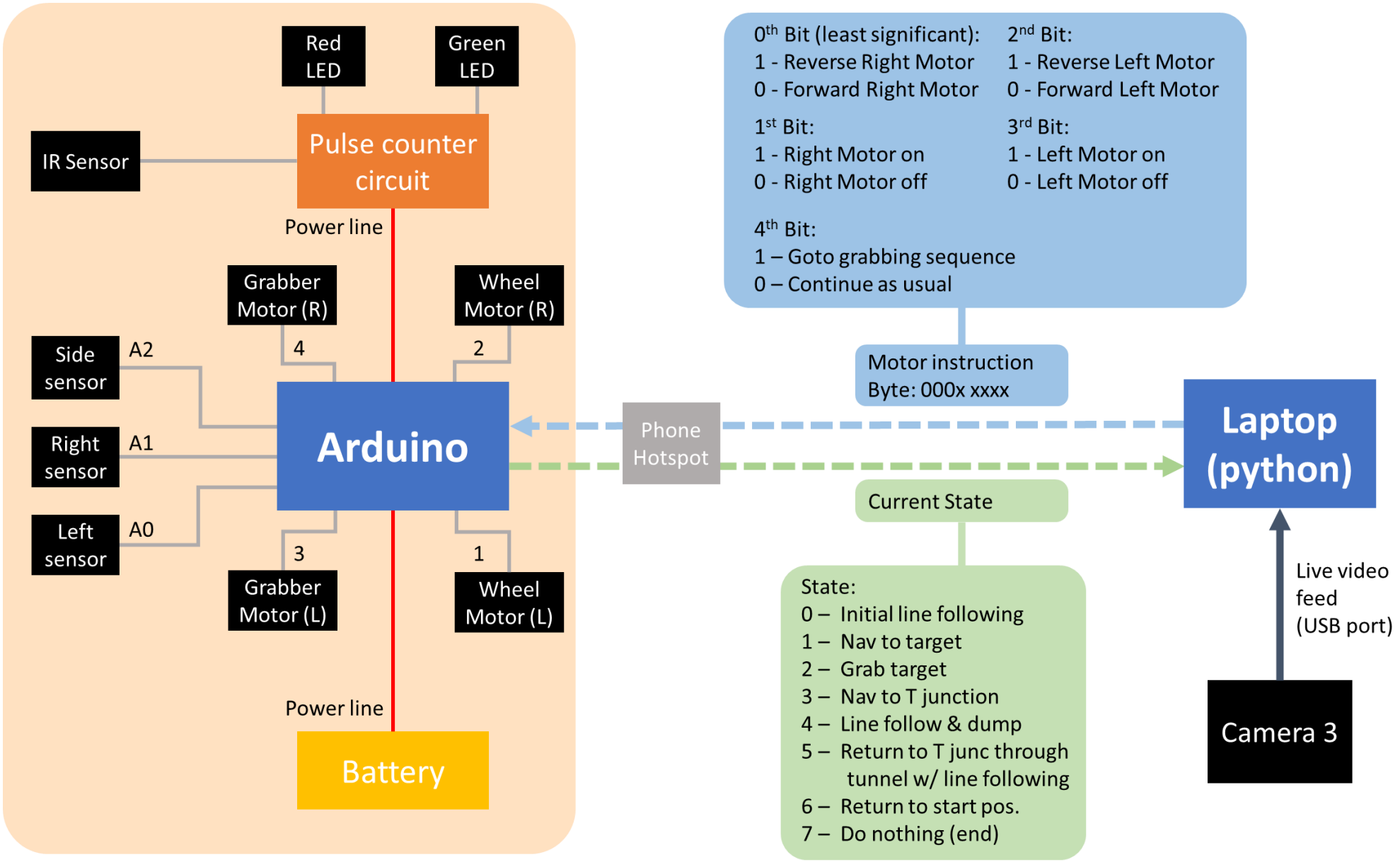
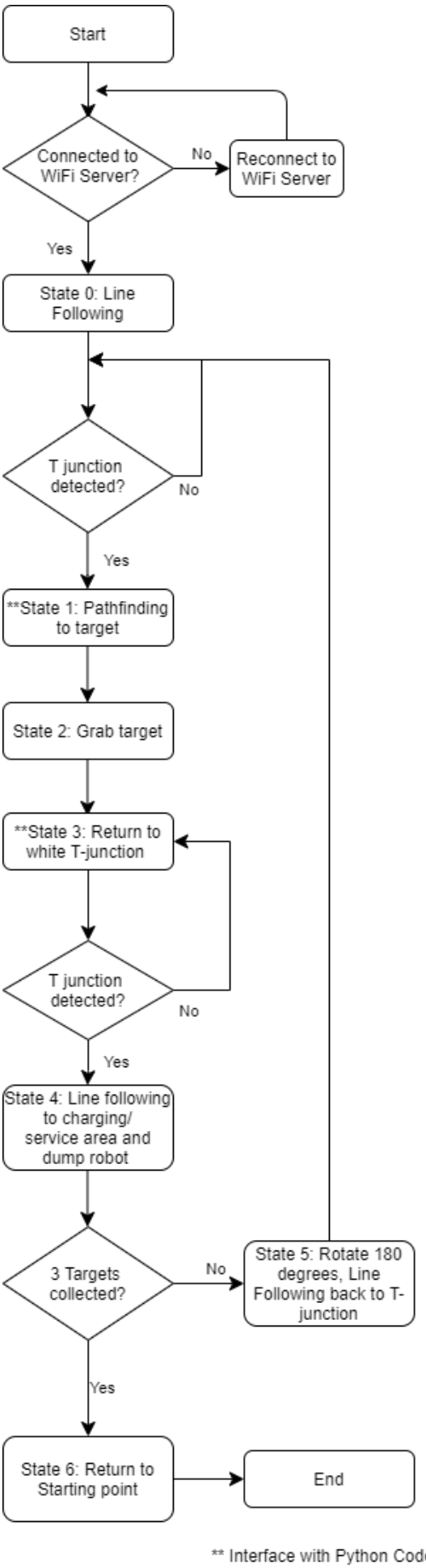
System and Comms Diagram

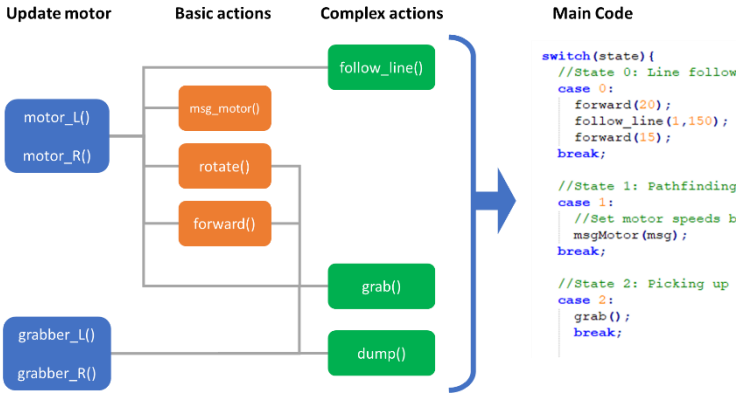


Arduino Flowchart

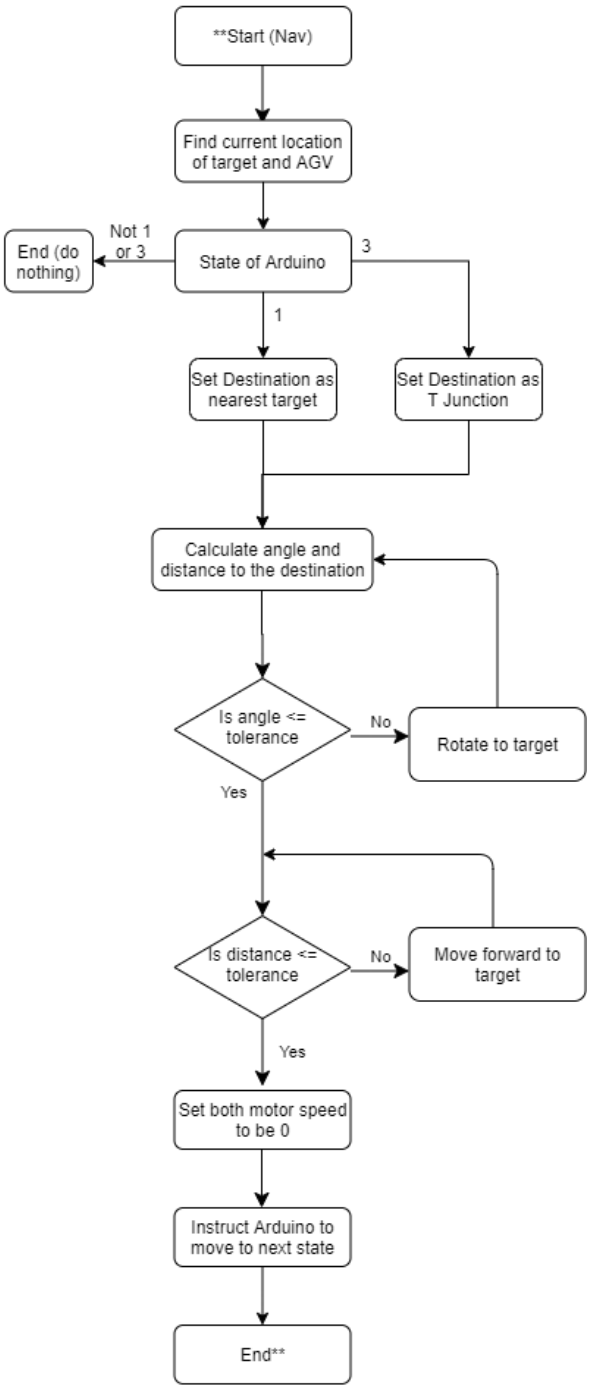


** Interface with Python Code

Code Structure



Python (Nav) Flowchart



Code for Arduino

```
#include <SPI.h>
#include <WiFiNINA.h>
#include <Wire.h>
#include <Adafruit_MotorShield.h>

//Motor init
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *motorL = AFMS.getMotor(1);     //Left wheel motor
Adafruit_DCMotor *motorR = AFMS.getMotor(2);     //Right wheel motor
Adafruit_DCMotor *motorGrL = AFMS.getMotor(3);   //Left grabber motor
Adafruit_DCMotor *motorGrR = AFMS.getMotor(4);   //Right wheel motor
#define LM1 3
#define RM1 4


//Variables
//----------------------------

//State of the robot i.e. current objective
int state = 0;             // 0 = line following, 1 = pathfinding ,2 = ....
int targetsCollected = 0;  //Number of targets currently collected

//Line following
bool sensor_l;          // Line following sensor readings
bool sensor_r;          // 1 = white, 0 = black
bool sensor_s;
int sensor_s_timer=0;   //Ignore side sensor for first x cycles
bool L_faster_LF;       // 1 = faster, 0 = slower
bool R_faster_LF;
double tol=700;         //Tolerance for left and right sensor
double tol_s=850;       //Tolerance for side sensor

//Wifi
char ssid[] = "OnePlus 7 Pro";                          //SSID
char pass[] = "www.youtube.com/watch?v=dQw4w9WgXcQ";    //password
uint8_t msg=0;                  //Variable to store message
uint8_t buf;                    //Byte in buffer
size_t siz = 1;                 //Size of buffer
int status = WL_IDLE_STATUS;    //Wifi status
bool alreadyConnected;

WiFiServer server(23);  //Start server

//Motor variables
int speed_L_current=0;   //Current speed of the motor (L)
int speed_R_current=0;   //Current speed of the motor (R)
int speed_L;       //Intended speed of the motor (L)
int speed_R;       //Intended speed of the motor (R)

uint8_t v=128;     //Default speed
uint8_t v_LF=200;  //Line following speed

//Motor parameters
double ang2t = 24;      //time (ms) taken to rotate one degree for AGV
double dis2t = 100;
uint8_t v_m = 128;      //Speed for grabber motor

//Gripper calibration
double ang2t_ml = 10.5; //time (ms) taken to rotate one degree for gripper motor (L)
