eYSIP2017

Formation Control of Multiple Swarm Robots

Chirag Shah
Om Singh
Mentors: Abhinav Sarkar
Avinash Kumar Dubey
Duration of Internship: 22/05/2017 – 07/07/2017

2017, e-Yantra Publication

Contents

Formation Control of Multiple Swarm Robots

1.1 Abstract

Implement formation control over a group of Spark V robots. We are using an overhead camera setup for localization of robots. Robot's position and orientation will be given by aruco markers.
The project is inspired by swarm behavior in bees and ants. This methodology is a hybrid between true swarm behavior and a complete master-slave methodology for controlling the bots. Localization of robots is achieved by an overhead camera and markers placed on the robot. The Master (Laptop) detects the markers and hence obtains the position and orientation of each robot. Any point in the arena can be chosen as a goal point for the robot, the master sends the robot's location in real time and the goal co-ordinate to the robot and the robot moves accordingly to the goal point avoiding other robots present in the way. Similarly multiple points are chosen and multiple robots are directed to desired goal points resulting in a desired formation.

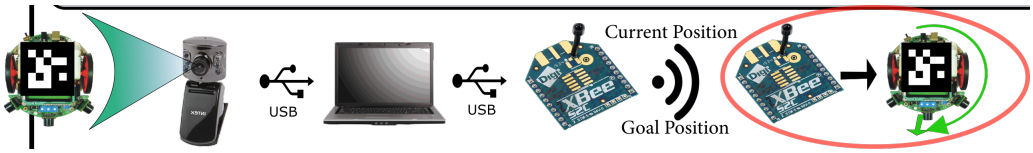
1.2 Project Overview

The objective of the project is to control multiple robots simultaneously to form any desired shapes. The shapes can be predefined: like the shapes of letters, or can be drawn on the screen. The robots will then attempt to form the shapes.

The robots used are Spark V from Nex Robotics. The robots are controlled using a python script running on a laptop (master). The robots are localized using ArUco markers placed on the robots. The markers provide the coordinates and the orientation of each of the robots. OpenCV libraries are used along with python for the detection of ArUco markers.

XBee radio modules are used for the communication between the laptop and the robots.

The laptop can individually communicate with each robot. It sends the robot's position and orientation to the robot along with the desired position of that robot. The robot then moves towards the required position using a Go-to-Goal PID controller running on the robot. Thus multiple robots are directed to the required positions to form a desired shape.



1.3 Completion status

- Aruco markers are reliably used for obtaining the position and orientation of Spark V robots
- Go-To-Goal is implemented using a PID controller on the Spark V robot
- The robots can form most letters
- A desired formation can be obtained given by drawing it on the screen

1.4 Hardware parts

1. Spark V - NEX Robotics
2. Xbee S2C Zigbee Module
3. Camera - Intex 306IT
4. Laptop

Spark V

Spark V Robot is a low cost robot designed for robotics hobbyists and enthusiasts. It is jointly designed by NEX Robotics with Department of Computer Science and Engineering, IIT Bombay.
Microcontroller: Atmel ATMega16A
Refer the hardware and software manuals for more details.

1.5 Software

1.5.1 IDEs used

1. Atmel Studio V7
2. Pycharm 2016.2.3
3. Python IDLE v2.7.12

1.5.2 Libraries used for Python v2.7.12

1. OpenCV Version-3.2.7
   Library used for image processing in python.

2. OpenCV-contrib
   Wrapper package for OpenCV python bindings. This is used for aruco marker detection.

3. Pyserial

4. Numpy (1.12.1)

5. XBee library

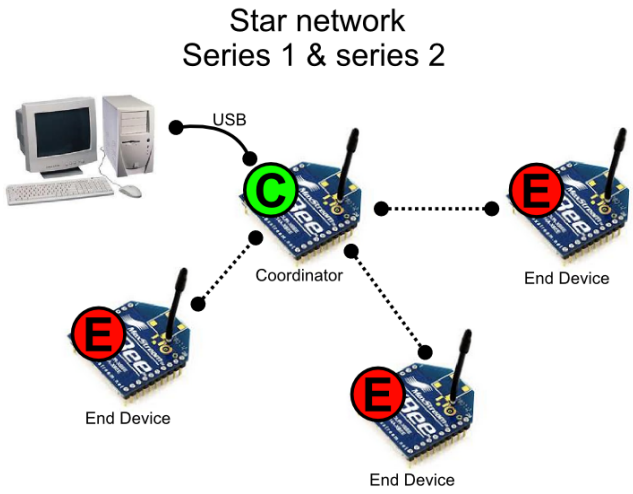
6. Math library

1.5.3 Other Software

1. PIP
   PIP is a package management system used to install and manage software packages written in Python. PIP is already installed if using Python 2.7.9 or Python 3.4 binaries downloaded from python.org, but you'll need to upgrade pip.

2. XCTU
   Software used to configure Xbee modules

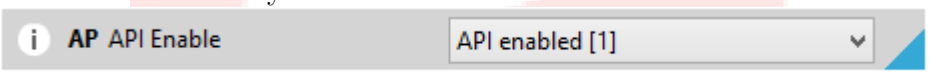
XBee Configuration



Set the baudrate to 115200 of every Xbee module. Baudrate is the speed of communication between XBee and connected device.

