e-Yantra Summer Internship Programme 2015

IIT Bombay

Dcumentation

PID based Path Planning

Interns:

Dhirendra Sagar
Email:sagar.dhirendra@gmail.com
Mob: 8858643336

Uttam Kumar Gupta
Email:23guptageek@gmail.com
Mob: 9473735800

Mentor:

Amiraj Dhawan

Under the guidance of
Prof.Kavi Arya

5 June 2015
to
8 July 2015

Contents

Abstract

- In this project we used PID controller to control the movements of Firebird V using image processing on frames of captured video.
- We detected target and source position on the basis of their colors then source follows a path towards target point with controlled motion so that it did not deviate from its path.
- When firebird V is deviated from its path, PID controller controls the left and right motor speed, accordingly it tried to regain its path as fast as possible and gives smoother movements.

Objective of the work

The aim of the project is to detect the Firebird V robot using image processing in an arena and given a moving end location, plan the robot's motion using PID closed loop feedback system. The arena will contain fixed obstacles.

Sr.No	Tasks	Deadline
1.)	Learning firebird V programming and PID controller	2 days
2.)	Develop Motion commands (Xbee communication)	1 days
3.)	Detection of Firebird V using image Processing and Map the Path for destination position	5 days
4.)	Implement the PID controller and Tune it	5 days
5.)	Testing/Documentation(usage Manual, Documented Code)	5 days

Tools Required

- Installed the following libraries:
  - **Atmel Studio 6.0**: For embedded C programming
  - **Python 2.7**: Python 2.7 IDLE is used for python programming.
  - **openCV**: This library provides various modules which are used for processing an image. color masking and contours detection etc, is done by it.
  - **numpy**: numpy used to handle array in python.
  - **X-CTU**: For configuration of xbee modules to send motion commands from laptop to firebird V.
  - **pyserial**: To send command using xbee serial communication through python.

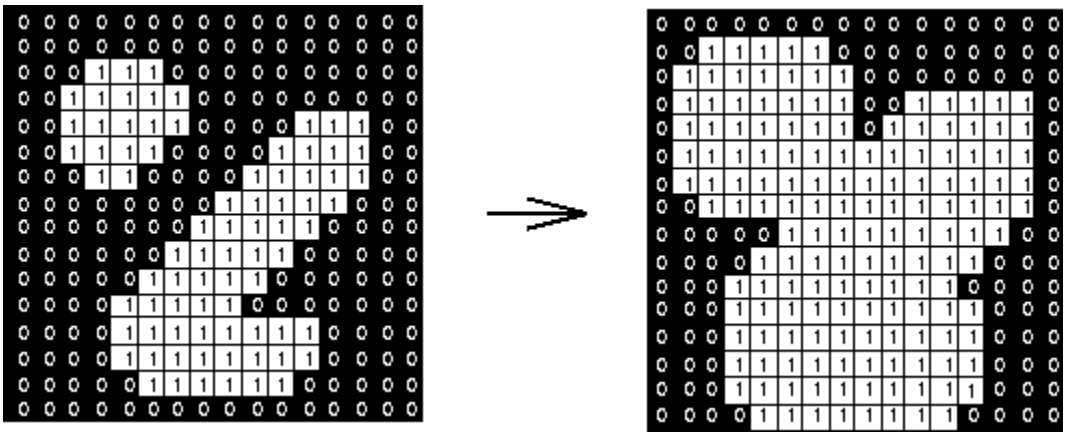
Introduction

HSV:

The first thing we usually notice about a color is its hue. Hue describes the shade of color and that color is found in the color spectrum. Red, yellow, and purple are words that describe hue. The next most significant aspect of color is the saturation,S. The saturation describes how pure the hue is with respect to a white reference.The next thing is brightness or value.This is a relative description of how much light is coming from the color. If the color reflects a lot of light, we would say that it is bright.

Masking : A mask is a black and white image of the same dimensions as the original image (or the region of interest you are working on). Each of the pixels in the mask can have therefore a value of 0 (black) or 1 (white). When executing operations on the image the mask is used to restrict the result to the pixels that are 1 (selected, active, white) in the mask. In this way the operation restricts to some parts of the image.

Dilation (Morphology Operation) : This operations consists of convoluting an image with some kernel ( ), which can have any shape or size, usually a square or circle. The kernel has a defined anchor point, usually being the center of the kernel. As the kernel is scanned over the image, we compute the maximal pixel value overlapped by and replace the image pixel in the anchor point position with that maximal value. As you can deduce, this maximizing operation causes bright regions within an image to grow (therefore the name dilation). Take as an example the image above. Applying dilation we can get:



The background (bright) dilates around the black regions of the letter. Closing It is obtained by the dilation of an image followed by an erosion.

Useful to remove small holes (dark regions).

Contours:

The contours are a useful tool for shape analysis, object detection and recognition of detected contours. Each contour is stored as a vector of points.

```
cv2.findContours(image, mode, method[, contours[, hierarchy[, offset]]])

cv2.drawContours(image, contours, contourIdx, color[, thickness[, lineType[,
```

Heapq Algorithm:

A heap is a tree-like data structure where the child nodes have a sort-order relationship with the parents. Binary heaps can be represented using a list or array organized so that the children of element N are at positions 2*N+1 and 2*N+2 (for zero-based indexes). This layout makes it possible to rearrange heaps in place, so it is not necessary to reallocate as much memory when adding or removing items. A max-heap ensures that the parent is larger than or equal to both of its children. A min-heap requires that the parent be less than or equal to its children. Pythons heapq module implements a min-heap.

Accessing Contents of Heap:

Once the heap is organized correctly, use heappop() to remove the element with the lowest value. In this example, taken from the stdlib documentation, heapify() and heappop() are used to sort a list of numbers.

PID controller:

- PID is an acronym for Proportional Integral Derivative.
- Main task of the PID controller is to minimize the error of whatever we are controlling.
- PID takes input, calculates the deviation from the intended behaviour and accordingly adjusts the output so that deviation from the intended behaviour is minimized and greater accuracy obtained.
- As the name suggests, PID algorithm consists of three basic coefficients: proportional, integral and derivative which are varied to get optimal response.

  The basic terminology we use to implement the PID Controller are:
- Error(e): The error is the amount by which robot is deviating from its set-point.
- Proportional(P): The proportional component depends only on the difference between the set point and the process variable. This difference is referred to as the Error term.
- Integral(I): The integral component sums the error term over time.
- Derivative(D): The derivative component causes the output to decrease if the process variable is increasing rapidly. The derivative response is proportional to the rate of change of the process variable.
  In PID Formula we calculate how much the output be altered using the error value.Here we use PID Formula to calculate the Correction as follows:

```
Correction = proportional*Kp + integral*Ki + derivative*Kd
```

Proportional: We would need to simply add the error value to the output to adjust the robots motion. And this would work, and is known as proportional control (the P in PID). It is often necessary to scale the error value before adding it to the output by using the constant(Kp).

$$\mathrm{proportional} = \mathrm{position} - \mathrm{setpoint}$$

But it is found that if we want a quick response time, by using a large constant Kp or if the error is very large, the output may overshoot from the set value. Hence the change in output may turn out to be unpredictable and oscillating. In order to control this, derivative expression comes to limelight.

Integral: The integral improves steady state performance, i.e. when the output is steady how far away is it from the setpoint. By adding together all previous errors it is possible to monitor if there are accumulating errors.

$$\mathrm{integral} = \mathrm{integral} + \mathrm{proportional}$$

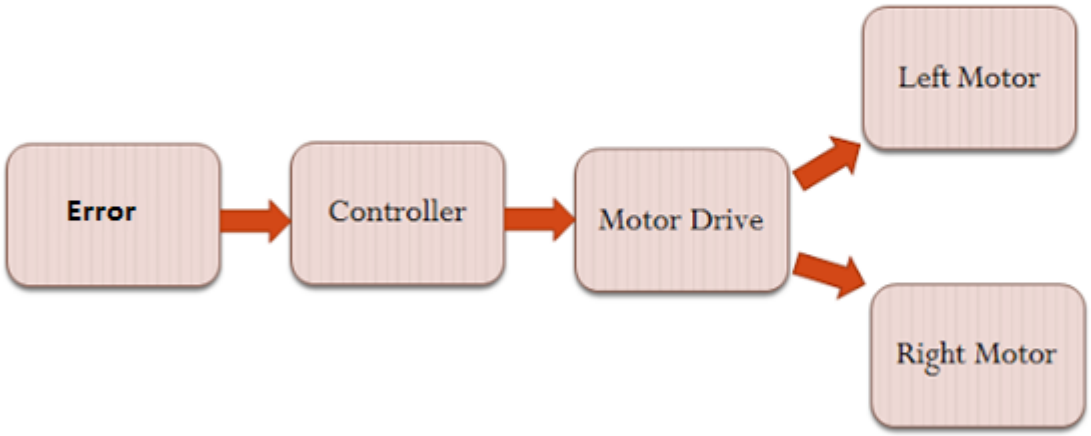
The integral response will continually increase over time unless the error is zero, so the effect is to drive the Steady-State error to zero. Steady-State error is the final difference between the process variable and set point.