double ang2t_mr = 35;   //time (ms) taken to rotate one degree for gripper motor (R)



//Functions
//----------------------------
```

```
//Update motor speed
void motor_L(int speed){
  if (speed != speed_L_current){
    motorL->run( (speed>=0) ? FORWARD : BACKWARD );
    motorL->setSpeed(abs(speed));
    speed_L_current = speed;
  }
}

void motor_R(int speed){
  if (speed != speed_R_current){
    motorR->run( (speed>=0) ? FORWARD : BACKWARD );
    motorR->setSpeed(abs(speed));
    speed_R_current = speed;
  }
}

//Rotation and move forward
void rotate(double angle){
  //Angle +ve clockwise
  motor_L((angle>=0) ? v : -v);
  motor_R((angle>=0) ? -v : v);
  delay(abs(angle)*ang2t);
  motor_L(0);
  motor_R(0);
}

void forward(double dist){
  //dist +ve forward
  motor_L((dist>=0) ? v : -v);
  motor_R((dist>=0) ? v : -v);
  delay(abs(dist)*dis2t);
  motor_L(0);
  motor_R(0);
}


//Line Follower
void follow_line(bool keepRight, int sensor_s_timer){
  // keepRight:  1 -> AGV follows right side of line , 0-> AGV follows left side of
line
  // sensor_s_timer = min number of cycles for line following

  while (1){
    //Sensor readings
    sensor_l = (analogRead(A0)>tol) ? 1 : 0;
    sensor_r = (analogRead(A1)>tol) ? 1 : 0;
    sensor_s = (analogRead(A2)>tol_s) ? 1 : 0;

    //Determine motor speeds via boolean logic
    L_faster_LF = keepRight ? (sensor_l || sensor_r) : (!sensor_l);
    R_faster_LF = keepRight ? (!sensor_r) : (sensor_l || sensor_r);

    //Exit condition
    if (sensor_s && sensor_s_timer<=0){
      motor_L(0);
      motor_R(0);
      return;
    }

    //Update speeds
    motor_L(L_faster_LF ? v_LF : 0);
    motor_R(R_faster_LF ? v_LF : 0);

    //Delay and move to next loop
    sensor_s_timer --;
```

```
    delay(50);
  }
}

//Update motor speeds from bytes sent from python
void msgMotor(uint8_t msg){
  //Left motor: 0th bit = move/not move, 1st bit = reverse
  motor_L( (bitRead(msg,3) ? v : 0) * (bitRead(msg,2) ? -1 : 1) );

  //Right motor: 2nd bit = move/not move, 3rd bit = reverse
  motor_R( (bitRead(msg,1) ? v : 0) * (bitRead(msg,0) ? -1 : 1) );
}

//Move grabber arm
void grabber_R(int angle){
  motorGrR->run( (angle>=0) ? FORWARD : BACKWARD );
  motorGrR->setSpeed(v_m);
  delay(abs(angle)*ang2t_mr);
  motorGrR->setSpeed(0);
}

void grabber_L(int angle){
  motorGrL->run( (angle>=0) ? BACKWARD : FORWARD );
  motorGrL->setSpeed(v_m);
