2x.ButtonPressed(PSB_PAD_UP);
  DOWN=ps2x.ButtonPressed(PSB_PAD_DOWN);
  LEFT=ps2x.ButtonPressed(PSB_PAD_LEFT);
  RIGHT=ps2x.ButtonPressed(PSB_PAD_RIGHT);
  SELECT=ps2x.ButtonPressed(PSB_SELECT);
  START=ps2x.ButtonPressed(PSB_START);
   if(green==1 && R1==0){
  Serial.println("Triangle just pressed");
  red=0;
  }
   if(red==1){
  Serial.println("Circle just pressed");
  blue=0;
  }
   if(blue==1){
  Serial.println("X just pressed");
```

```
  pink=0;
  }
   if(pink==1)
  Serial.println("Square just pressed");
  if(L1==1 && R2==0){
  Serial.println("L1 just pressed");
  R1=0;
  }
if(L2==1 && LEFT==0){
  Serial.println("L2 just pressed");
  R2=0;
  }
  if(R1==1 && green==1)
  Serial.println("R1 just pressed");
  if(R2==1 && L1==1)
  Serial.println("R2 just pressed");
   if(UP==1 && START==0){
  Serial.println("UP just presses");
  RIGHT==0;
  }
  if(DOWN==1 && RIGHT==0){
  Serial.println("DOWN just presses");
LEFT=0;
  }
   if(LEFT==1 && L2==1)
  Serial.println("LEFT just presses");
  if(RIGHT==1 && DOWN==1)
  Serial.println("RIGHT just presses");
  if(SELECT==1 )
  Serial.println("SELECT just presses");
```

```
  if(START==1 )
  Serial.println("START just presses");
  /*if(ps2x.Analog(PSS_LX)<110)
  {
   }*/
  //Serial.print("Left Stick:");
  int l_axis=(114-ps2x.Analog(PSS_LY));
  int r_axis=(ps2x.Analog(PSS_RX)-114);
  if((l_axis>20)&&(r_axis>-10)&&(r_axis<10))
  Serial.println("GO FRONT");
  else if((l_axis<-20)&&(r_axis>-10)&&(r_axis<10))
  Serial.println("GO BACK");
 /* Serial.print(ps2x.Analog(PSS_LY));
  Serial.print("      Right Stick:");
  Serial.print(ps2x.Analog(PSS_RX));
  Serial.print(",");
  Serial.println(ps2x.Analog(PSS_RY));*/
 else if((r_axis>30)&&(l_axis>-50)&&(l_axis<50))
  Serial.println("GO RIGHT");
  else if((r_axis<-30)&&(l_axis>-50)&&(l_axis<50))
  Serial.println("GO LEFT");
 else if((r_axis>20)&&(l_axis>20))
  Serial.println("TURN RIGHT FORWARD");
  else if((r_axis<-20)&&(l_axis>20))
  Serial.println("TURN LEFT FORWARD");
 }
```

Left and Right Analog:

Apart from the buttons we did the calibration of the 2 analog buttons so that the left analog is used just for moving front and back whereas the right analog is used for moving left and right .We have also made a combination when the left analog is moved forward and right analog is used simultaneously. Therefore we can use whatever combination we want for a specific movement of the bot

Problems Faced:

The ps2 controller gives random values on pressing individual values as in it gives multiple buttons pressed when we press triangle ,square etc .Therefore we need to first initialize all the buttons as 0 and while reading the value of each individual button we need to make the unwanted buttons as 0.

Apart from this we wasted some time to figure out why the ps2 wasn't giving proper outputs and we found that 2 jumpers were damaged .

Since there are limited buttons in the ps2 controller we need to either make combinations or use a flag as condition to control multiple tasks.

Also while using the ps2 controller one needs to press hard the buttons and wait before each action as there is a very minute delay between 2 actions.

Code For Using Combination of Buttons:

```
#include <PS2X_lib.h>

PS2X ps2x;