Derivative: Derivative provides us the change of error. This would help us know how quickly does the error change and accordingly we can set the output.

$$\mathrm{derivative} = \mathrm{proportional} - \mathrm{last\ proportional}$$

Derivative action can compensate for a changing measurement. Thus derivative takes action to prevent more rapid changes of the measurement than proportional action.

Block Diagram



Theme Description

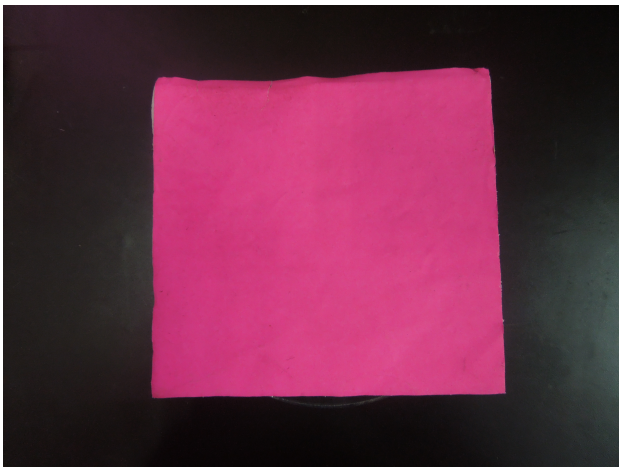
Two firebird V are used where one of them is Master which is controlled manually and other one is slave which takes commands from computer as the video is processed using the webcam fitted at top of arena. Slave Bot will follow the Master Bot as the master bot moves, by avoiding the obstacles of green color that are moving in arena. Master Bot has markers of pink color and slave has of sky blue color, small rectangle in front and big one in rear.

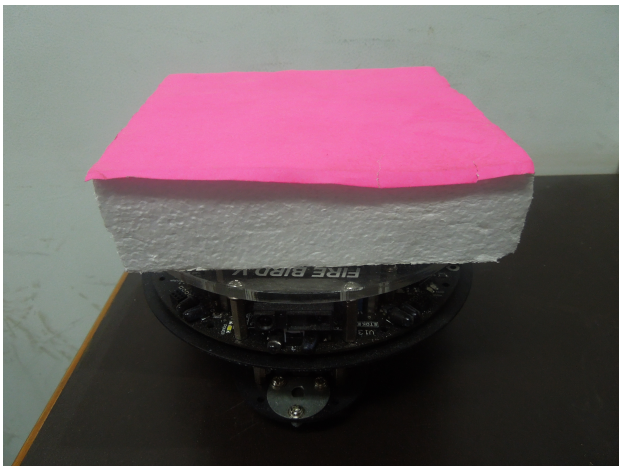
Web Cam:

To cover arena a web cam is fitted on roof of height of at least 8 feet. Web cam used is shown in figure

Master Bot:

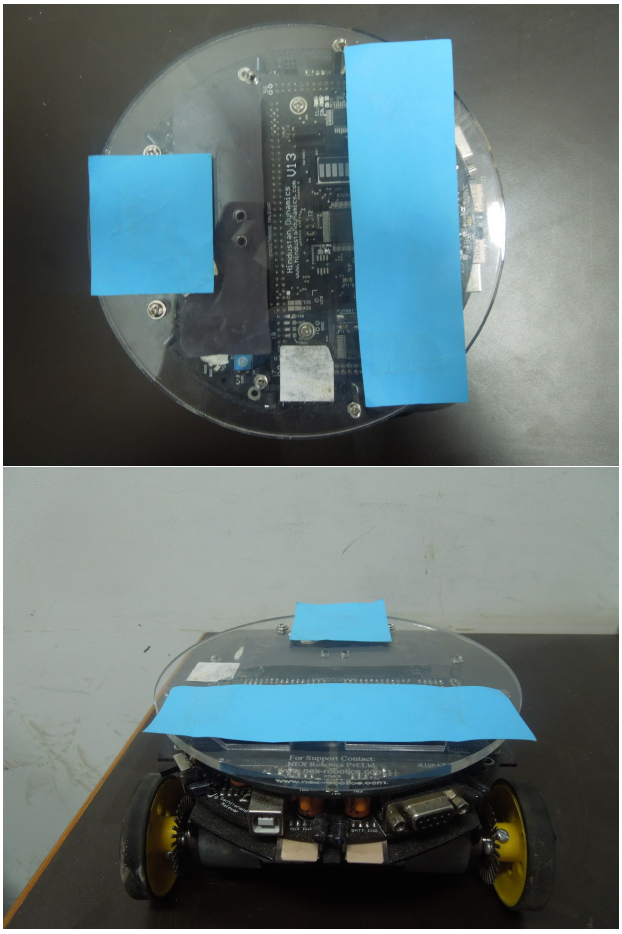
Master bot have marker of pink color on it as shown in figure.





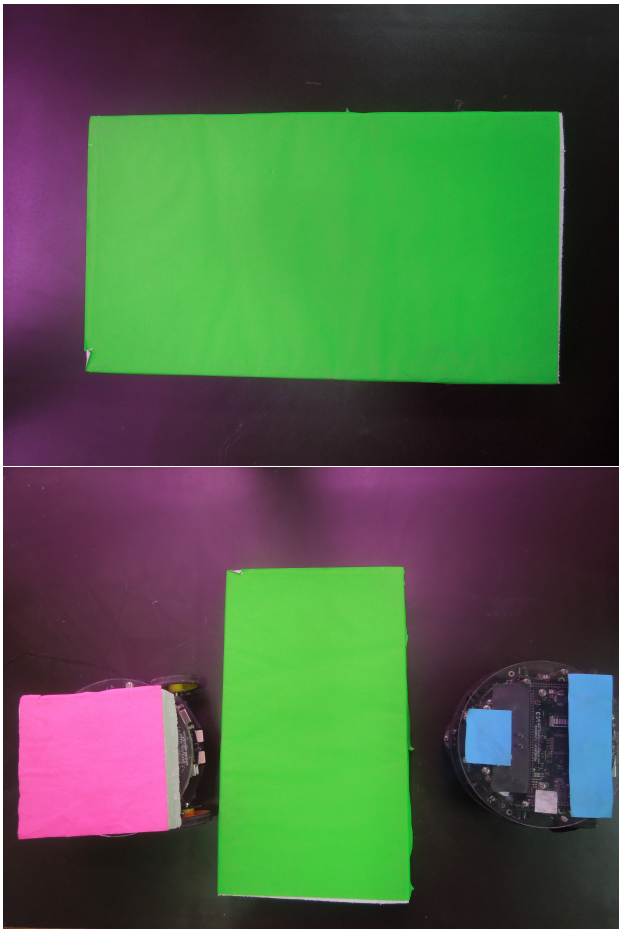
Slave Bot:

Slave Bot has two markers small rectangle in front and bigger rectangle in rear. To find the orientation of slave bot we use small one as a reference. Slave bot is shown in figure.

