

INTER IIT TECH MEET'21

**DRDO DRGE'S VISION BASED
OBSTACLE AVOIDANCE DRONE**

**BY:
TEAM 18**



Problem

The Problem Statement is to design an autonomous drone that navigates in a complex static environment by avoiding any collision with the on-field obstacles and reaching the target destination after its correct detection.

**ADDITIONAL
SOFTWARES
INSTALLED FOR
PERFORMING THE
SIMULATION**

ARDUPILOT

MAVROS

DRONEKIT SITL

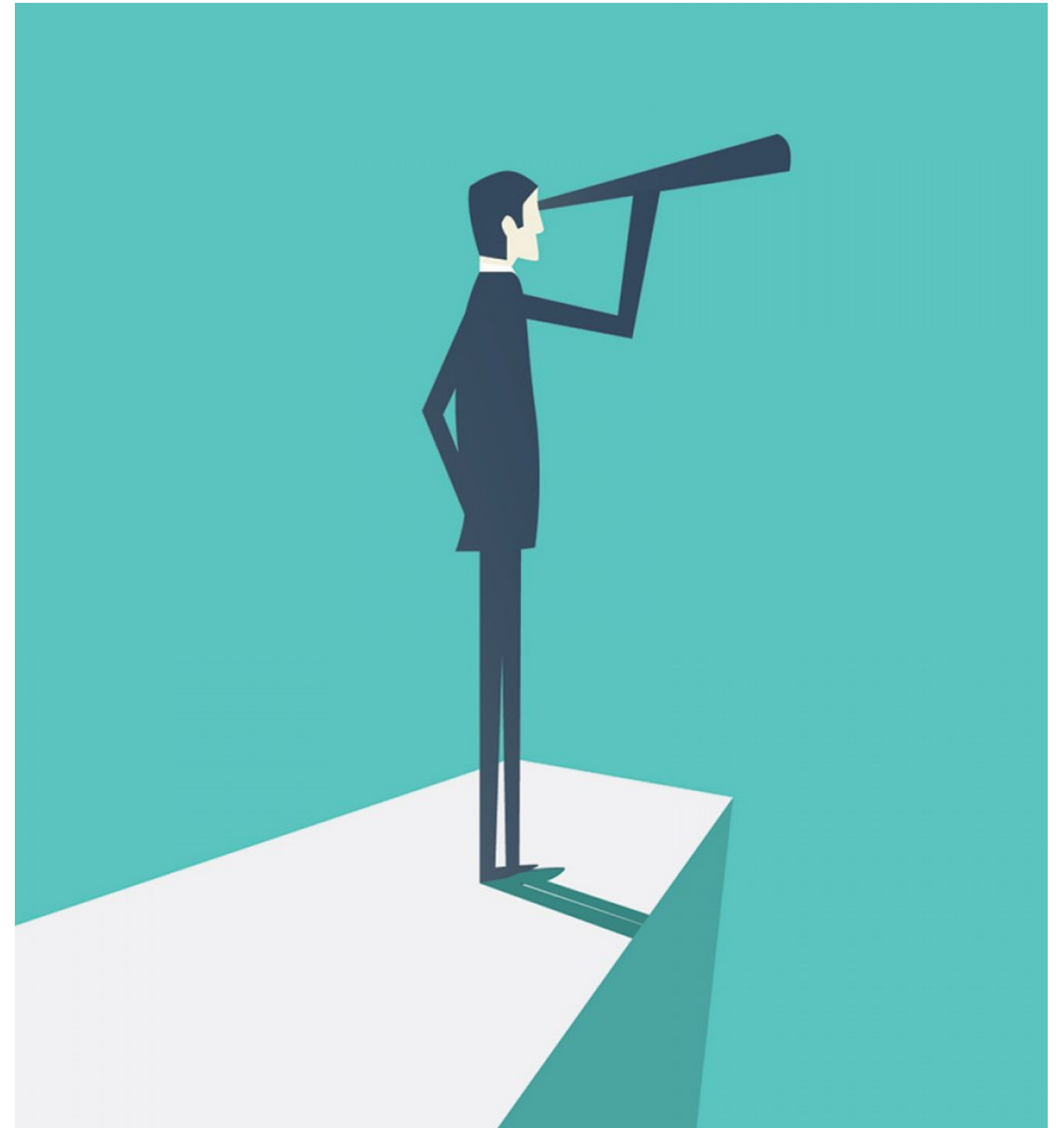
Task Overview

We Have completed this task in 3 steps:

Controlling Motion of Drone

Guiding Path of the Drone

Marker Detection to Land



Controlling Motion of the Drone

This is a separate script (executor.py) which takes the drone at its desired location (the location which is provided by controller script).