#define PS2_DAT        13

#define PS2_CMD        11

#define PS2_SEL        10

#define PS2_CLK        12

int pin1=6;

int green=0; int red=0; int blue=0; int pink=0;

int L1=0; int L2=0; int R1=0; int R2=0;

int UP=0; int DOWN=0; int LEFT=0; int RIGHT=0;

int SELECT=0; int START=0;

void setup() {

  ps2x.config_gamepad(PS2_CLK, PS2_CMD, PS2_SEL, PS2_DAT, false, false);

  Serial.begin(9600);

  Serial.println("Start");

  delay(2000);


}

void loop()

{
```

```
 ps2x.read_gamepad();
   green=ps2x.ButtonPressed(PSB_GREEN);
   red=ps2x.ButtonPressed(PSB_RED);
   blue=ps2x.ButtonPressed(PSB_BLUE);
   pink=ps2x.ButtonPressed(PSB_PINK);
   R1=ps2x.ButtonPressed(PSB_R1);
   SELECT=ps2x.ButtonPressed(PSB_SELECT);
   START=ps2x.ButtonPressed(PSB_START);
   UP=ps2x.ButtonPressed(PSB_PAD_UP);
   DOWN=ps2x.ButtonPressed(PSB_PAD_DOWN);
   LEFT=ps2x.ButtonPressed(PSB_PAD_LEFT);
   RIGHT=ps2x.ButtonPressed(PSB_PAD_RIGHT);
   L1=ps2x.ButtonPressed(PSB_L1);
   L2=ps2x.ButtonPressed(PSB_L2);
    R2=ps2x.ButtonPressed(PSB_R2);
if(pink==1)
   pneumatic(1);
   else if(green==1 && R1==0  )
   {
   red=0;
   pneumatic(2);
   }
   else if(red==1)
   {
  pneumatic(3);
   blue=0;
   red=0;
   }
   else if(START==1)
   {
```

```
  pneumatic(4);
  }
  else if(R1==1 && green==1)
  pneumatic(5);
}
//global
int pneumatic(int a)
{
  UP=ps2x.ButtonPressed(PSB_PAD_UP);
  DOWN=ps2x.ButtonPressed(PSB_PAD_DOWN);
  LEFT=ps2x.ButtonPressed(PSB_PAD_LEFT);
  RIGHT=ps2x.ButtonPressed(PSB_PAD_RIGHT);
  L1=ps2x.ButtonPressed(PSB_L1);
  L2=ps2x.ButtonPressed(PSB_L2);
  R2=ps2x.ButtonPressed(PSB_R2);
    Serial.println(a);
 /* Serial.println("pneumatic is activated ");*/
if(a==1)
 {
       if(UP==1 && START==0)
       {
         Serial.println(a);
         //Serial.println(" will extend pneumatic");
         digitalWrite(pin1,HIGH);
       }
 else if(DOWN==1 && RIGHT==0)
       {
         Serial.println(a);
         //Serial.println(" will contract pneumatic");
       digitalWrite(pin1,LOW);
```

```
        }
      else if(L2==1 && LEFT==0)
        {
          Serial.println(a);
          Serial.print(" will leave the bottle pneumatic");
        }
        else if(L1==1 && R2==0)
        {
          Serial.println(a);
          Serial.println(" will grip the bottle pneumatic");
        }
  }
else if(a==2)
 {
  Serial.print("Inside") ;
  Serial.println(a);
        if(L1==1 && R2==0)
        {
          Serial.println(a);
          Serial.print("HEllo") ;
          Serial.println(" will grip the bottle pneumatic");
          red=0;
        }
        else if(L2==1 && LEFT==0)
        {
          Serial.println(a);
          Serial.print("Hello") ;
          Serial.print(" will leave the bottle pneumatic");
          red=0;
        }
```