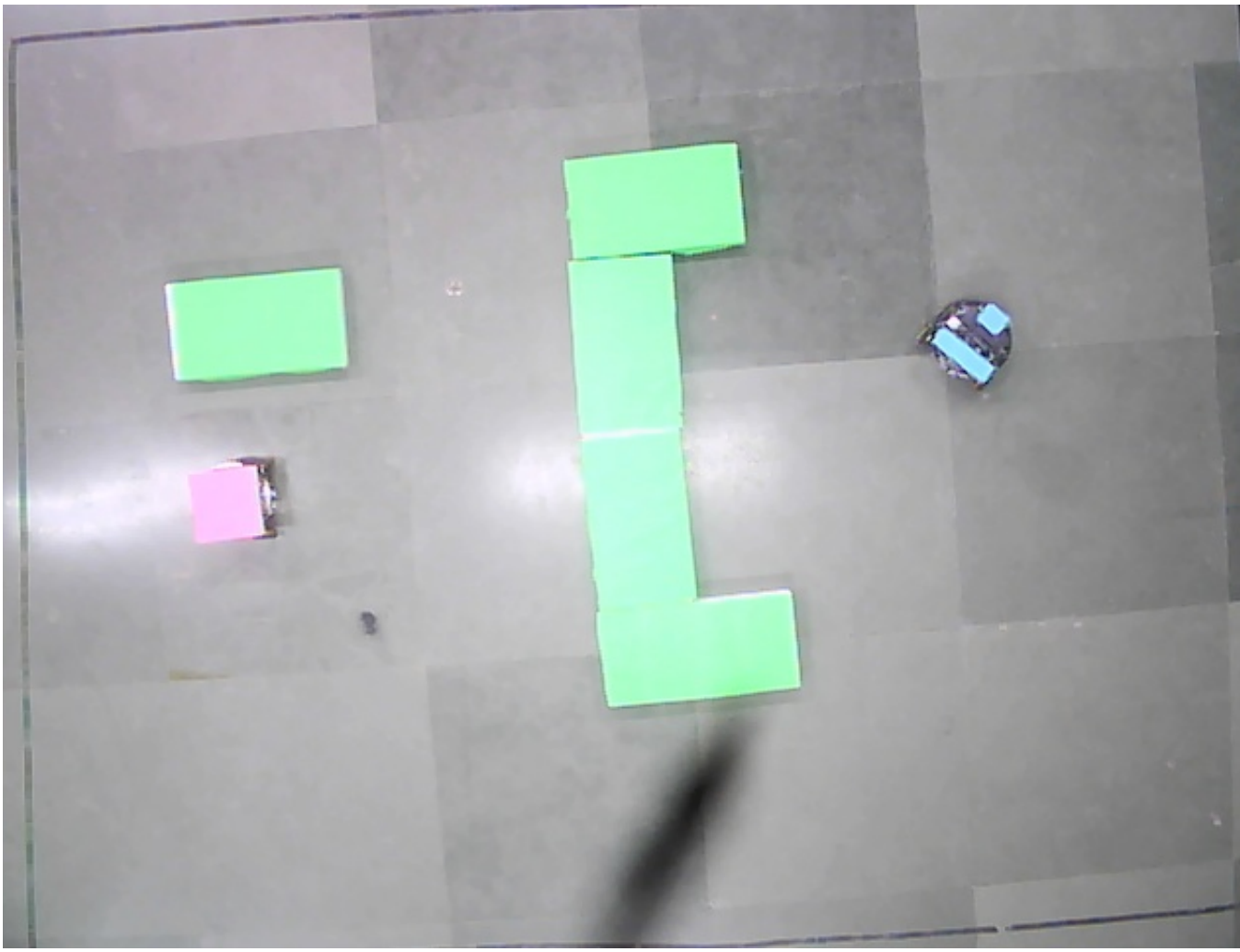


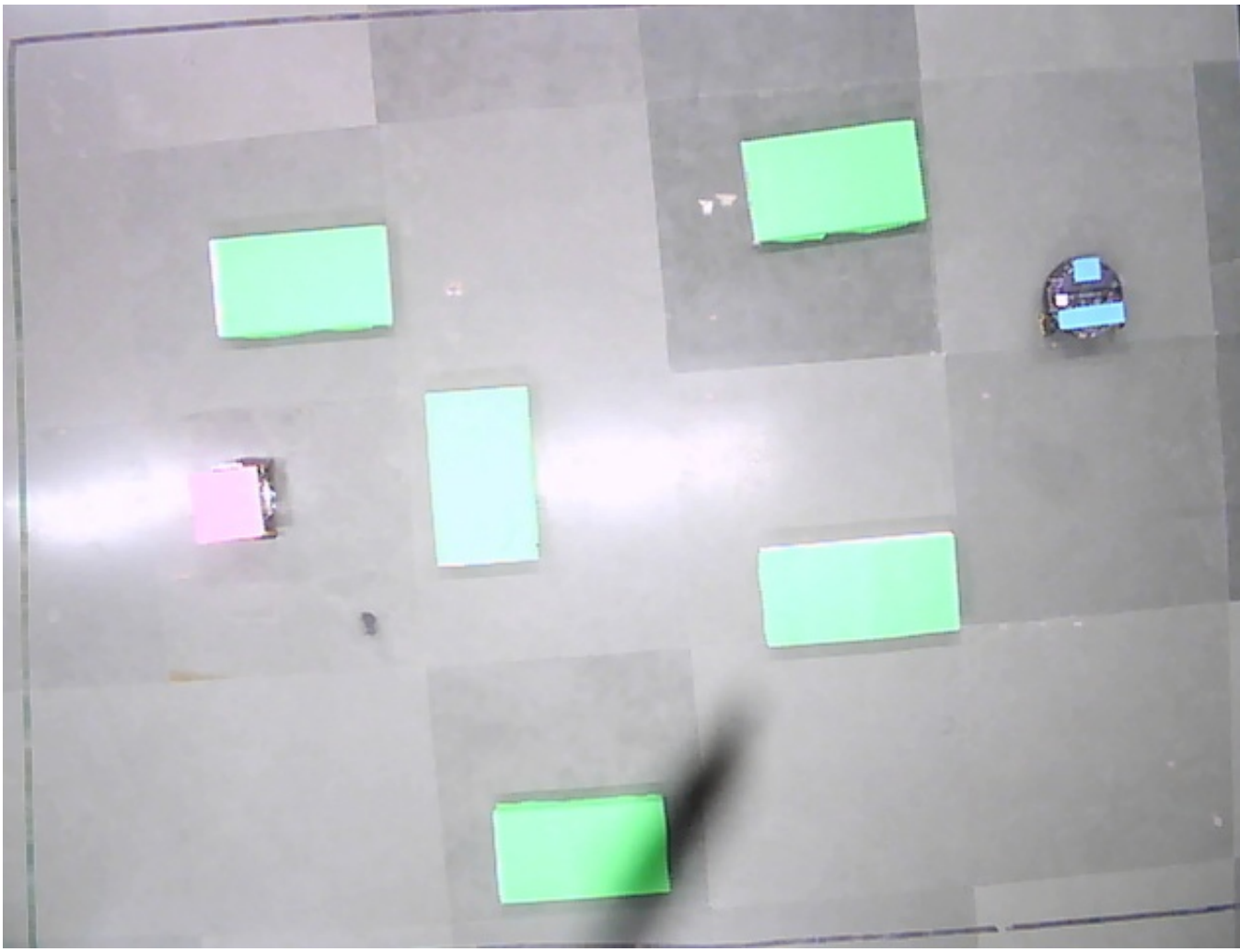
Walls:

To simulate real environments thermocol of green color is used as walls.



Setup of Arena to simulate real environment is as shown in figure.





Algorithm

- First we fixed a web cam on the roof at least 8 feet height and created a boundary which is covered by webcam. Now webcam is initialized to capture the video of the floor,for this OpenCV video capture function is used. Then we read frames from video at every interval of 1/80 sec so that we can get 10-12 frames per second for image processing.
- The video captured is in RGB format,so for greater processing RGB image is converted into HSV format.
- Then hsv image is send to the bot position and destination function where Master and slave Bot markers are masked and centroid of contours of those markers is stored for further use.
- Now for detection of walls on floor hsv image is send to the wall detection function where walls are masked and contours of walls is found.
- To be compatible with grid map based path morphological operation of closing following with dilation is done on walls. So the size of walls increased to its true size.
- Now to create a grid map across those walls this dilated image is send to grid map of walls function.
- For this dilated image is divided into grids of 30*30. And center of every grid is checked if it is white then 1 is placed at respective place of grid matrix of 30*30 else its 0 and boundaries of matrix is default 1 to be in safer side.
- Now a bot route function is called which returns the path to the destination point having walls in it.
- To find shortest path to the destination heapq algorithm is used.
- In heapq all the path to the destination are stored in heap and shortest path is returned on the basis of shortest length.
- As in video processing path is created at every instant so we update new path only if present path length is less than the last path.
- To show the path following we call route draw function which draws path from the slave bot to destination on frames which can be seen as a reference on screen.
- After it bot traverse function is called which handles the slave bot movements, where orientation function is called.
- As the returned path is links of coordinates of grids. So for movements of path angle between the lines of slave bots rear marker to route paths first coordinate and between slave bot front and rear markers is taken using the mathematical function with the help of slope of both line.
- As this angle is fed to PID controller as error input which process this value and return correction.
- If correction is negative then bot moves rights if its positive then left.
- To send these values to the slave bot serial communication via xbee is used.
- If pid correction approaches zero then slave bots moves forward.
- So if slave bot reaches to cell then coordinates of next cell from path is updated to follow so in that way local destination point is followed to reach global destination.
- If bot reaches near to final destination then stop.