  delay(abs(angle)*ang2t_ml);
  motorGrL->setSpeed(0);
}

//Grab target sequence
void grab(){
  forward(-15);
  grabber_R(110);
  grabber_L(150);
  forward(22);
  grabber_R(-95);
  grabber_R(7);
  grabber_L(-150);
  delay(1000);
  grabber_R(-35);
}

//Release target sequence
void dump(){
  grabber_R(90);
  forward(-20);
  grabber_R(-110);
}

//Actual code in robot
//----------------------------
void setup(){
  //Turn on motorshield and serial
  AFMS.begin();
  Serial.begin(9600);
  server.begin();

  //Sensor inputs
  pinMode(A0, INPUT); //left sensor as input
  pinMode(A1, INPUT); //right sensor as input
  pinMode(A2, INPUT); //Side sensor as input
}

void loop(){

  //Wifi stuff
  if (status != WiFi.status()) {
    status = WiFi.status();
```

```
}

//Keep tring to connect if not connected
if (status != WL_CONNECTED){
  //Stop robot if disconnected
  motor_L(0);
  motor_R(0);
  while (status != WL_CONNECTED) {
    status = WiFi.begin(ssid, pass);
    // wait 5 seconds for each attempt
    delay(5000);
  }
  alreadyConnected = false;
}

//Read connection if there is client
WiFiClient client = server.available();
if (client) {
  if (!alreadyConnected) {
    // clead out the input buffer:
    client.flush();
    alreadyConnected = true;
  }

  if (client.available() > 0) {
    // read the bytes incoming from the client:
    client.read(&buf , siz);
    msg = buf+1-1;
  }
}


//Switch statement based on current state of the robot

switch(state){
  //State 0: First Line following
  case 0:
    forward(20);
    follow_line(1,150);
    forward(15);
    state = 1;
  break;


  //State 1: Pathfinding to target
  case 1:
    //Update motor speeds based on inputs from python
    msgMotor(msg);

    //Exit condition (4th bit is signal from python that it is in front of the robot)
    if (bitRead(msg,4)){
      motor_L(0);
      motor_R(0);
      state = 2;
    }
  break;


  //State 2: Picking up target
  case 2:
    grab();
    state=3;
    targetsCollected++;
    break;

  //State 3: return to T junction
  case 3:
```

```
      //Sensor readings
      sensor_l = (analogRead(A0)>tol) ? 1 : 0;
      sensor_r = (analogRead(A1)>tol) ? 1 : 0;

      //Update motor speeds based on inputs from python
      msgMotor(msg);

      //Exit condition (if both sensors read white it thinks it reached the T junction)
      if (sensor_r && sensor_l){
        state = 4;