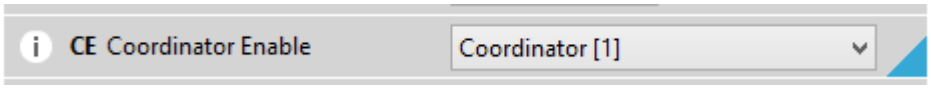
2.1 API mode

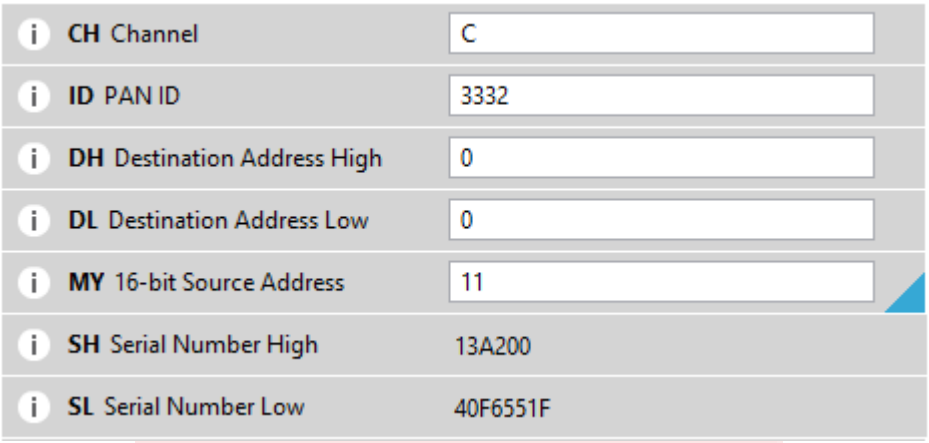
Set API Enable for every xbee module



In API mode data, is sent from one XBee to another in packets which ensures that the data is not corrupted and is legal. It also enable point to point communication of two Xbee devices without disturbing other Xbee on the channel. Each XBee on the channel has a unique id which is embedded in the packet so the XBee with the id will decode the packet.

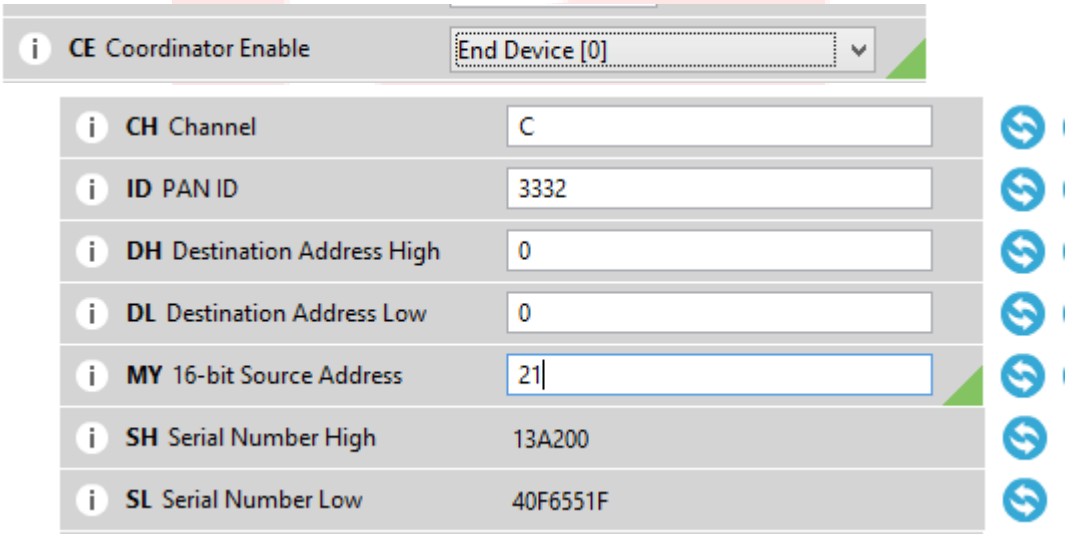
2.1.1 Coordinator





DH-0
DL-0
My id-11 #unique id for each xbee
API mode :Enabled

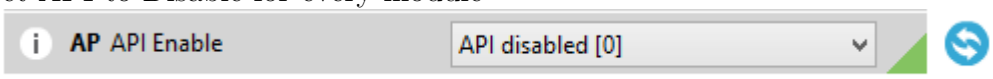
2.1.2 End device



DH-0 DL=0
My id:21 #unique id for every end xbee. For example(22,23,24 etc..)
API mode : Enabled