Our executor script connects with the drone at its local server and checks the arm status of drone and if the drone is initialized or not.

Guiding Path of Drone

This is main script which acts as the brain of the drone. It is responsible for detecting the nearby obstacles and also generates the shortest path to the local set point where there is no obstacle.

Solution Approach

GRID MAPPING



Initially we have divided the whole area into 100X100 grids where each grid was initialized to 0 indicating that grid to be obstacle free

OBSTACLE DETECTION



RGB Depth Camera was used to get point cloud of nearby obstacles. Point cloud contains the location (relative to drone) of each small particle, either they've integrated to form huge obstacle or they're acting as a single obstacle

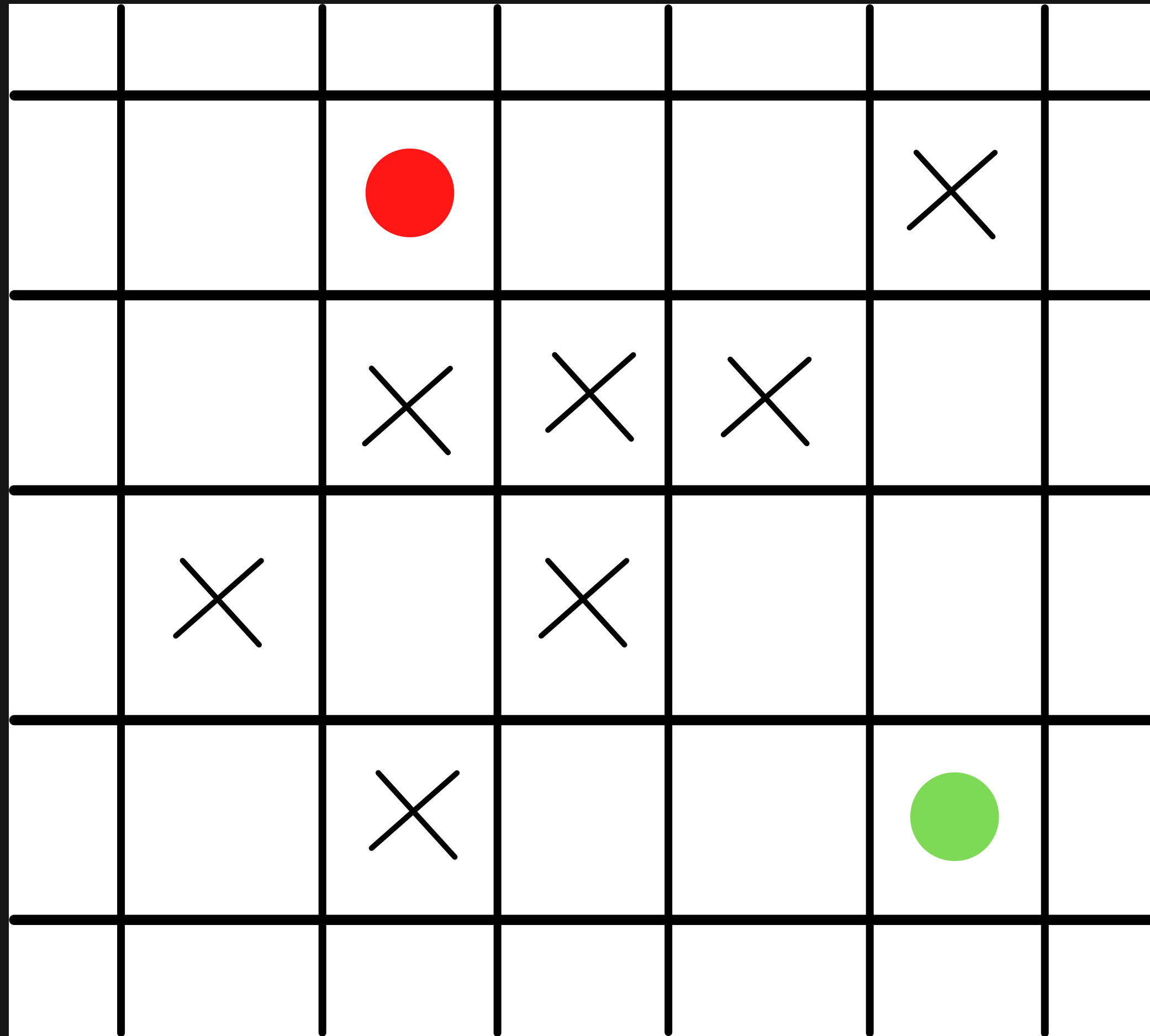
LOCATING OBSTACLES

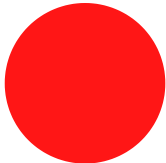










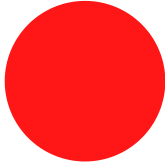
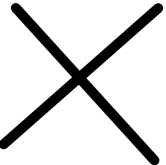







Once the location of obstacles was received, the corresponding grid is marked 1 indicating presence of obstacle.

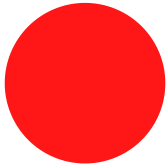
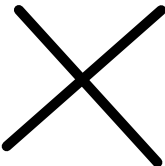


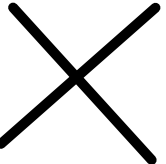

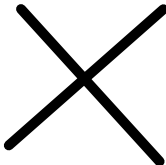
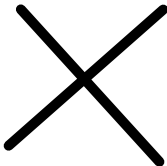

Path Planning

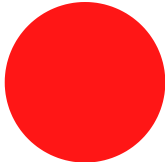

After detecting the nearby obstacles, it gets the series of coordinates which takes it to the destination , but if it gets obstacle again in its path, the series can be modified. basically each coordinate in its series act as checkpoint where, when drone reaches in a range of specific threshold of x and y, it again checks for the nearby obstacle.