        motor_L(0);
        motor_R(0);
      }
      break;

    //State 4: Line follow to charging/service area
    case 4:
      forward(10);
      rotate(10);
      follow_line(1,0);
      //Dump after line follow
      dump();
      forward(10);

      //If all target collected, go to state 6, else go to state 5
      if (targetsCollected >= 3){
        state = 6;
      } else {
        state = 5;
      }
      break;

    //State 5: Turn around and line follow back to T junction
    case 5:
      rotate(-200);
      follow_line(0,150);
      forward(15);
      state = 1;
      msg=0;
      break;

    //State 6: End - Return to starting white box
    case 6:
      rotate(-150);
      forward(45);
      follow_line(1,0);
      forward(30);
      state = 7;
      break;

    //State 7: end
    case 7:
        motor_L(0);
        motor_R(0);
        //do nothing
        break;

  }
  //Send current state to python
  server.write(state);
}
```

Code for Python

```
import numpy as np
import cv2 as cv
import telnetlib

#------------Initialisation-----------#

#Import video
cap = cv.VideoCapture(0)
width = int(cap.get(3))
height =int(cap.get(4))
print("Vid dimentions: ",width,"x",height)

#Connect to arduino via telnetlib
ip = "192.168.43.224"
port = 23
connection = False

#Start trigger
StartTrigger = input("Enter any key")

#------------Parameters-----------#

#Cropping parameters (pixels)
crop = [19,320]               #[Left,Right]
tunnel = [163,288,288,384]    #[Left,top,right,bottom]

#Display Box params, w= width, h=height, dof = offset from center
agv = {'w':65,'h':100,'dof':10}
grp = {'w':90,'h':40,'dof':70}

#Thresholding for rectangle detection for AGV
#ca_th : min contour area compared to bounding rectangle threshold
markerH = {'min_area':450,'max_area':800,'min_ratio':2,'max_ratio':4.5,'ca_th':0.5}
markerV = {'min_area':200,'max_area':450,'min_ratio':2,'max_ratio':4.5,'ca_th':0.5}

#Navigation parameters
agv_COR = 1.95         #Placement of the center of rotation of the AGV
min_dist_target = 91   #Stopping distance (pixels) when navigating to target
tol_dist_point = 3     #Stopping distance (pixels) when navigating to a point
ang2t = 7              #Time taken (frames) for rotation of one radian
tol_angle = 0.1        #Tolerance for angle rotation (radians)
T_coords = [250,225]   #Coordinates for T junction (pixels)

#Init variables