2.2 AT mode

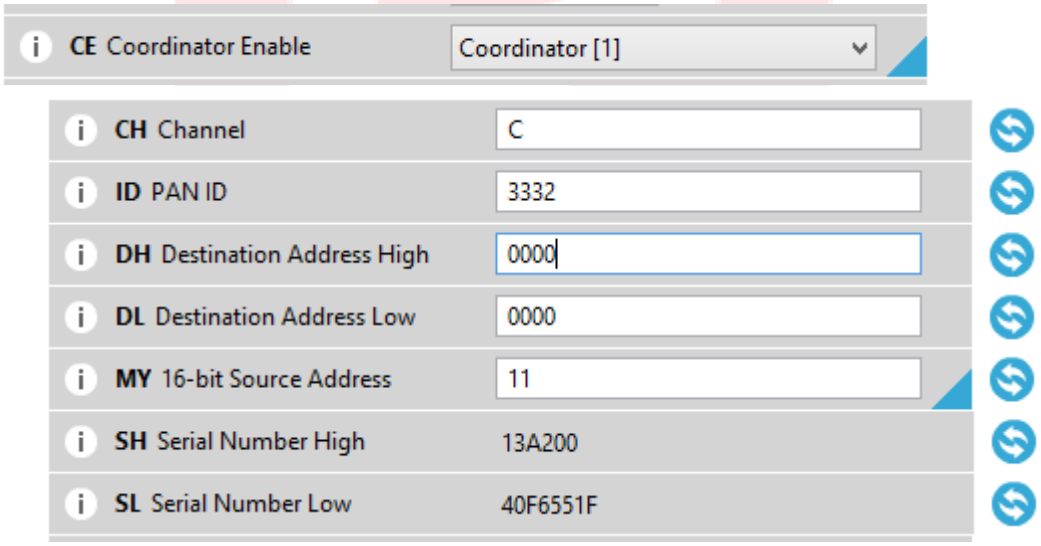
Set API to Disable for every module



In AT mode data is send character by character to every XBee or only a pre-defined XBee based on the destination address. Chances of data corruption is higher.

Sending all the data to each and every XBee is tough to use in swarm robots as the microcontroller has to be able to receive and decode all the data which is sent to all the XBees. Microcontroller will need to have good data handling capabilities to handle so much data. AT mode is good for point to point communication.

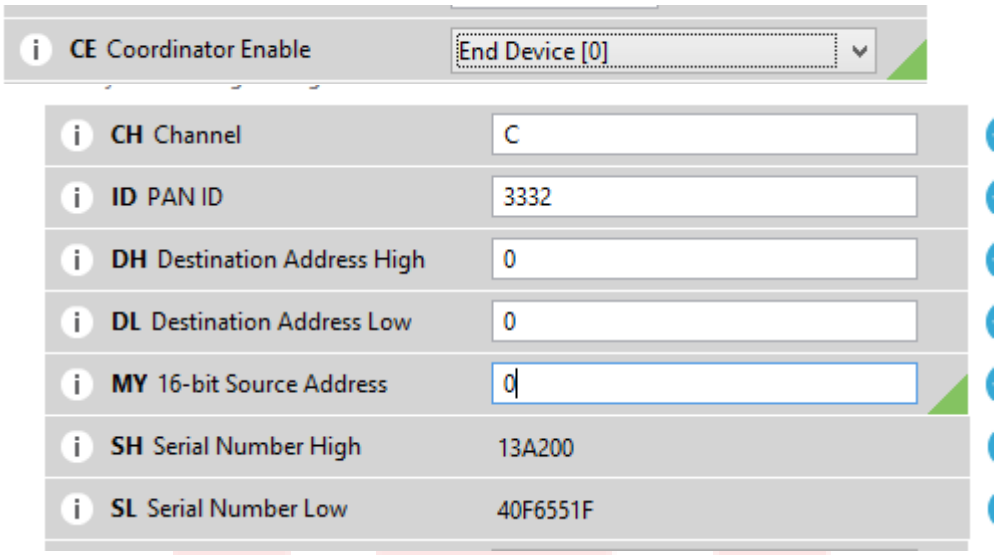
2.2.1 Coordinator



Broadcast Mode
DH-0000
DL-0000
My id: "any positive int 16 bit"

Point to Point DH-High serial number of end module
DL-Low serial number of end module Or My id of end module
My id: "any positive int 16 bit"

2.2.2 End device



Broadcast Mode DH-0000
DL-0000
My id: 0000

Point to Point DH-0000
DL-0000
My id: "any positive int 16 bit"

Spark V Programming

1. Setting up Spark V
2. XBee interfacing with Spark V
3. Go-to-Goal and PID controller
4. Collision avoidance

Get the Spark V code on the git-hub repository

3.1 Setting up Spark V

Refer the hardware and software manuals for using the Spark V robot.

3.1.1 Programming the Spark V using AVRDude

1. Copy the AVRDude folder onto your computer
2. Open the Command Prompt in the AVRDude folder
3. Run - stkrun.bat "SparkV.hex" to program the Spark V with the file "SparkV.hex". Similarly you can program any file by giving the location of the file along with the file name.

3.1.2 Programming the Spark V using Atmel studio and AVRDude

1. Copy the AVRDude folder onto your computer
2. Open the `Tools>External` tools in Atmel Studio
3. Add STK500v2 as a tool
4. Name - STK500v2
5. Set the location of AVRDude.exe in the commands box
   For eg `C:\Users\Chirag\Desktop\eYSIP_2017\AVRDude\avrdude.exe`
6. Arguments box - -c stk500v2 -p m16 -P NEX-USB-ISP -U flash:w:`$(TargetName).hex:i`
7. Initial Directory - `$(TargetDir)`
8. Go to `tools>optinons>environment>keyboard`
9. Select Tools.ExternalCommand1 and set any key as a shortcut (say alt+p)

You can now directly program Spark V from Atmel Studio using the shortcut.