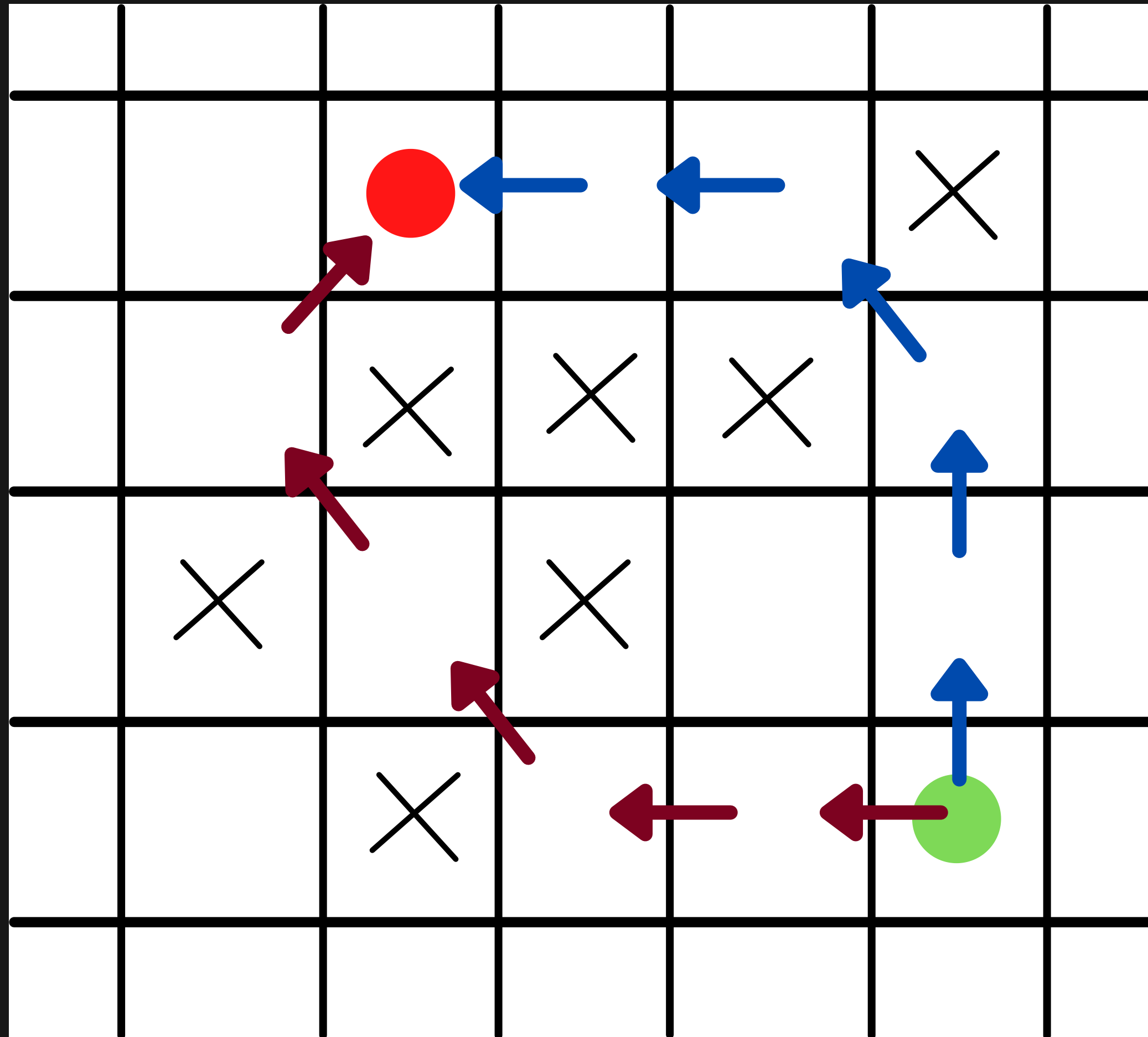
						
						
				1	1	1
				1		1
				1	1	1

						
					2	2
				1	1	1
			2	1		1
			2	1	1	1

				3		3
					2	2
		3		1	1	1
			2	1		1
		3	2	1	1	1

			4	4	4	4
			4	3	×	3
	4	×	×	×	2	2
	×	3	×	1	1	1
	4	×	2	1		1
	4	3	2	1	1	1

		5	4	4	4	4
5	5	5	4	3	×	3
5	4	×	×	×	2	2
5	×	3	×	1	1	1
5	4	×	2	1	0	1
5	4	3	2	1	1	1



SHORTEST POSSIBLE PATHS

Marker Detection

The detect_marker.py script contains the methods for detecting and identifying the correct Aruco marker for Landing. The image stream from the downward facing camera is processed using functions of Open CV library. In built functions of Open Cv for Aruco marker detection are used to detect the corners of the marker in the image and its identity. The distance of the drone from the marker are calculated from the formulae:

$\text{Error_x} = (\text{image width} - \text{center_x}) * \text{height of the drone} / \text{focal length}$

$\text{Error_y} = (\text{image width} - \text{center_y}) * \text{height of the drone} / \text{focal length}$