agv_coords = [0,440]   #Default position for AGV
rot_angle = 0          #Default rotation angle
motor = [0,0,0,0]      #Bits:[Left motor On, Left motor reverse, Right motor on, Right
motor reverse]

#Keep track of current action and target
action = {'mode':'none','timer':0,'dir':1}  #Mode:none,fwd,rot,stop, dir: 0:fwd, 1:rev
nav = {'type':'t','target' : [70,250]}      #Target (t), Waypoint(w), Endpoint(p)

targets = []           #Target list
state = -1             #Default state

#------------Functions-----------#

#Returns angle between two vectors
def angle(v1,v2):
    return np.arctan2(np.cross(v2,v1),np.dot(v2,v1))

#Rounds number to nearest multiple
def nearestmultiple(x,base):
    return base*round(x/base)
```

```
#Returns black white image which is a linear sum combination of RGB channels (stored in
a)
def linrgb(img,a):
    if(len(a))==3:
        a.append(0)
    img = np.asarray(img,dtype='int16')
    out = a[0]*img[:,:,0]+a[1]*img[:,:,1]+a[2]*img[:,:,2]+a[3]
    out = np.clip(out,a_min=0,a_max=255)
    return np.asarray(out, dtype='uint8' )

#Check is rectangle is correct for AGV tracking
def valid_rect(w,h,min_area,max_area,min_ratio,max_ratio,carea,ca_th):
    w,h = int(w),int(h)
    area = w*h
    if area>0 and area >= min_area and area <= max_area and carea>area*ca_th:
        ratio = max(w/h,h/w)
        if ratio>=min_ratio and ratio<=max_ratio :
            return True
    return False

#Drawing visuals on laptop
def draw_visuals(output,agv_coords,rot_angle):
    #Box for agv
    rect_agv = (

(agv_coords[0]+agv['dof']*np.cos(rot_angle),agv_coords[1]+agv['dof']*np.sin(rot_angle))
,
        (agv['h'],agv['w']),rot_angle * 180/np.pi)
    box_agv = np.int0(cv.boxPoints(rect_agv))


    #Box for gripper area
    rect_grp = (

(agv_coords[0]+grp['dof']*np.cos(rot_angle),agv_coords[1]+grp['dof']*np.sin(rot_angle))
,
        (grp['h'],grp['w']),rot_angle * 180/np.pi)
    box_grp = np.int0(cv.boxPoints(rect_grp))

    #Line
    line_length = 200
    centre_point = (int(agv_coords[0]),int(agv_coords[1]))
    line_point =
(int(agv_coords[0]+line_length*np.cos(rot_angle)),int(agv_coords[1]+line_length*np.sin(
rot_angle)))

    #Draw boxes
    cv.circle(output,centre_point,5,(128,128,0),10)
    cv.circle(output,tuple(T_coords),5,(128,128,128),10)
    cv.drawContours(output,[box_agv],0,(255,255,0),3)
    cv.drawContours(output,[box_grp],0,(255,255,0),3)