3.2 XBee interfacing with Spark V

The data is received character by character over UART. When a character is received it is very briefly stored in a buffer and a flag is set. This is done internally by the microcontroller. This flag then triggers an interrupt.
The interrupt then decides what to do with this character of data. If it is a valid byte of data it will be stored in the array "data_string_var". After a complete packet of data is received it is copied to the array "data_string".

<#.....#> is a proper string which then saved to "data_string" by the ISR.

Listing 3.1: Interrupt Service Routine

```
ISR(USART_RXC_vect)
{
   previous_data = data;
   data = UDR;

   strcpy((char*) data_string, (const char*)data_string_var);
       //Entire string received!! Save it!!

   if (previous_data==0x3C && data == 0x23)//< and #
   {
      append_on = 1;
      i=0;
   }

   else if (previous_data==0x23 && data==0x3E)//# and >
   {
      append_on=0;
      strcpy((char*)data_string, (const char*)
          data_string_var); //Entire string received!! Save
          it!!
   }

   else if (append_on==1 && data != 0x23)
   {
      data_string_var[i]=data;
      i++;
   }
}
```

3.3 Parsing the String

Listing 3.2: String Parsing Function

```
void update_values()
{
   char parts[10][5];
   char data_string1[40];
   strcpy((char*)data_string1, (const char*)data_string);

   char *p_start, *p_end;
   unsigned char i=0;
   p_start = data_string1;

   while(1)
   {
      p_end = strchr(p_start, '/');
      if (p_end)
      {
         strncpy(parts[i], p_start, p_end-p_start);
         parts[i][p_end-p_start] = 0;
         i++;
         p_start = p_end + 1;
      }
      else
      break;
   }
      x_current = atoi(parts[1]);
      y_current = atoi(parts[2]);
      theta_current = abs(atoi(parts[3])-360+180-360);
          //(0)-(360)
      x_req = atoi(parts[4]);
      y_req = atoi(parts[5]);
      theta_req = abs(atoi(parts[6])-180-360); //(0)-(360)
      trigger = atoi(parts[7]);
      trigger_angle = atoi(parts[8]);
}
```

The string is received in the form

`<#id/x_current/y_current/theta_current/x_required/y_required/theta_required/trigger/trigger_angle#>`

This is then parsed to obtain the values in int data type.

3.4 Go-to-Goal and PID

Listing 3.3: PID controller

```
int v_left,v_right,R=3.5,L=11.5,V;
float kp, ki, kd;
float w;

kp=.7;
ki=0;
kd=70;
V=400;

distance=sqrt(square(y_req-y_current)+square(x_req-x_current));
destination=(distance<100)?1:0;

if (!x_req || !y_req)
{
   hard_stop();
   destination = 1;
   return;
}

V=(distance<100)?200:400;
kp=(distance<100)?.4:.7;

if(distance>20)
{
   er=theta_current-theta_req; //-360 to 360
   er = atan2(sin(er*3.14/180),
       cos(er*3.14/180))*(180/3.14); //-180 to 180

   integral= integral+er*dt;
   derivative = (er-previous_er)/dt;
   previous_er = er;

   w=kp*er + ki*integral + kd*derivative;
   v_left=((2*V+w*L)/(2*R));
   v_right=((2*V-w*L)/(2*R));

   velocity(v_left,v_right);
   destination=(distance<100)?1:0;
}
```

```
else
{
   hard_stop();
}
```

A Go-To-Goal PID controller is implemented on the microcontroller.
The velocity of the motors is considered to be proportional to the PWM input to the wheels. The PID controller takes care of any non-linearity between the velocity and the PWM input.
The velocity and the kp values are reduced as the robot nears its goal point to avoid overshoot and to avoid oscillations.
Obstacle avoidance behavior is also turned off (by setting destination=1) as the robot nears its goal point so that it does not miss its goal point if another robot is near the goal point.

```
v_left=((2*V+w*L)/(2*R));
v_right=((2*V-w*L)/(2*R));
```

These equations convert the unicycle model to the differential drive model. The unicycle model gives the state of the robot in terms of its liner velocity 'V' (cm/sec) and its angular velocity 'w'.
The differential drive model gives the state of the robot in terms of the angular velocity (RPM) of each wheels.
While this equations are unnecessary in this case they could be made applicable if we had precise control over the angular velocity of the wheels(RPM).

We use a modified function for velocity along with the Go-to-Goal controller. Negative velocity will make the wheel rotate in the opposite direction. Giving a value greater then 255 will have no effect and it will be capped to 255.

Listing 3.4: velocity function

```
void velocity(int left_motor, int right_motor)
{
   if (left_motor>=0 && right_motor>=0)
   {
      forward();
   }

   else if (left_motor<0 && right_motor>0)
   {
      left();
      left_motor=abs(left_motor)+40; //some offset is added so
          that the wheel still turns at lower values
   }

   else if (left_motor>0 && right_motor<0)
   {
      right();
      right_motor=abs(right_motor)+40;//some offset is added
          so that the wheel still turns at lower values
   }
   else
   {
      back();
      left_motor=abs(left_motor);
      right_motor=abs(right_motor);
   }
   OCR1AL = left_motor;   // duty cycle 'ON' period of PWM out
       for Left motor
   OCR1BL = right_motor;   // duty cycle 'ON' period of PWM
       out for Right motor
}
```