```
        else if(UP==1 && START==0)
        {
          Serial.println(a);
          Serial.println(" will extend pneumatic");
          red=0;
        }
        else if(DOWN==1 && RIGHT==0)
        {
          Serial.println(a);
          Serial.println(" will contract pneumatic");
          red=0;
        }
    }

else if(a==3)
 {
        if(UP==1 && START==0)
        {
          Serial.println(a);
          Serial.println(" will extend pneumatic");
        }
else if(DOWN==1 && RIGHT==0)
        {
          Serial.println(a);
          Serial.println(" will contract pneumatic");
        }
        else if(L1==1 && R2==0)
        {
          Serial.println(a);
          Serial.println(" will grip the bottle pneumatic");
```

```
        }
        else if(L2==1 && LEFT==0)
        {
          Serial.println(a);
          Serial.print(" will leave the bottle pneumatic");
        }

 }
else if(a==4)
   Serial.println("Extend base pneumatic");
else if(a==5)
   Serial.println("Contract base pneumatic");
 }
```

EXPLANATION:

In this we tried using the pneumatics using combination of buttons like when **(square + up button)** on left side of joystick would be pressed the arms of the bot would extend when square and L2 would be pressed the gripper would extend and when L1 will be pressed the gripper will grip the bottle. After that when we press the **(square + down button)** the arms would come back to original position.

Similarly we set the combinations for triangle and circle.

Problems while using combinations of buttons:

Both the square and circle combinations are working properly but while using the triangle it gives absurd values which made it quite difficult to fix the issue.

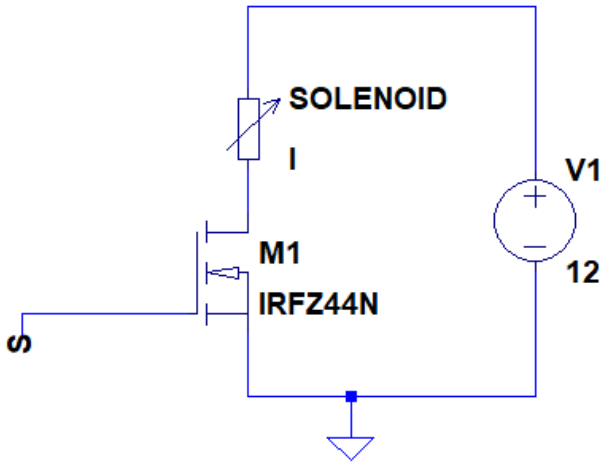
Therefore we decided to use flags instead to combinations .Using flags gives an added advantage of using the same button for both actions(extending and contraction of pneumatics).

Pneumatics:

Circuit Diagram:

Initially we made a circuit where we connected the load in parallel to the 12V supply in which we had to give 0 for switching on the mosfet and 1 for switching off, but due to the complexity of the circuit and better understanding we connected the load in series with the supply.

We also thought of using 14 mosfets and we designed the board according to that but due to lack of time and unavailability of components we switched to rather 4 mosfets .



Hsere my S is the Arduino which uses a digital pin to give supply to the mosfets and in our task we have used 4 mosfets for 4 single acting pneumatics 1 for both arms working together, 2 for gripping and 1 for base.

Final code for Pneumatic based on flag:

```
#include <PS2X_lib.h>

PS2X ps2x;

#define PS2_DAT       13

#define PS2_CMD       11

#define PS2_SEL       10

#define PS2_CLK       12

int pnuematic1=7;

int pnuematic2=6;

int pnuematic3=5;

int pnuematic4=4;

int pnuematic5=3;

int green=0; int red=0; int blue=0; int pink=0;

int L1=0; int L2=0; int R1=0; int R2=0;

int UP=0; int DOWN=0; int LEFT=0; int RIGHT=0;

int SELECT=0; int START=0;

int c=0,c1=0,c2=0,c3=0,c4=0,c5=0,c6=0,c7=0; //Counters
```

```
void setup() {
  Serial.begin(9600);
  ps2x.config_gamepad(PS2_CLK, PS2_CMD, PS2_SEL, PS2_DAT, false, false);
  delay(5000);
  pinMode(pnuematic1,OUTPUT);
  pinMode(pnuematic2,OUTPUT);
  pinMode(pnuematic3,OUTPUT);
  pinMode(pnuematic4,OUTPUT);
  pinMode(pnuematic5,OUTPUT);
  /*digitalWrite(pnuematic1,LOW);
  digitalWrite(pnuematic2,LOW);
  digitalWrite(pnuematic3,LOW);
  digitalWrite(pnuematic4,LOW);
  digitalWrite(pnuematic5,LOW);*/
}
void loop() {
   ps2x.read_gamepad();