    cv.line(output,centre_point,line_point,(0,0,255),1)

    #Draw cropped area
    cv.rectangle(output,(tunnel[1],tunnel[0]),(tunnel[3],tunnel[2]),(255,0,0),3)
    cv.rectangle(output,(crop[0],0),(crop[1],width),(255,0,0),3)


#------------Main Loop-----------#

while(cap.isOpened()):
    #Read frame
    ret,frame = cap.read()
    #Quit if no frame
    if frame is None:
        break
```

```
    #Attempt to connect to arduino
    if connection == False:
        print("Trying to reconnect")
        try:
            connection = telnetlib.Telnet(ip,port,2)
        except:
            connection = False
            print("No Connection")

    #Display output
    output = frame+0

    #Crop frame
    frame[:,crop[1]:width,:] = 0
    frame[:,0:crop[0],:] = 0
    frame[tunnel[0]:tunnel[2],tunnel[1]:tunnel[3],:] = 0

    #Read input msg
    prevstate = state + 0
    if connection != False:
        try:
            msg = connection.read_eager()
            if len(msg)>0:
                state = msg[-1]
        except:
            connection = False

    #-----Finding the targets-----#

    #Isolate green channel and low pass filter
    mask_th_tgt = linrgb(frame,[-2,4,-2])
    mask_th_tgt = cv.medianBlur(mask_th_tgt,5)

    #Thresholding and finding countours
    thresh_lower = 150
    cv.imshow('Mask_t',mask_th_tgt)
    _,mask_th_tgt = cv.threshold(mask_th_tgt,thresh_lower,255,cv.THRESH_BINARY)
    contours,_ = cv.findContours(mask_th_tgt,cv.RETR_TREE,cv.CHAIN_APPROX_NONE)

    #Store targets in a list
    tgt_count = 0
    targets = []
    for contour in contours:
        (x,y,w,h) = cv.boundingRect(contour)
        if cv.contourArea(contour)>20:
            tgt_count += 1
            cv.rectangle(output,(x,y),(x+w,y+h),(0,255,255),3)
            targets.append([
                int(x+w/2),
                int(y+h/2),
                nearestmultiple( np.linalg.norm([x+w/2-T_coords[0],y+h/2-
T_coords[1]]) ,30)
            ])

    #Sort by distance, and then by y coordinate
    targets = sorted(targets, key=lambda x:x[1])
    targets = sorted(targets, key=lambda x:x[2])


    #-----Finding the AGV------#

    #Isolate red and blue channel and low pass filter
    mask_th_agv = linrgb(frame,[2,-4,2,0])
    mask_th_agv = cv.medianBlur(mask_th_agv,5)

    #Thresholding and finding countours
```

```
    cv.imshow('Mask_agv',mask_th_agv)