3.5 Collision avoidance

Listing 3.5: Collision avoidance function

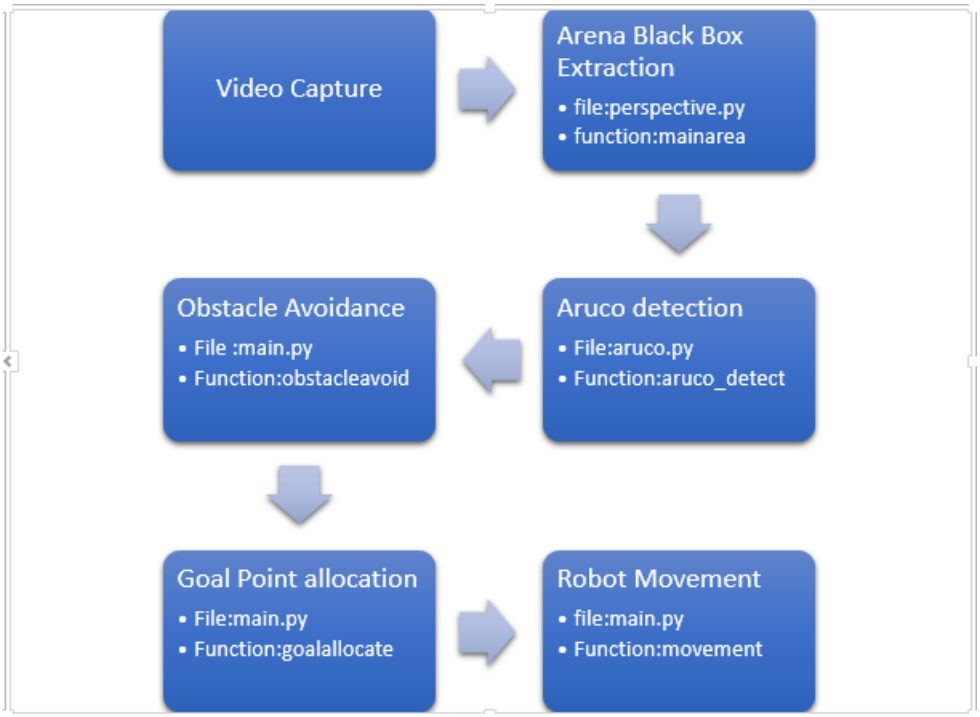
```
void avoid_obstacle()
{
   if (trigger==1)
   {
      hard_stop();
      _delay_ms(100);
      velocity2(90, 90);
      if (trigger_angle > 50) back_mm(50);
      left_degrees(trigger_angle);
      forward_mm(50);
      hard_stop();
   }
   else if (trigger==2)
   {
      hard_stop();
      _delay_ms(100);
      velocity2(90, 90);
      if (trigger_angle > 50) back_mm(50);
      right_degrees(trigger_angle);
      forward_mm(50);
      hard_stop();
   }
}
```

If a trigger is received the robot executes this function. The robot reduces its speed, turns a specified number of degrees and moves forward. If the angle is greater then a certain value(50) the robot moves backwards before taking a turn. This is to avoid a head-on collision with another robot.

Python Programming

Get the python script on the git-hub repository

1. Video capture
2. Arena black box extraction
3. ArUco detection
4. Collision avoidance
5. Goal point allocation
6. Robot movement
7. Xbee communication



4.1 Video Capture

To capture image using usb camera python uses a object created by the function cv2.VideoCapture('camera number'), read() method of the cv2.VideoCapture('camera number') is used to read single frame . This full process is executed in a infinite while loop to process frame after frame and get a resulting video.

Listing 4.1: Capturing Video

```
cap=cv2.VideoCapture(1)
while(True):
   ret,frame=cap.read()
   cv2.imshow('window1',frame)
   k=cv2.waitKey(20) & 0xFF
   if k==27:
      cv2.destroyAllWindows()
            break
'''cap is a object of cv2.VideoCapture'''
```

4.2 Arena Black Box Extraction

The main arena area is enclosed by a black boundary which can be extracted using the contour function.This function detects the shape with same color or intensity.

Steps:

1. Convert the image into a grayscale image
2. Threshold the image
3. Apply the contour function
4. Loop over all the contours and find the desired contour by using functions like approxPolyDP or area
5. Perspective Transform