   green=ps2x.ButtonPressed(PSB_GREEN);
   red=ps2x.ButtonPressed(PSB_RED);
   blue=ps2x.ButtonPressed(PSB_BLUE);
   pink=ps2x.ButtonPressed(PSB_PINK);
   L1=ps2x.ButtonPressed(PSB_L1);
   L2=ps2x.ButtonPressed(PSB_L2);
   R1=ps2x.ButtonPressed(PSB_R1);
   R2=ps2x.ButtonPressed(PSB_R2);
   UP=ps2x.ButtonPressed(PSB_PAD_UP);
   DOWN=ps2x.ButtonPressed(PSB_PAD_DOWN);
   LEFT=ps2x.ButtonPressed(PSB_PAD_LEFT);
```

```
RIGHT=ps2x.ButtonPressed(PSB_PAD_RIGHT);
SELECT=ps2x.ButtonPressed(PSB_SELECT);
START=ps2x.ButtonPressed(PSB_START);

//green button for pnuematic 1
if(green==1 && R1==0 && c==0)
{
  Serial.println("Green Activated");
  //digitalWrite(pnuematic1,LOW);
  digitalWrite(pnuematic1,HIGH);
  c=1;
  red=0;
 }
else if(green==1 && R1==0 && c==1)
{
 Serial.println("Green DeActivated");
 c=0;
 red=0;
digitalWrite(pnuematic1,LOW);
}
//red button for pnuematic2
if(red==1 && c3==0)
{
 c3=1;
 blue=0;
 digitalWrite(pnuematic2,HIGH);
}
else if(red==1 & c3==1)
{
 c3=0;
```

```
    blue=0;
   digitalWrite(pnuematic2,LOW);
  }
  //blue button for pnuematic3
  if(blue==1 && c1==0)
  {
   pink=0;
   c1=1;
   digitalWrite(pnuematic3,HIGH);
  }
  else if(blue==1 && c1==1)
  {
   pink=0;
   c1=0;
   digitalWrite(pnuematic3,LOW);
  }
  //pink button for pnuematic4
  if(pink==1 && c2==0)
  {
   c2=1;
   digitalWrite(pnuematic4,HIGH);
  }
  else if(pink==1 && c2==1)
  {
    c2=0;
    digitalWrite(pnuematic4,LOW);
  }
  //L1 button for pnuematic5
if(L1==1 && R2==0 && c4==0){
  R1=0;
```

```
  c4=1;
  digitalWrite(pnuematic5,HIGH);
  }
 else if(L1==1 & R2==0 && c4==1){
  R1=0;
  c4=0;
  digitalWrite(pnuematic5,LOW);
 }
}
```

LINE FOLLOWING:

Codes For Line Following Using Camera:

```
#Basic logic used for line following using camera
#Codes:

#importing required libraries
import cv2
import numpy as np
import RPi.GPIO as g
g.setmode(g.BOARD)#define the numbering system to be used for the pin(This way is the simple way)

#set up pins for giving apropriate outputs to move the bot

#right side's motor
gpio.setup(31, gpio.OUT)#motor1 pwm
gpio.setup(5, gpio.OUT)#motor1 dir

#left side's motor
gpio.setup(33, gpio.OUT)#motor2pwm
gpio.setup(11, gpio.OUT)#motor2dir
```

```
#make PWM objects to use PWM methods(object-oriented programming: objects are variables of class type used to access the functions of the class)
motor1 = gpio.PWM(31, 100)#for right-side's motor
motor2 = gpio.PWM(33, 100)#for left-side's motor