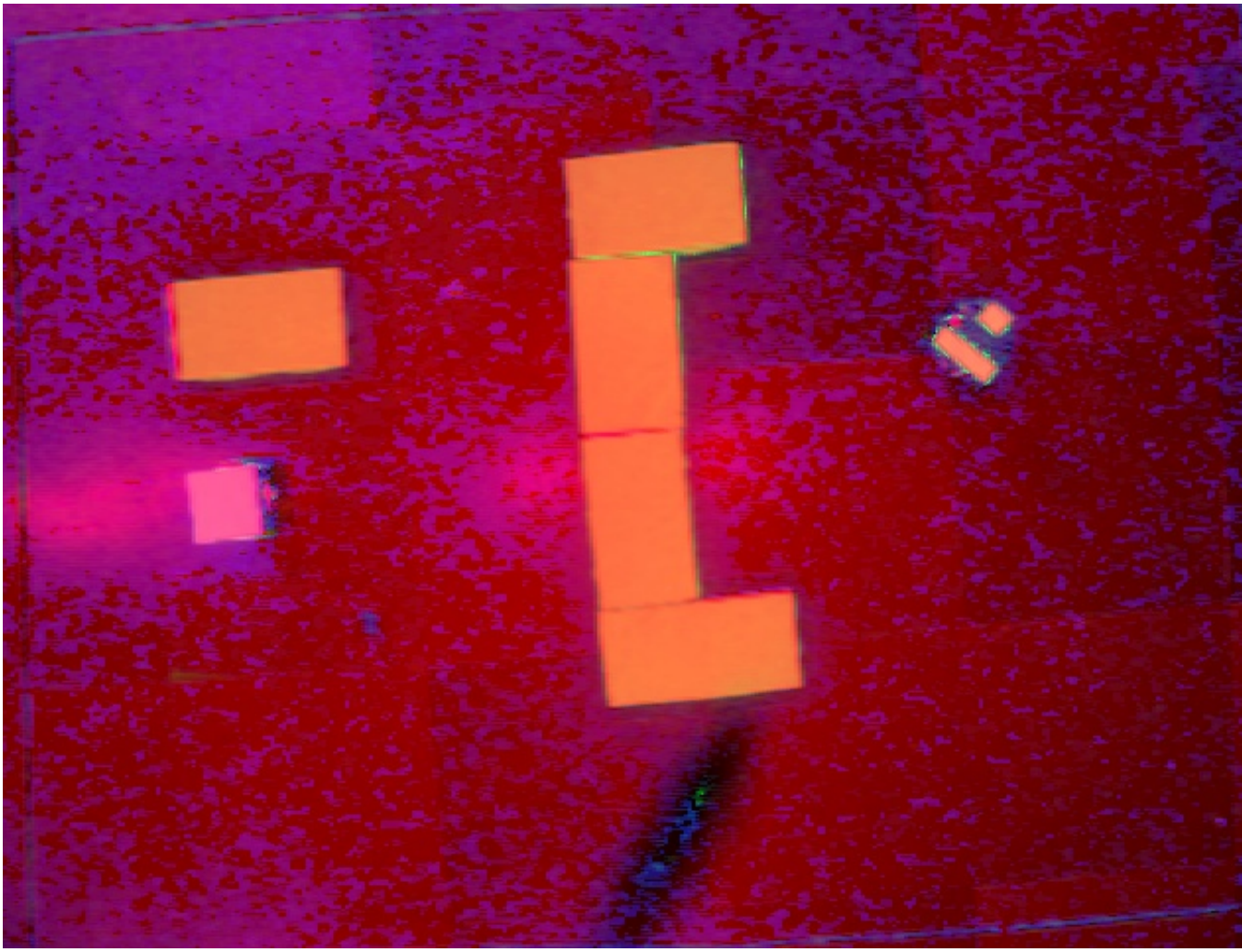
Steps involved in Image Processing

- Capturing Video of the Arena from external web cam situated on top of Arena.
- Markers placed on Master and slave firebird V
- Sky blue color small rectangle size shows the front of slave and Bigger one for rear end of slave firebird
- Pink Marker for Master Firebird V.
- Walls are defined by green color.
- Centroid of contours of masked Master, slave and walls are stored.
- Frames from video is converted in to grid map.
- Walls are mapped to the grid matrix by denoting 1 if there is wall else 0. Boundaries are 1 in default.
- Shortest path is find using heapq algorithm for maze solving on grid.
- Angle between Slave markers and path is fed to the PID controller to get correct speed of Motors.
- PID controlled commands are given to the motor to follow the path.
- Check the slave rear markers if it has reached near the destination then stop.

Implementation

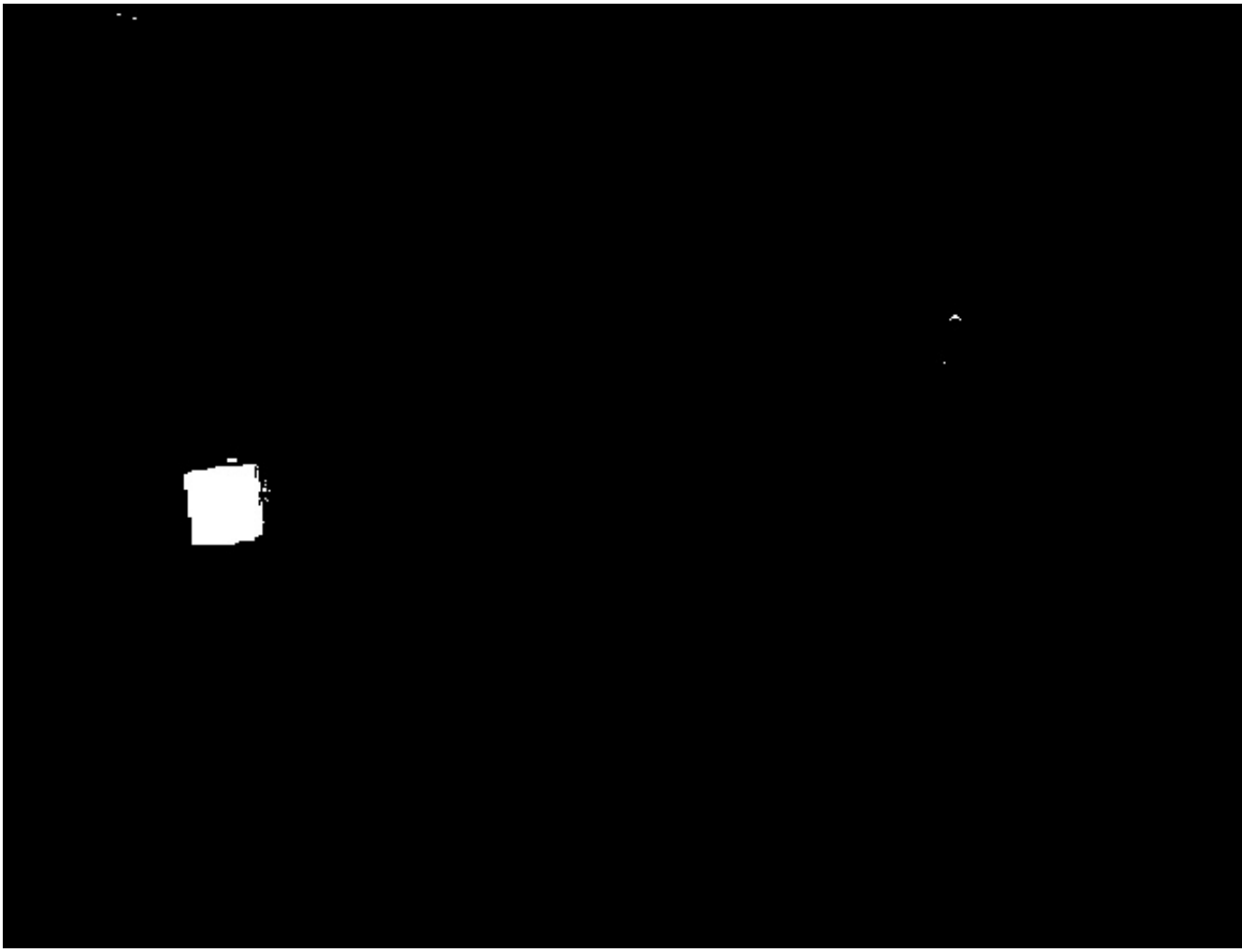
HSV Format:

After capturing frame from video it is converted into HSV. Which is shown in figure