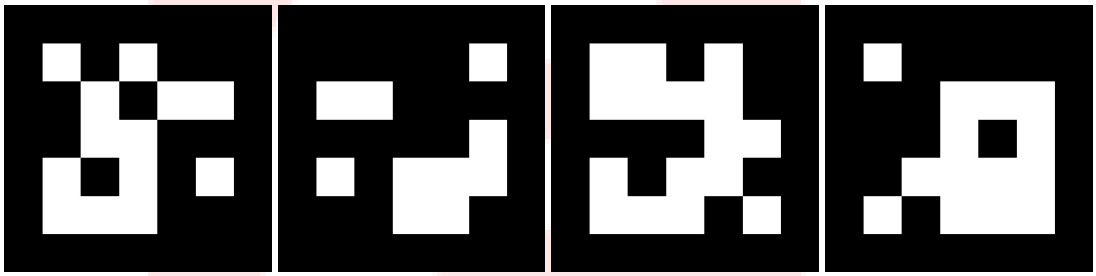
Listing 4.2: Black Box area crop and perspective trasform

```
img_gray=cv2.cvtColor(img_rgb,cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(img_gray,127,255,cv2.THRESH_BINARY)
_,contours,hierarchy=cv2.findContours(thresh,cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)
for contour in contours:
          approx =
              cv2.approxPolyDP(contour,0.01*cv2.arcLength(contour,True),True)
          area = cv2.contourArea(contour)
          if (area >100000)and area<250000 :
              x,y,w,h = cv2.boundingRect(contour)
              cv2.rectangle(img_rgb,(x,y),(x+w,y+h),(0,0,255),2)
pts1 = np.float32([[x,y],[x+w,y],[x,y+h],[x+w,y+h]])
pts2 = np.float32([[0,0],[w,0],[0,h],[w,h]])
M = cv2.getPerspectiveTransform(pts1,pts2)
arena = cv2.warpPerspective(crop_img,M,(w,h))
M = cv2.getPerspectiveTransform(pts1,pts2)
dst = cv2.warpPerspective(img,M,(300,300))



'''x,y are the coordinate of left top corner of the rectangle ,w
   and h are width and height of rectangle respectively,dst is the
   transformed image'''
```

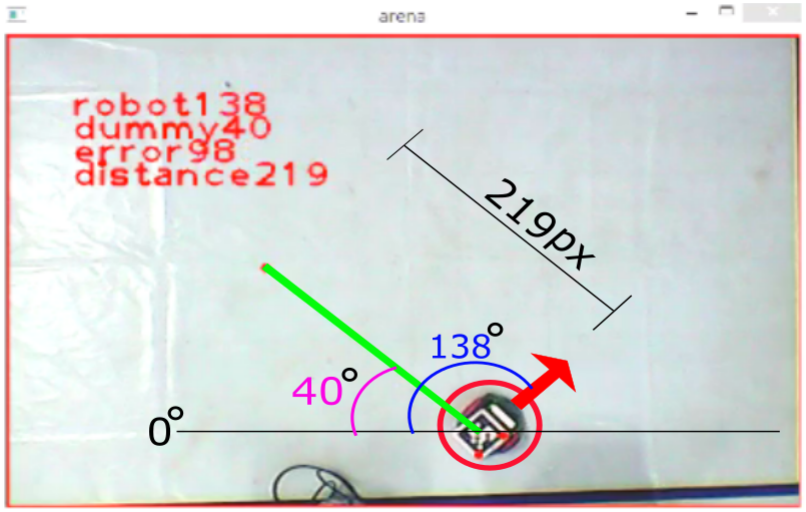
4.3 ArUco Detection

An ArUco marker is a synthetic square marker composed by a wide black border and a inner binary matrix which determines its identifier(id). The black border facilitates its fast detection in the image and the binary codification allows its identification and the application of error detection and correction techniques. The marker size determines the size of the internal matrix. For instance a marker size of 4x4 is composed by 16 bits.

ArUco marker pattern is based on hamming code where two column represents a data bit and three column represent parity bits. Data bits when decoded gives the marker id whereas parity bits are useful in calculating the orientation and better detection of marker id.



ArUco detection is easily be done by using the arucodetectMarkers function of the opencv ArUco library. The function requires a grayscale image, aruco dictionary and the returned value from detectorParameter_create. detectParameter has data members such as thresholding, refinement, corner refinement etc to detect the ArUco from the image. aruco.detectMarkers function returns the coordinates of the corners of the detected markers and their respective ids. These corners are used to find the centroid and orientation of the aruco marker with respect to the horizontal axis in range -180 degrees to 180 degrees.
A dictionary named robot is created having key as id of the marker containing items x and centroid and orientation angle of marker.
Vist the OpenCV ArUco docs for more information.



Listing 4.3: Robot id as dictionary key with item centroid and angle

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
aruco_dict =aruco.Dictionary_get(aruco.DICT_5X5_250)
parameters = aruco.DetectorParameters_create()
corners,ids,_=aruco.detectMarkers(gray,aruco_dict,parameters=parameters)

for marker in range(len(ids)):
   x_center= int((corners[marker][0][0][0] +
       corners[marker][0][1][0] + corners[marker][0][2][0] +
       corners[marker][0][3][0])/4)
   y_center= int((corners[marker][0][0][1] +
       corners[marker][0][1][1] + corners[marker][0][2][1] +
       corners[marker][0][3][1])/4)
   cv2.circle(frame,(x_center,y_center),2,(0,0,255),2)
   x1 = int(corners[marker][0][0][0])
   x3 = int(corners[marker][0][3][0])
   y1 = int(corners[marker][0][0][1])
   y3 = int(corners[marker][0][3][1])
   pt1=(x3,y3)
   pt2=(x1,y1)
   cv2.circle(frame,pt1,2,(0,0,255),2)
   cv2.circle(frame,pt2,2,(0,0,255),2)
   angle = angle_calculate(pt1,pt2)
       robot[int(ids[marker])]=(int(x_center),int(y_center),int(angle))

print robot
```

4.4 Collision Avoidance

Since Atmega16 cannot handle data for all the robots in the arena therefore we need a collision avoidance check on the master which will only send a unique trigger command to the robot when its near collision.

Collision avoidance requires two factors, relative angle and distance between the robots.

Algorithm Used for collision avoidance:

1. Calculate distance between robots.
2. Calculate Relative angle between robots.
3. Send command to the particular robot whose distance and relative angle are below threshold.

Distance matrix: A distance matrix is a square matrix (two-dimensional array) containing the distances, taken pairwise, between the elements of a set.

Listing 4.4: Distance Matrix