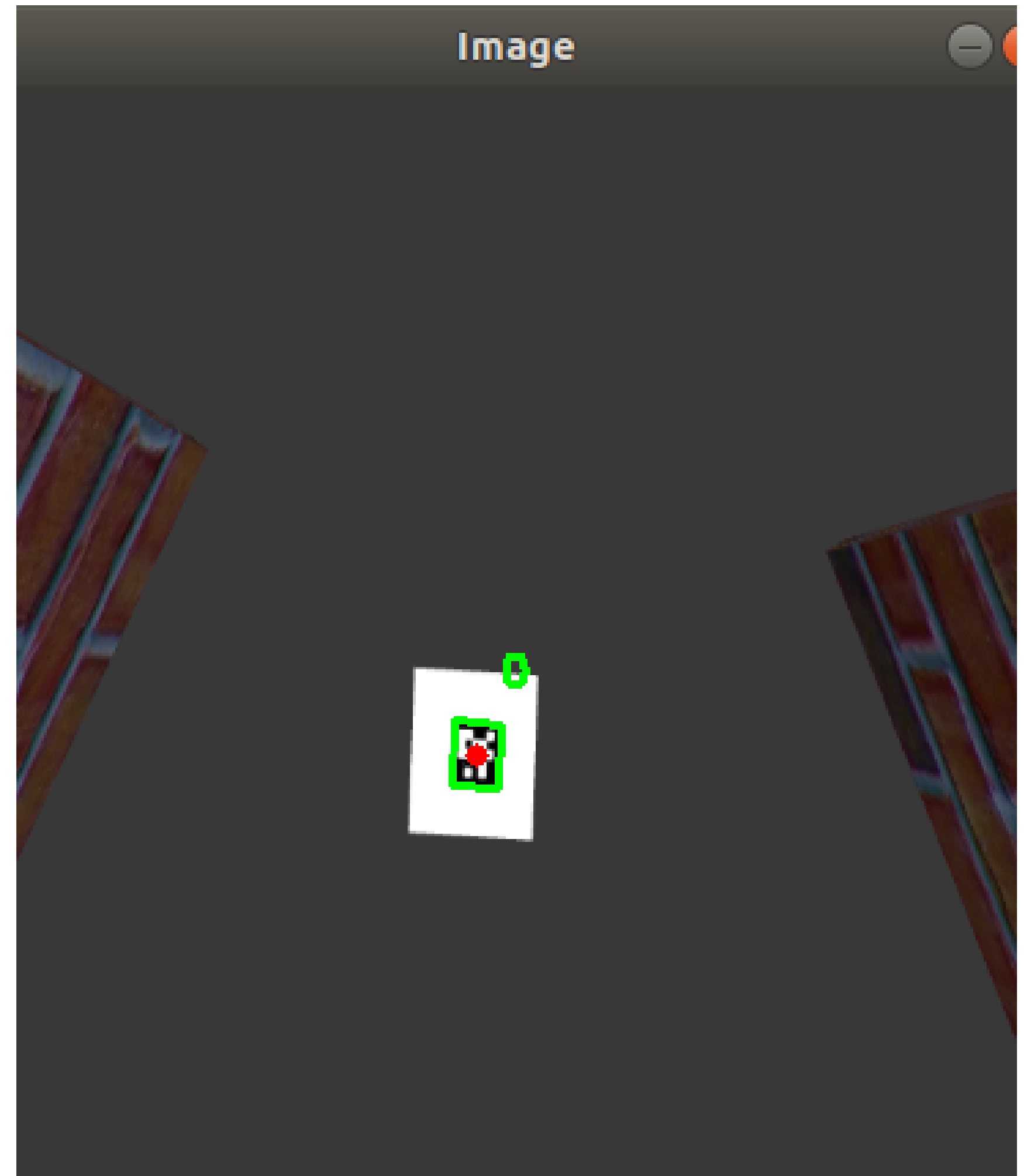
Focal length of the camera is calculated using the formula:

$$\text{Focal length} = (\text{image_width}/2)/[\tan(\text{HFOV}/2)]$$

where HFOV stands for Horizontal Field of View (1.04 for the current camera).

Green coloured rectangle is drawn on the marker and its id is also shown in the processed image.

Marker ID and Landing status is published on a topic named `/Aruco/message`



Failed Approaches

One failed approach was that we tried to work with the RtabMap to get the Real Time appearance as seen by the drone in order to implement the SLAM algorithm but we were unable to get proper real time appearance and hence we stood with our own algorithm.

Possible Improvements

We can try to work with the drone angles (roll, pitch and yaw) more precisely and in order to have more directional motion rather than random motion. Having proper Yaw we can map the obstacles more precisely and can proceed with more accuracy avoiding the obstacles.

Our Team



KASINA JYOTHI SWAROOP

ANIKET RAJ

AYUSH TRIPATHI

MRINAL PATHAK

AYUSH JHA

AMAN AGRAWAL

ALOK KUMAR

KARTIK SAINI

THANK YOU

ANY QUESTIONS?