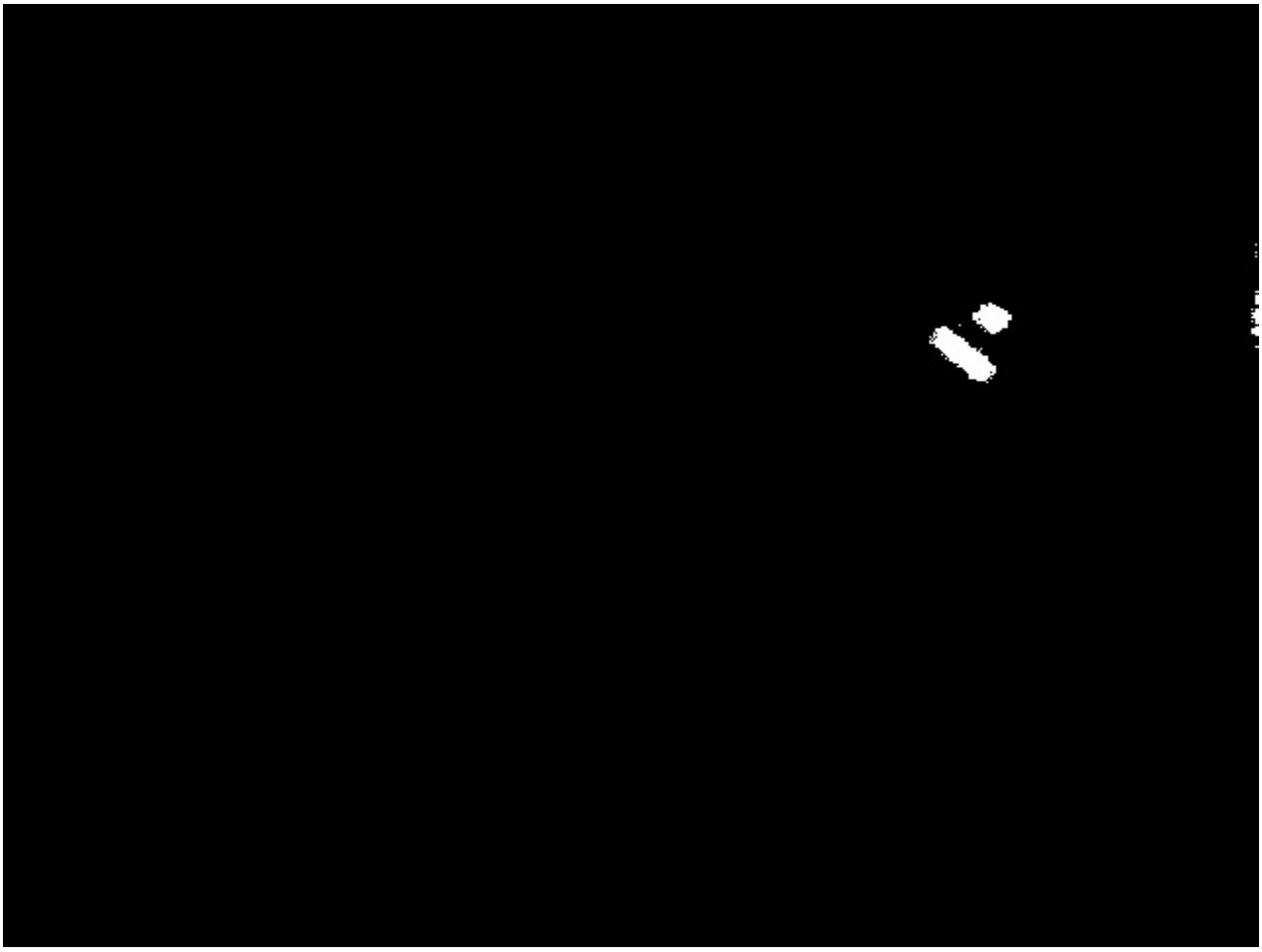
Destination masking:

To locate destination point or Master Bot, masking of pink color is done on hsv image of frames captured by video as shown in figure.



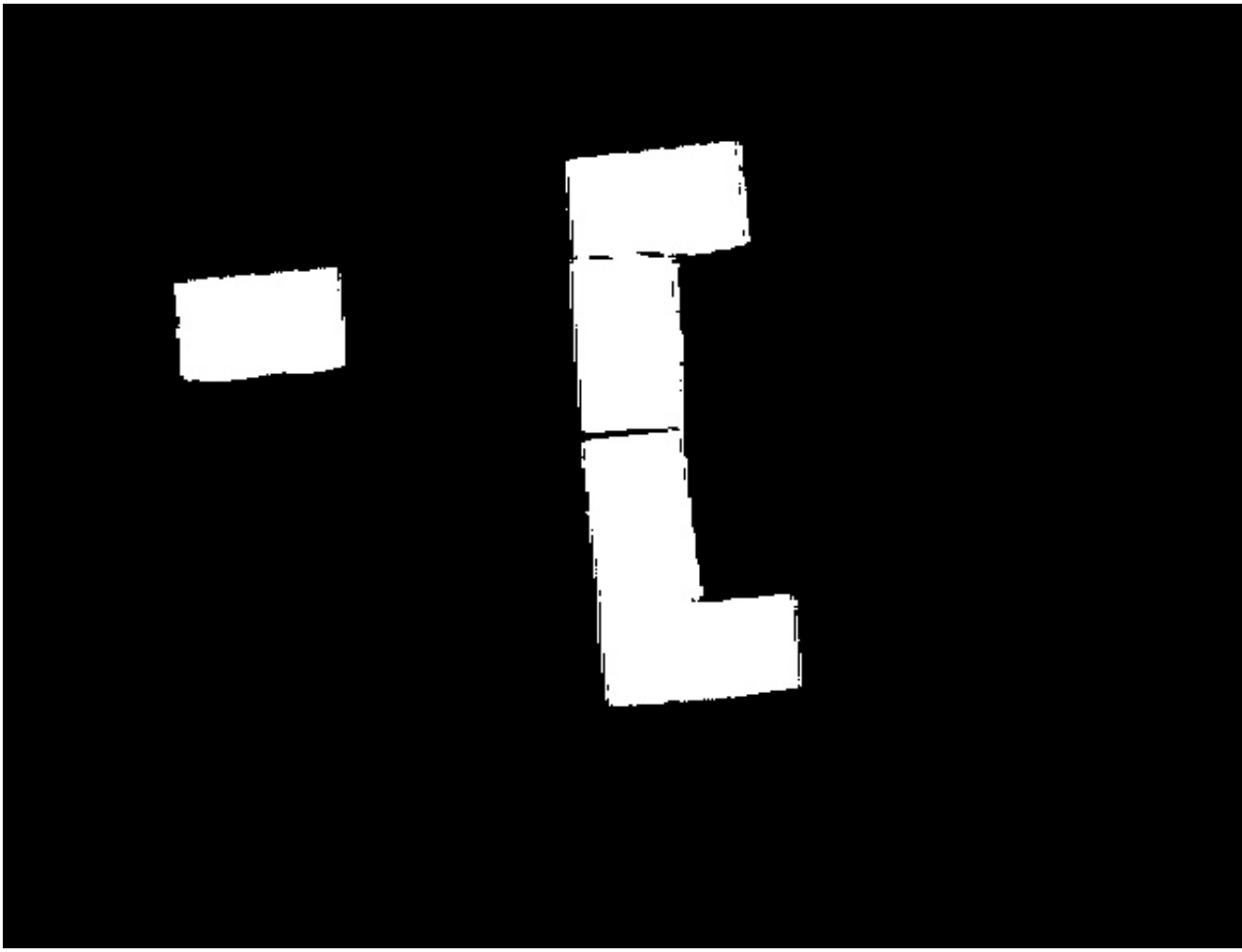
Source masking:

To locate source or slave bot markers are masked. Slave bot markers are of sky blue color. Masked area is shown in figure.