    _,mask_th_agv = cv.threshold(mask_th_agv,50,255,cv.THRESH_BINARY)
    contours,_ = cv.findContours(mask_th_agv,cv.RETR_TREE,cv.CHAIN_APPROX_NONE)

    #Bounding rectangle
    markers_V = []
    markers_H = []
    for contour in contours:
        #Find bounding rectangle
        rect = cv.minAreaRect(contour)
        (cx,cy), (w,h), rot_angle_rect = rect
        #Test if valid rectangle
        if
valid_rect(w,h,markerV['min_area'],markerV['max_area'],markerV['min_ratio'],markerV['ma
x_ratio'],cv.contourArea(contour),markerV['ca_th']):
            markers_V.append(rect)
            box_test = np.int0(cv.boxPoints(rect))
            cv.drawContours(output, [box_test], 0, (0,255,0), 3)

        if
valid_rect(w,h,markerH['min_area'],markerH['max_area'],markerH['min_ratio'],markerH['ma
x_ratio'],cv.contourArea(contour),markerH['ca_th']):
            markers_H.append(rect)
            box_test = np.int0(cv.boxPoints(rect))
            cv.drawContours(output, [box_test], 0, (0,255,0), 3)


    #-----Updating location of the AGV------#

    if len(markers_V)*len(markers_H) == 1:
        #Box
        rect_marker_v = markers_V[0]
        rect_marker_h = markers_H[0]

        #Update Rot angle and coords
        rot_angle = np.arctan2(rect_marker_h[0][1]-rect_marker_v[0][1],
rect_marker_h[0][0] -rect_marker_v[0][0])
        agv_coords[0] = int((1-agv_COR)* rect_marker_h[0][0]
+agv_COR*rect_marker_v[0][0])
        agv_coords[1] = int((1-agv_COR)* rect_marker_h[0][1]
+agv_COR*rect_marker_v[0][1])

    #Drawing visuals
    draw_visuals(output,agv_coords,rot_angle)

    #-----Navigation------#
    #Init per loop
    motor = [0,0,0,0]
    ArrivedDestination = False

    #Set target
    if state == 1:        #State = 1
        if len(targets)>0:
            nav['target'] = targets[0][0:2]
            nav['type'] = 't'
        else:
            #If nothing is seen, go to preset location and grab air
            nav = {'type':'t','target' : [70,250]}

    if state == 3:    #State = 3
        if nav['type'] != 'p':
            nav['target'] = [T_coords[0]-90,T_coords[1]]   #Waypoint
            nav['type'] = 'w'
        else:
            nav['target'] = [T_coords[0]+250,T_coords[1]]   #Final point

    #Draw line to target
```

```
cv.line(output,tuple(agv_coords),tuple(nav['target']),(0,128,255),1)

#Target angle
agv_to_target = np.array(nav['target'])-np.array(agv_coords)