#initiate the objects with duty-cycle of 100
motor1.start(100)
motor2.start(100)
#for direction pins; last I checked; if 0 is given as output, rotation is counter-clockwise and if 1 is given as output, rotation is clockwise

#capture video from camera (index : -1=>any random cam is selected; 0 => system's cam is selected; 1,2,3,...=>specify the external cams connected)
cap=cv2.VideoCapture(1)
try:
while True:
_,frame=cap.read()#keep taking the video frome by frame

blur=cv2.GaussianBlur(frame,(15,15),0) # blur is applied to remove noice from the frames

gray=cv2.cvtColor(blur,cv2.COLOR_BGR2GRAY) # convert the blured image to grayscale to simplify the image

#Thresholding the feed so that all the pixel values are either 100 or 0. Simplifies the image further(Extreamly, infact)
ret,otsu=cv2.threshold(gray,0,100,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

#finding the contours in the thresholded feed
contours,_=cv2.findContours(otsu.copy(),1,cv2.CHAIN_APPROX_NONE)

if len(contours)>0:#if find contours

#taking maximum region contour which will be our line to be followed
c=max(contours,key=cv2.contourArea)

M=cv2.moments(c) #Taking moments of the maximum contour in the feed

#How moments work:
```

```
#The moments actually are the measure of spatial distribution of points
#if points represent a value; zeroth moment is the total; 1 st moment devided by zeroth moment(i.e. total) gives the centre for the data;

#center coordinates
cx=int(M['m10']/M['m00'])#centre co-ordinates for x
cy=int(M['m01']/M['m00'])#centre co-ordinates for y

#Keep the centre of contours at centre of the screen
if cx>155: #value for this part (let) a= centre of the window in x + 10
#Turn to the right if centre of the contour gets out of the range
#Change duty cycle of motors to control the speed or rotation
motor2.ChangeDutyCycle(50)
motor1.ChangeDutyCycle(50)
#to turn the bot to right; motor1 moves in clockwise direction and motor2 in counter clock direction
gpio.output(5, 1)
gpio.output(11, 0)

elif cx<145:#value for this part (let) b= centre of window in x - 10
#Turn to left if contour gets out of range
motor2.ChangeDutyCycle(50)
motor1.ChangeDutyCycle(50)
#to turn the bot to left; motor2 moves in clockwise direction and motor1 in counter clock direction
gpio.output(5, 0)
gpio.output(11, 1)


elif 140<cx<160: #Here the values are a>cx>b
#Keep moving straight if the centre is in range
motor2.ChangeDutyCycle(50)
motor1.ChangeDutyCycle(50)
#to turn the bot to right; motor1 and motor2 move in counter clockwise direction
gpio.output(5, 0)
gpio.output(11, 0)
```

```
if cv2.waitKey(1) & 0xFF == ord('q'):#stop capturing if 'q' is pressed
break
except KeyboardInterrupt :
cap.release()#release the camera
g.cleanup()#clear all the outputs given to the board pins
```

BACKTRACKING:

Rotary encoder are basically used for calculate the angle of shaft or number of turns that a well take. Here in our task we used it for calculated the number of turns that a wheel taken. The count will increase for each turn and when turn will come then value of counter is given to stack. And then set the value of counter again equal to zero. The procedure is repeated until over task is finished. As in our stack the value which is going first will come out first, therefore it can be used for retracing the path of robot without line following according to problem statement.

The arduino code:

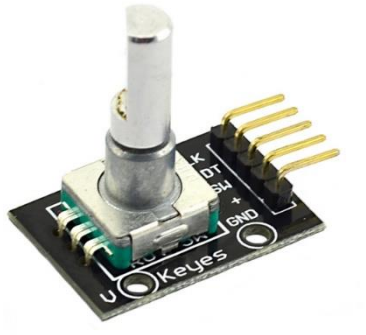
```
#define outputA 6
 #define outputB 7
 int counter = 0;
 int aState;
 int aLastState;
 void setup() {
   pinMode (outputA,INPUT);
   pinMode (outputB,INPUT);
   pinMode (8,OUTPUT);