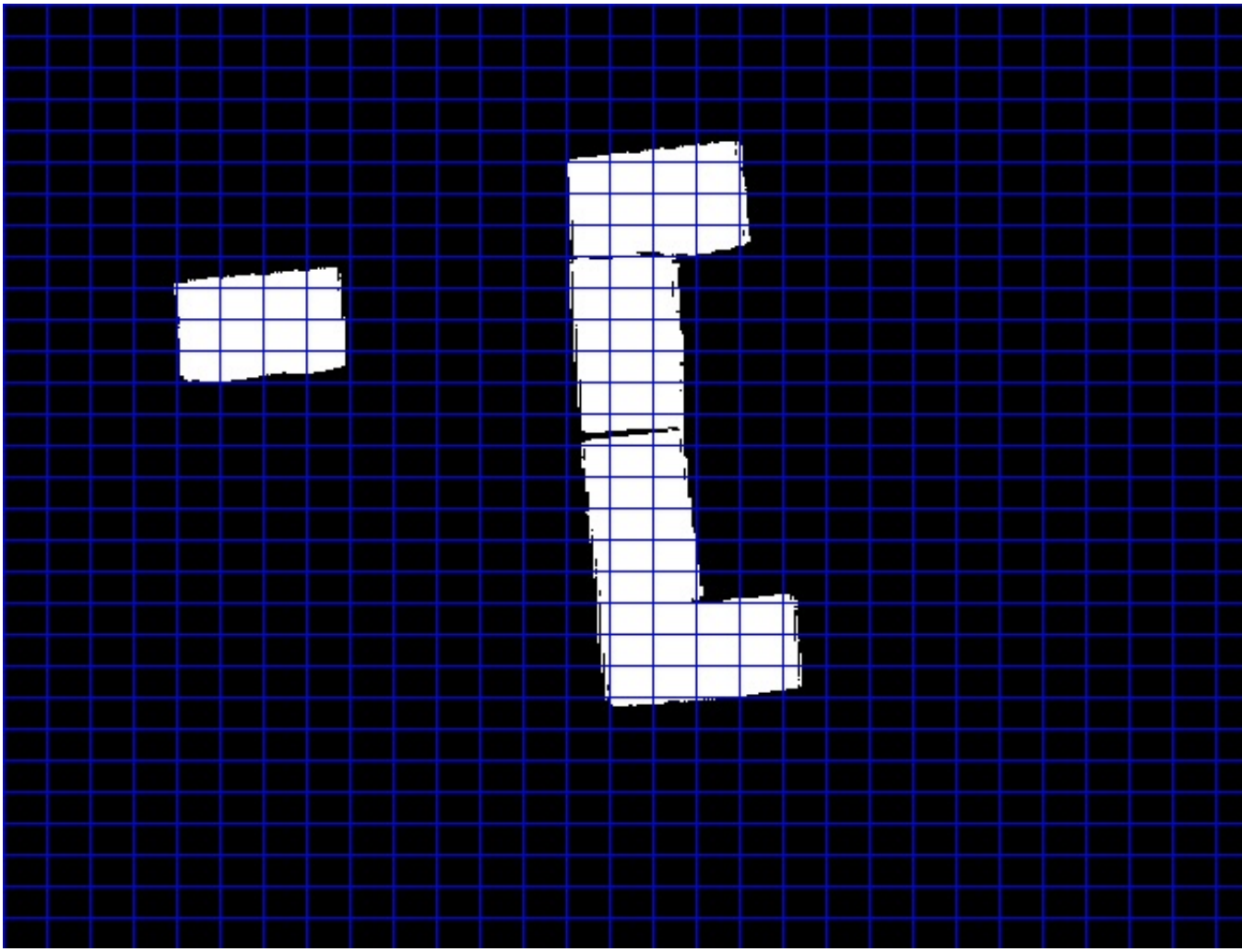
Wall Detection:

To detect walls masking of green color is done. Mask area of walls is given by figure. And then morphological operation is done closing followed by dilation to use for avoiding obstacle.



Grid Draw:

For path mapping image is converted into grid of 30*30. As you can see in figure



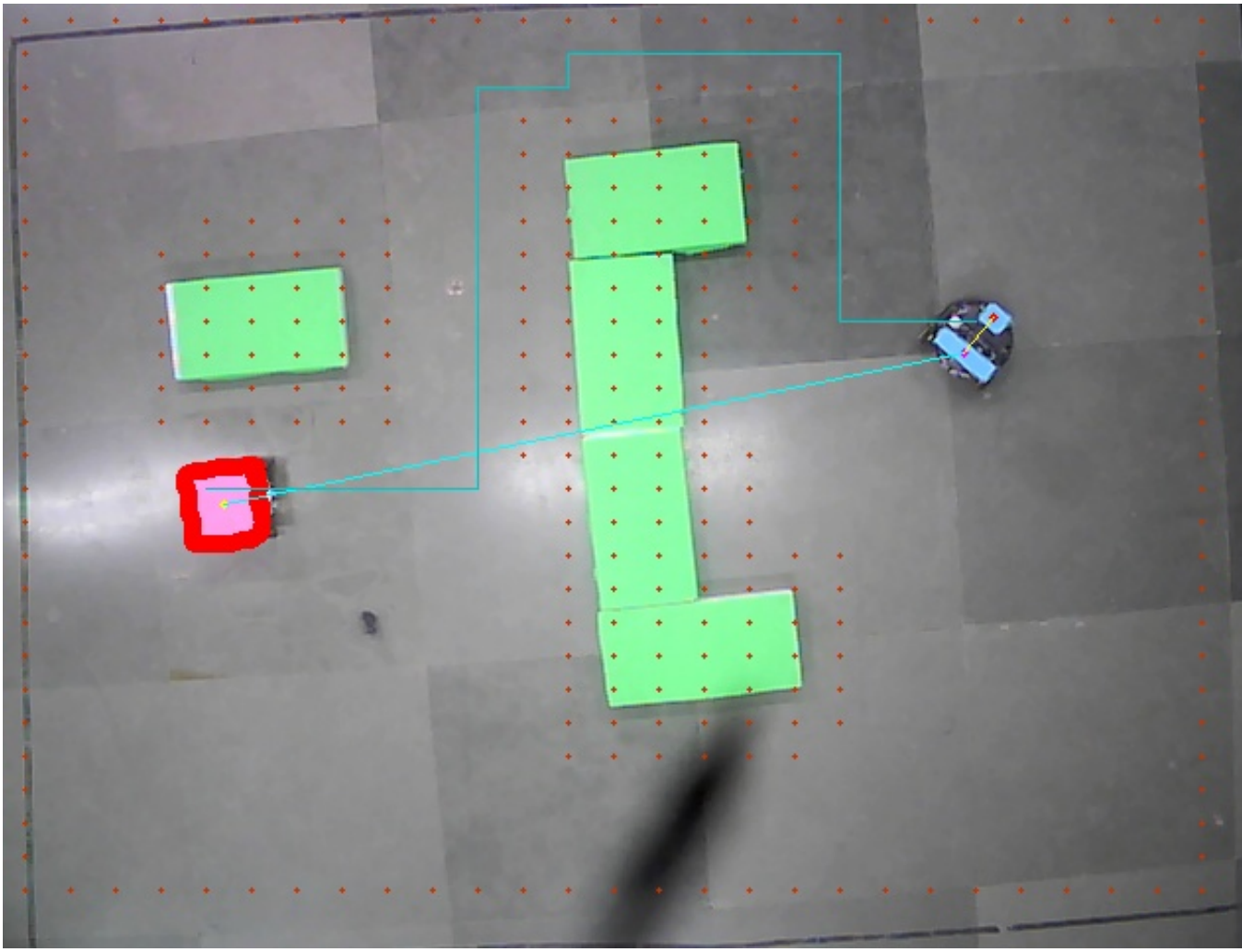
Grid map:

A grid map is also initialized to find the path from source to destination having obstacles on it. Grid map is a 30*30 matrix in which if center of grids from dilated image of walls is white then 1 is placed at respective place in matrix else it is 0. Boundaries are 1 as default just to be in safer side for path finding. Grid matrix across walls is shown here:

```
[[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 1 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

Mapping Path:

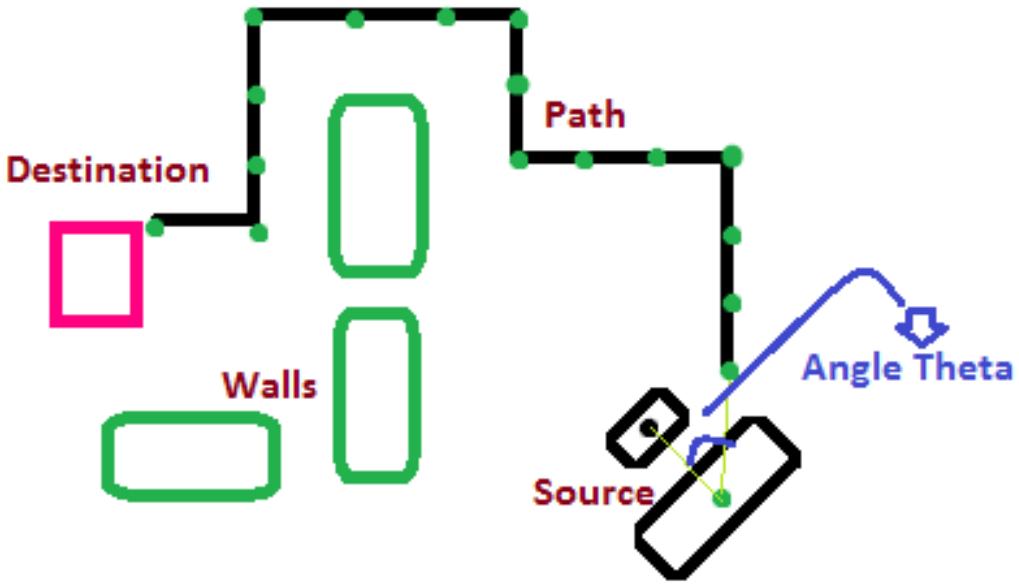
Path is mapped using the grid map created. To find the shortest path heapq algorithm is used. In heapq all path from source to destination is stored into the min heap and shortest path is returned on the basis of shortest length of path. A path mapped from source to destination avoiding obstacles are shown in figure



Red points denotes unreachable grid. A light blue line from start to end location to show where to reach. And light green color line following grids with avoiding obstacles is the path will slave bot will follow.

Motion of Bot:

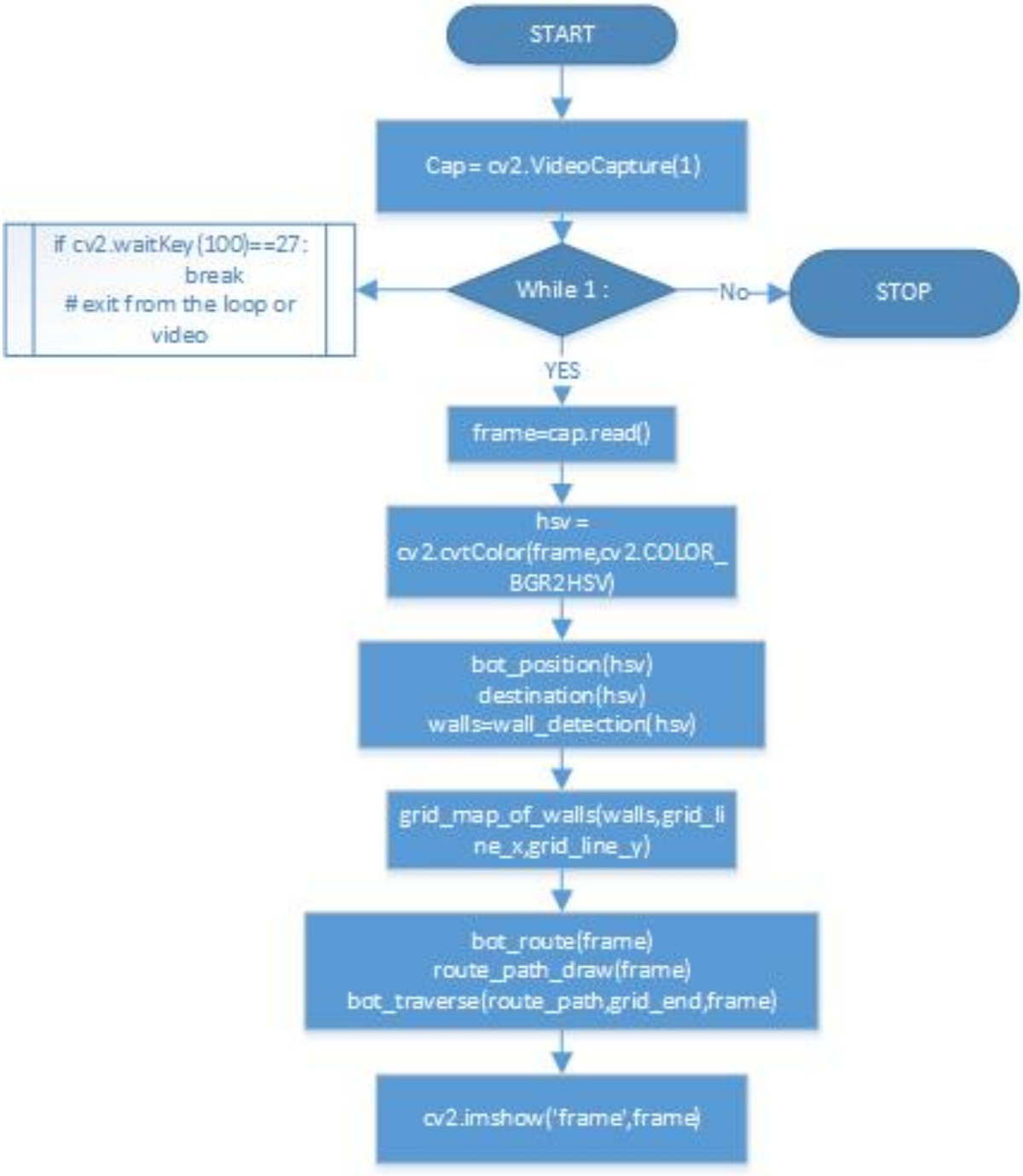
For movement of Bot angle theta shown in figure is fed to the PID controller.



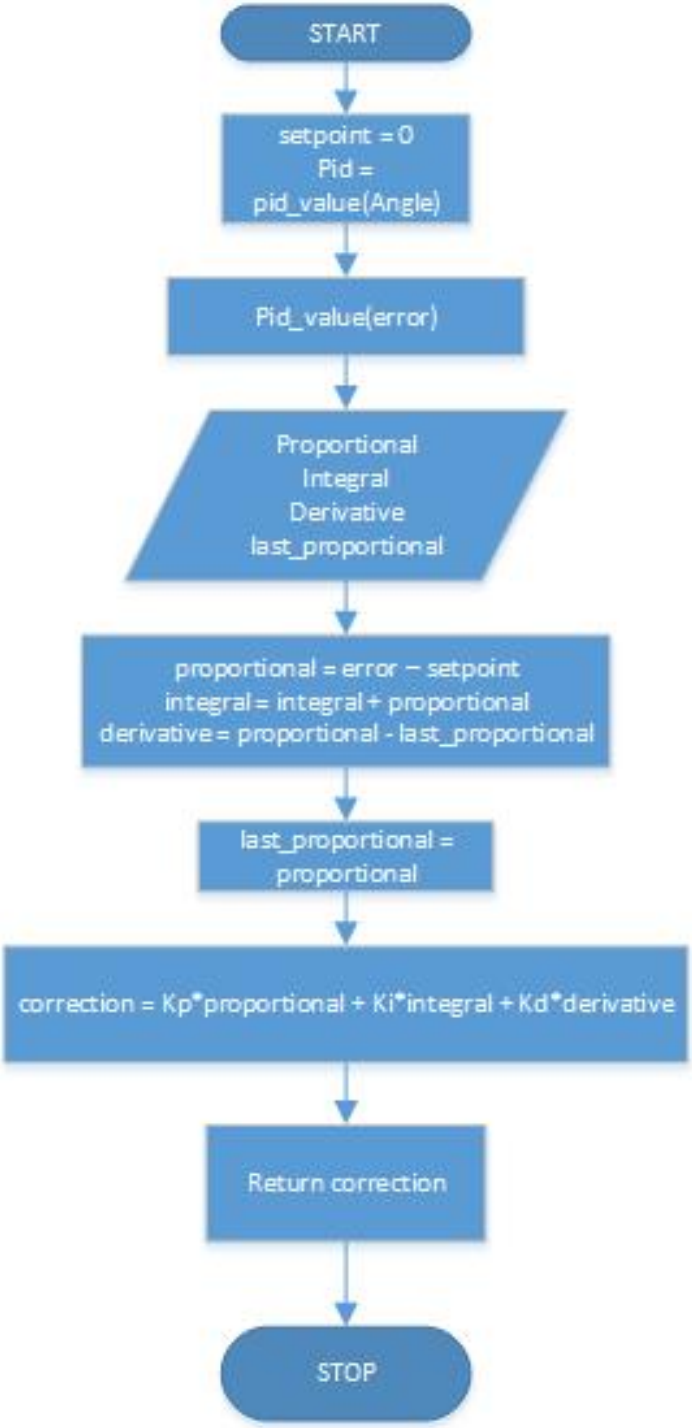
angle is negative if it is on left side of the path and positive if on right side of path. This results in a negative and positive error. So PID controller use these value and return a correction value which added to motor speed which helps to control the fire bird to be in path. To send these values to the firebird xbee communication is used.

Flow Chart

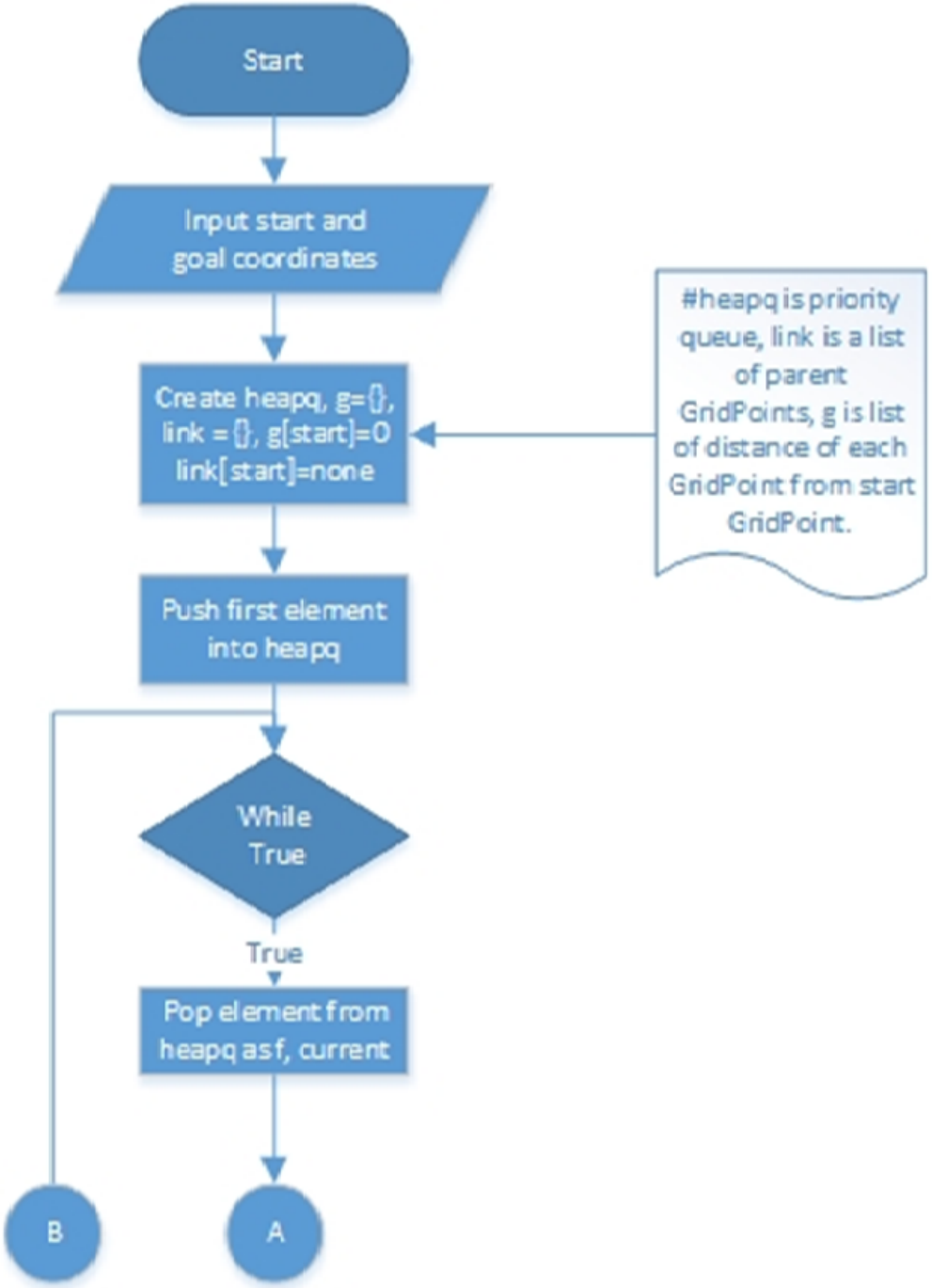
Overview of Algorithm:

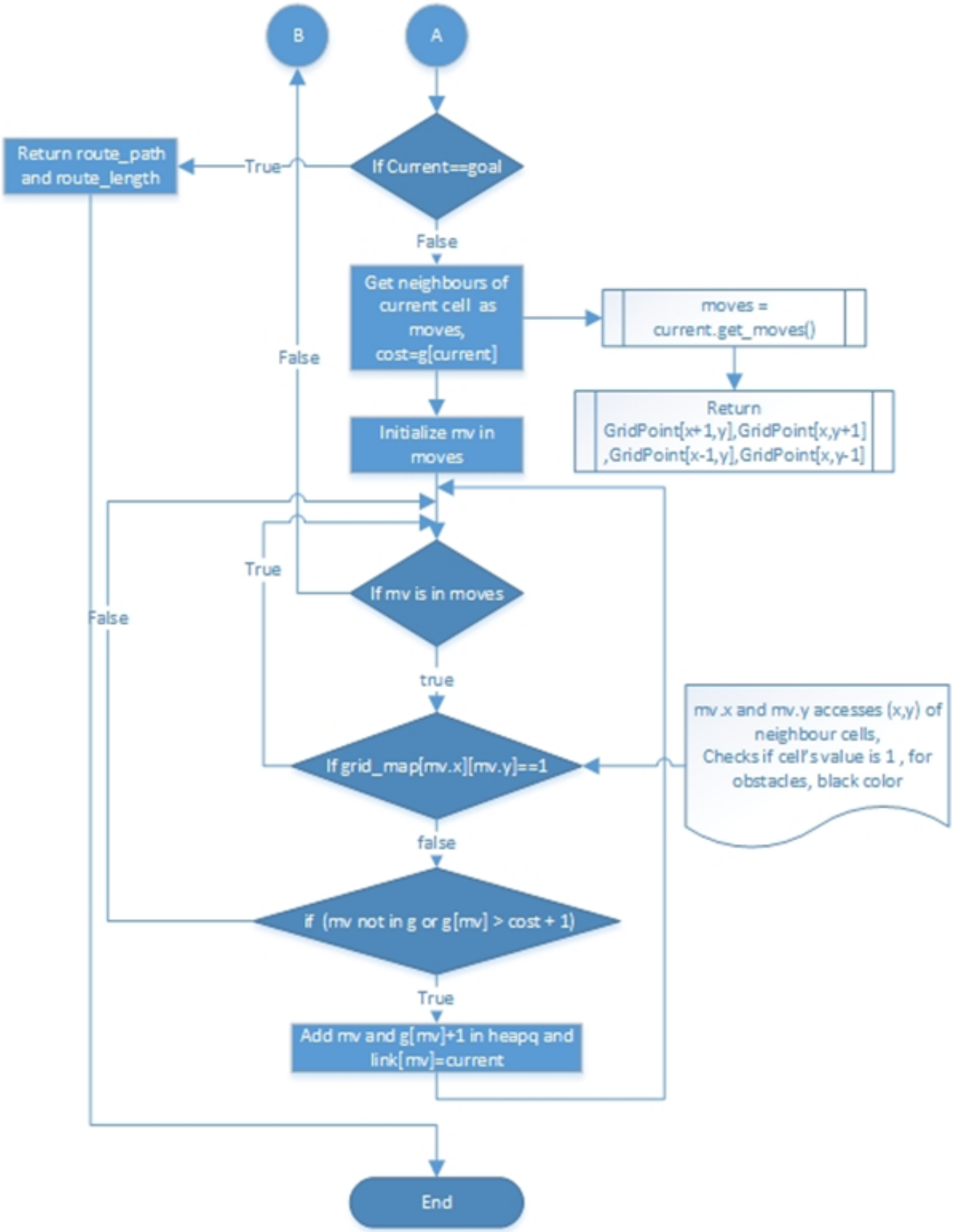


Working Of PID Controller:


START
setpoint = 0
Pid =
pid_value(Angle)
Pid_value(error)
Proportional
Integral
Derivative
last_proportional
proportional = error − setpoint
integral = integral + proportional
derivative = proportional - last_proportional