distance_to_target = np.linalg.norm(agv_to_target)
diff_angle = angle(agv_to_target,[np.cos(rot_angle),np.sin(rot_angle)])

#Reached target
if distance_to_target < min_dist_target and nav['type'] == 't':
    action['mode'] = 'stop'
    motor = [0,0,0,0]
    ArrivedDestination = True

#Reached final point
if distance_to_target < tol_dist_point and nav['type'] == 'p':
    action['mode'] = 'stop'
    motor = [0,0,0,0]
    ArrivedDestination = True

#Reached waypoint
if distance_to_target < tol_dist_point and nav['type'] == 'w':
    action['mode'] = 'stop'
    motor = [0,0,0,0]
    if state == 1:
        nav['type'] = 't'
    elif state == 3:
        nav['type'] = 'p'

#Reset action if timer is zero
if action['timer'] == 0 and action['mode'] in ['fwd','rot']:
    action['mode'] = 'none'
    action['timer'] = 10
    motor = [0,0,0,0]

#Reset action if angle is too far off
if abs(diff_angle) > tol_angle and action['mode'] != 'rot':
    action['mode'] = 'none'
    action['timer'] = 10

#If no action taken currently, take new action
if action['mode'] == 'none' or action['timer'] <= 0:
    if abs(diff_angle) < tol_angle:
        #Move forward to target
        action['mode'] = 'fwd'
        action['timer'] = 10000
        action['dir'] = 0
    else:
        #Rotate to target
        action['mode'] = 'rot'
        action['timer'] = int(abs(diff_angle) * ang2t)
        action['dir'] = (1-np.sign(diff_angle))/2



#Set motor speeds based on current action
if action['mode'] == 'fwd':
    motor = [1,action['dir'],1,action['dir']]
if action['mode'] == 'rot':
    motor = [1,action['dir'],1,1-action['dir']]

#Decrement timer
action['timer'] -= 1

if state not in [1,3]:
    ArrivedDestination = False

#-----Send signal to arduino------#
```

```
    motor = int("".join( [str(int(i)) for i in motor] ),2)

    if ArrivedDestination:
        motor += 16   #Set 5th bit to high if arrived at destionation

    if state ==1 or state == 3:
        try:
            connection.write(bytes([motor]))
        except:
            connection = False
            print("Message failed to send")

    #-----Show output------#

    #Show video output
    cv.imshow('output',output)

    #Press q to quit
    if cv.waitKey(1) & 0xFF == ord('q'):
        cap.release()
        cv.destroyAllWindows()
        break



#-----End sequence------#

cap.release()
cv.destroyAllWindows()
```