```
for i in robot:
   for j in robot:

      distancemat[i][j]=distance((robot[i[0],robot[i][1]),
      (robot[j][0],robot[j][1])) }
```

Angle Matrix: An angle matrix is a square matrix (two-dimensional array) containing the angle, taken pairwise, between the elements of a set. In this case angle is the relative angle that is angle of j with respect to i and vice-verse.

Listing 4.5: Angle Matrix

```
for i in robot:
   for j in robot:
      obsy=(robot[j][1]-robot[i][1])
      obsx=(robot[j][0]-robot[i][0])
      angle_360=angle_calculate((robot[j][0],robot[j][1]),
         (robot[i][0],robot[i][1]))-robot[i][2]
         #angle b/w robot i and robot j, with respect to angle of
            robot i (relative angle)
      anglemat[i][j]=math.degrees(math.atan2(math.sin(angle_360*(math.pi/180)),math.cos
          #relative angle in range(-180 to 180)
```

Apply a threshold on distance and angle to trigger robot to avoid collision. np.where function of numpy is used to check the array and return the index of the element which is below the threshold value.

Listing 4.6: Searching matrix

```
item=np.where(np.logical_and(distancemat[i][j]<80,distancemat[i][j]>0))
an=np.where(np.logical_and(anglemat[i][j]<90 , anglemat[i][j]>-90))
if (len(an[0])!=0) and i!=j and len(item[0])!=0 :
   obstacleavoid=True
```

4.5 Goal Point allocation

We use three ways to mark goal points on the canvas or image. The three list goal, path, points are used to keep track of the points.

- Goal-It contains the actual goal point for the robots.
- Path-It contains the possible goal points for all robots.
- Points-It contains raw points from the shape drawn on canvas.

The process of Goal allocation starts from points array if there exist one, then preferred points are transferred to the path array from where each robot chose its closes goal point and the chosen point is transferred to the goal array and becomes the true goal point for the particular robot.

1. Predefined points:

   These points are directly defined in the array manually.

   path= [(237, 160), (350, 179), (419, 132), (370, 69), (278, 79), (255, 273), (347, 296), (434, 279)]

2. One by one points:

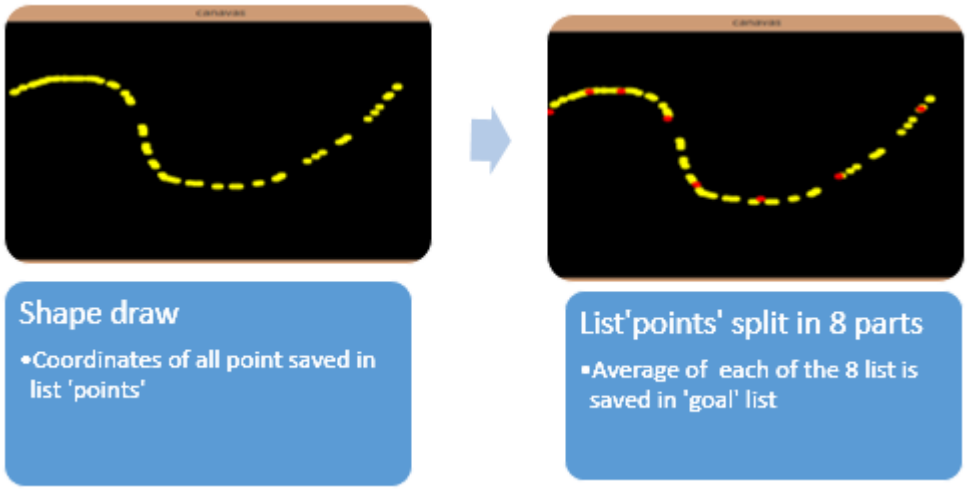
   To select points on image we use an opencv function

   cv2.setMouseCallback('Window', function to call) This function sets the handler for mouse activity calls the 'function to call' when ever mouse is in the specified window.

Listing 4.7: Mouse activity call back function

```
def draw_shape(event,x,y,flags,param):
    if event ==cv2.EVENT_RBUTTONDOWN:
      ''''Statement'''
    if event == cv2.EVENT_LBUTTONDOWN:
      '''Statement'''
    elif  event == cv2.EVENT_MOUSEMOVE:
      '''Statement'''
    elif event == cv2.EVENT_LBUTTONUP:
      '''Statement'''
```

3. single stroke shape:



In this method all the points of the shape drawn on canvas are saved in the 'Points' array ,then the points are split into 8 parts maybe or may not be equal ,then average of each list is saved in 'path' array.

Listing 4.8: Point selection from a shape

```
This function takes an array and divides it into 8 parts
def chunkIt(seq, num):
  avg = len(seq) / float(num)
  out = []
  last = 0.0
  while last < len(seq):
    out.append(seq[int(last):int(last + avg)])
    last += avg
  return out
```