last_proportional =
proportional
correction = Kp*proportional + Ki*integral + Kd*derivative
Return correction
STOP

Finding Shortest Path:




B
A
If Current==goal
True
Return route_path and route_length
False
Get neighbours of current cell as moves, cost=g[current]
moves = current.get_moves()
Return GridPoint[x+1,y],GridPoint[x,y+1] ,GridPoint[x-1,y],GridPoint[x,y-1]
Initialize mv in moves
False
True
If mv is in moves
False
true
mv.x and mv.y accesses (x,y) of neighbour cells, Checks if cell's value is 1 , for obstacles, black color
If grid_map[mv.x][mv.y]==1
false
if (mv not in g or g[mv] > cost + 1)
True
Add mv and g[mv]+1 in heapq and link[mv]=current
End

Results and Discussion

- Using image processing we are able to control the slave bot to follow master bot with avoiding obstacles.
- For Masking different color we used hsv format of color.
- We used grid map based path planning which gives problem in returning path which are near to walls.
- To overcome with those problem we tried to dilate the walls. But it is not a good practice.
- We can by pass this by planning a path on the basis of local path then global path based planning.
- PID controller for controlling bot using image processing is working fine.

Problems Faced

During the project we faced following problem.

- Masking of color is toughest challenge. To handle it we created a hsv track-bar tool where we can vary the HSV range individually to mask a color.
- Choosing color combination for markers as lighting condition giving problem.
- To find the orientation of path respective to path at every instant.
- Mapping a path at every instant results in different source grid point. As there are more than one shortest path to the destination on the basis of grid map.

References

- http://www.coursera.org/course/conrob
- http://www.robotics.in/tutorials/categ/avr/pid
- https://www.docs.python.org
- https://www.docs.opencv.org
- https://stackoverflow.com/