   Serial.begin (9600);
   aLastState = digitalRead(outputA);
 }
 void loop() {
   counter=digitalRead(8);
   aState = digitalRead(outputA);
   if (aState != aLastState){
     if (digitalRead(outputB) != aState) {
       counter ++;
     }
     Serial.print("Position: ");
     Serial.println(counter);
   }
   aLastState = aState;
 }
```

In above code as we used incremental rotary encoder it has two output say A and B ,if we move rotary in clockwise direction then it counter of A will increase and when anticlockwise the B will increase or the value of counter decrease here to minimize the skidding we remove the code for decrement of counter.

Position of rotary encoder:

The rotary encoder will placed with the shaft of wheel. As wheel moves the value of counter moves and help in retracing the path.



WEIGHT MEASUREMENT USING OPEN CV:

MODULE1: Weight-sensing using OpenCv

Statement:
--> Job is to approximate the weight of water contained in the pepsi-bottle to decide on path of bottle

3 approaches were involved in obtaining the required results

1)Contours searching

2)HSV color detection

3)HAAR-Cascading

4)Background subtraction

CONTOUR APPROACH

A built-in algorithm to detect shapes cv2.findContours() and draw boundaries over it

On trying the only contour approach, we see problems arising like

i)Unfiltered noise # can be rectified to an extent

ii)Ugly contours - not having sufficient knowledge over the mathematical aspect of the contour approximation method made it
difficult to rectify the issue

The bottle did get deteced in face of ideal conditions and extremely heavy filtering function like PYRMEANSHIFTFILTERING()

2)HSV approach

Focused on detecting only the known colors of the colored and water

i)Not feasible independently as background objects with similar rgb values show up as incredible noise

ii) Extremely susceptible to lighting conditions

3) HAAR-Cascading

..Involves the Viola-Jones feature detection algorithm
...The computer is taught to recognize the bottle by feeding it up with 1000s of positive images of the bottle against
approimately half the no. of negative images

....It recognizes multitudes of unique features in the feeded images and based on those detects the requied objects

Failures concieved
i) Even after successfully creating a haar-cascade of 20 stages, due to small nature of the trained size of image, it was
insufficient to detect bottle of size

ii) Process is slow and not fesible for low-juice machines as time exponentially increases t train the computer in incresing size and complexity of the object

4) Background subtraction

Works on the principle of two images being taken,one with an empty bottle and the other with the water-filled one
The two images then being operated on as bitwise exor, masking off the equal bits and keeping the different bits ie the water

This approach falters in two ways

i) We do not take into account the constantly changing light condition that arent detected by the human eye but the camera thus adding
un filterable noise

ii)Replacing empty bottle with the filled one with exactly fitting bits is next to impossible and adds the etra bit of noise

WORKING METHOD:

The approach of HSV and contour was combined to finally accurately detect the weight of water
Complete ALGORITHM:

1)Input image stream

2)Convert image from RGB to HSV

3)Create a mask for the red-colored Water based on rgb-->hsv conversions say mask1

4)Repeat step 3 for blue colored cap say mask2

5)Apply mask 1 to create an image stream showing only water say water_hsv

6)Apply mask 2 to create an image stream showing only cap say cap_hsv

7)Convert water_hsv and cap_hsv into GRAY_SCALE

8)Apply Otsu binarization on both with appropriate tested values for accurate resulting shapes say water_otsu, cap_otsu

9)Apply contours detection to both streams and store as contour_cap , contour_water

10)Find max AREA contour among each contour arrays say , c1 and c2 //these will be our objects

11)Bound the contours in rectangles and determine its coordinates

12)These will give height of bottle=(lower y_coord of bottle-upper y_coord of cap) //heights in pixels

HEIGHT OF WATER

13)Ratio of height to pixel for bottle an dwater is same
thus water_actual_height is found

14)Bottle is mapped with known density of water i.e height vs weight thus, weight obtained from height

Initially instead of contours we tried linear searching of blue and red pixels to find heights but increasing time-complexity
and susceptible to noise

bottle10.xml

bottle20.xml