```
Averaging points in each list created by chunkIt function.
for i in range(len(chunk)):
   sumx=0
    sumy=0
    for j in (chunk[i]):
      sumx=(j[0]+sumx)
        sumy=(j[1]+sumy)
    tx=sumx/len(chunk[i])
    ty=sumy/len(chunk[i])
    path.append((tx,ty))
    return path
```

The final step is to move the points in path list to goal list. This is done by calculating distance between the robot and all the points in path list and chose the minimum distance point.

Listing 4.9: Goal allocation

```
initailly the Goal list is filled with (0,0)
 for botid in robot:
        distance_list=[[]]
        for count,j in enumerate(path):
            pt3=(robot[botid][0],robot[botid][1])
            pt4=j
            distance_list[count]=distance(pt3,pt4)
        goal_index=distance_list.index(min(distance_list))
        if (goal[botid]==(0,0)) and path!=[]:
            goal[botid]=path[goal_index]
            path.remove(goal[botid])
```

4.6 Robot Movement

To make the robot move for every frame a XBee packet is sent to bot containing its current robot id, location , orientation, its destination, angle of goal point , flag, rotation angle. Flag and rotation are 0 in case of no obstacle avoidance mode and 1(left) or 2(right) in obstacle avoidance mode. Similarly rotation angle is 0 for no obstacle avoidance mode and some integer for obstacle avoidance mode.

Listing 4.10: Robot Movement

```
angle_dummy=angle_calculate(goal point,robot location)
'''here angle dummy is the angle of line between goal point robot
    location , w.r.t horizontal axis'''
        if obstacleavoid==False:
            if botid==0:
               xbees.tx(dest_addr='\x00\x20',
               data='<#'+str(i)+'/'+str(robot[i][0])+'/'+str(robot[i][1])+
               '/'+str(robot[i][2]+360)+
               '/'+str(dummy[0])+
               '/'+str(dummy[1])+
               '/'+str(angle_dummy+360+
               '/'+'0'+'0'+'/#>')
```

4.7 Xbee Communication

To enable serial communication we use a serial library of python. It opens the COM port on which the XBee is connected and sets the baudrate for sending data.

In AT mode write method of 'serial' object can be used which does not create packets but sends the data serially character by character .

Xbee library of python is used when Xbee is configured in API mode. The Xbee.tx function automatically create a acceptable packet for other xbees of a unique id (destination address) and the given data.

Listing 4.11: Xbee API mode packet generation

```
xbees.tx(dest_addr='\x00\x20',data='any string')
```

Instructions for user

1. Upload the AVR code to all the Spark V. The code does not need to be modified for each robot.

2. Print 0 through 7 aruco marker using the script "generate.py" in the test code folder. Use the dictionary `5x5_250`. Keep in mind that not all encodings of aruco markers are the same, check your source for obtaining aruco markers. Place them on the robot in the same orientation they are generated.

3. Configure 8 XBees as end devices in API mode with their "my id" ranging from 20-27 and insert them in the bots with aruco markers 0-7 respectively.

4. Configure one XBee as coordinator in API mode with its "my id" as 11 and use it with the master(laptop).

5. Open the python script main.py on idle/PyCharm. You will need to change the COM port for the XBee in "serial.Serial()" and the camera number in "VideoCapture()" in the main file.

6. Run the python script "main.py".

7. `!` will open the interface for drawing shapes. Draw shapes in a single stroke. # will then get the goal points from the shape and start the robots.

8. A E e N T R Y are the letters which are predefined. The robots will form these shapes on pressing these keys.

9. `/` will stop the bots. If desired you can now manually give 8 goal points for ids 0-7 by clicking on the desired area on the window.

Conclusion

6.1 Bug report

With the current python script, occasionally the error "index list out of range" occurs during runtime.
Positions of the goal points sometimes are very close together. Averaging of points in the list may not be appropriate for some shapes.
Shapes need to be drawn in a single stroke.

6.2 Challenges

- When using the on board IR sensors, they would detect any obstacle successfully, but they would also be triggered by the IR from the other robots in the arena even from a considerable (3-4 feet). It was concluded that the only way to use the onboard sensors was to make hardware changes like adding a transistor for switching the sensors.

- If the XBees were kept in AT mode so that the data of every robot is received by each robot, the robot was not able to handle a large amount of data (of more then 3-4 robots). The robot would go into some undefined behavior and had to be reset. This was suspected to be because having a long interrupt function for communication on the robot and the buffer becoming full with a large amount of data.
  This was an issue for having collision avoidance on the robot. This might be solved by having better communication algorithms (like circular buffers) on the robot.

6.3 Future Work

This is potentially a very powerful swarm robotics system as we can obtain the state of each robot with respect to a global co-ordinate system.
The robot can know the state of all the robots. Hence collision avoidance and neighboring robot detection can be moved to the robot. Since the robot knows the position of every robot, algorithms for different swarm behaviors can be developed.
Complex swarm behaviors can be obtained by implementing the algorithms for inter-robot communication.
Algorithm for drawing shapes and for goal point allocation for robots can be improved.

Bibliography

[1] OpenCV docs

[2] Aruco detection blog

[3] OpenCV docs - ArUco detection

[4] Coursera-Control of Mobile Robots

[5] XBee library for python

[6] Wikipedia - Distance martix

[7] OpenCV - User Interface

[8] Sparkfun XBee Guide

[9] API mode guide