D.R.D.O. SASE's UAV Fleet Challenge

Mid-Term Evaluation Report, IIT Kharagpur

Introduction

Unmanned Aerial Vehicles (UAVs) have a high potential to support search tasks in unstructured environments. Small, lightweight, agile UAVs, such as multirotor platforms, can incorporate many kinds of sensors that are suitable for detecting the objects of interest in cluttered outdoor areas. However, due to their limited endurance, moderate computing power, and imperfect sensing, mini-UAVs should work in groups using **swarm coordination algorithms** to perform tasks in a scalable, reliable and robust manner.

The solution was developed, keeping in mind two crucial factors: **time** and **cost** while obtaining the most accurate results. A highly scalable and intuitive solution was developed, which inherits a lot from how humans understand a scene. The drones map probable regions of interest and then plan their paths accordingly. The complete strategy has been designed to prevent any communication loss between the UAV's in the swarm, thus ensuring the uninterrupted transfer of information. The backbone of this solution is our robust search and detection algorithm as well as a simplistic hardware design, which optimises cost, ensures proper sensor positions, and has additional safety features like shock-absorbing structure.

Design Approach for UAVs

Various parameters, such as economic feasibility, maximum stability, and high power to weight ratio, have been taken into consideration while designing the UAV for optimum performance. In an outdoor environment, it is of utmost priority to stabilize the drone against the strong winds. The stability of the multirotor increases with an increase in size. However, it is accompanied by a steep increase in the cost of compatible accessories.

The frame size of 500mm has been selected to have the right balance between cost and functionality. The 10-inch diameter, 4.5-inch pitch propellers, and EMAX MT2216 810KV motors have been used to gain a **thrust to weight ratio of 2**. The flight time for this configuration with a take-off weight of 2.3 kg and 4S 8000mAh battery was calculated to be nearly **19 minutes**, which is an optimum balance between the flight time and cost of batteries up to the capacity of 12000mAh.

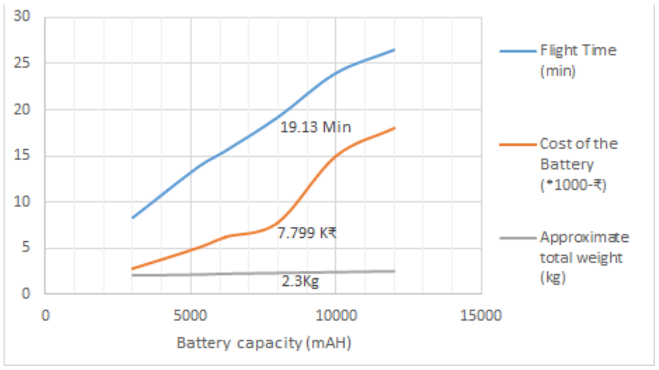


Fig.1 Comparison of 4S Batteries

Since the arena is small, pinpointing the precise locations of the target boxes has been achieved using a high-precision GPS module. For accurate and real-time detection algorithms, we needed an on-board processor to perform heavy computation. The most economical solution, was Odroid XU4, with an ARM Cortex-A15 quad-core processor running at 2.0 GHz and Cortex-A7 quad-core CPUs with a 2GB DDR3 RAM. The camera used is SJ4000 with a field of view of 170 degrees and resolution of 1280 x 720 pixels at 30 frames per second, to achieve wide range in much lesser height and fast data collection, without compromising on the quality. Also, flying at a lower height minimises the effect of environmental factors such as wind and fog.

The flight controller used is Pixhawk 2.4.8 flashed with Ardupilot firmware stack which has been experimentally determined to be more robust than other available firmware. Considering the economic feasibility of the solution, the estimated cost of each drone was only **INR 46k** which is far less than that of any commercially available drone with the same specifications. Since this drone is self-assembled, in case of any damage to its parts, the user can easily replace the part, which would decrease its cost and increase its reliability in the long run. The final components specification is as follows:

Frame Size	*500mm*
Battery	*8000 mAh 4S*
Calculated Weight	*2300 gm*
GPS	*HERE 2*
Flight Controller	*Pixhawk 2.4.8*
Motors	*EMAX MT2216 810KV*
ESC	*EMAX SIMON K 30A*
Transmitter	*FlySky FS-i6X*
Receiver	*FlySky FS-iA6B*
Processor	*Odroid XU4*
Camera	*SJ4000*
Propeller Size	*1045*
Wi-Fi Adapter	*802.11/b/g/n/ac compatible*

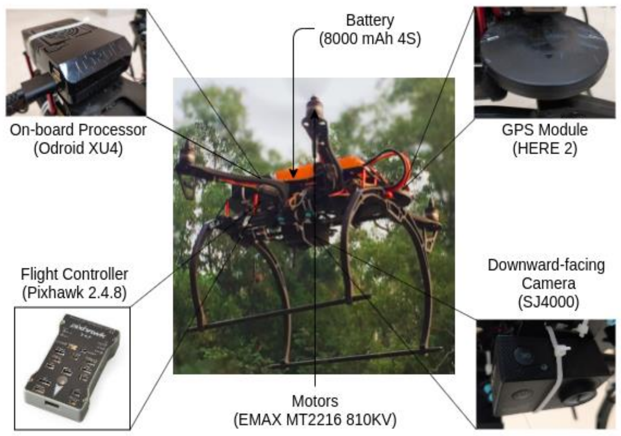

Battery (8000 mAh 4S)
On-board Processor (Odroid XU4)
GPS Module (HERE 2)
Flight Controller (Pixhawk 2.4.8)
Downward-facing Camera (SJ4000)
Motors (EMAX MT2216 810KV)

Fig. 2 Hardware Setup

Swarm Communication

Search and rescue drones need to work in conditions where they can remain detached from ground stations and still be able to communicate with the peer drones to coordinate and collaborate. A **decentralized communication** system allows the drones to communicate with each other, even in the absence of a centralized communication backbone. This has been realized using **Ad-hoc** wireless communication, which is based on *a* **peer-to-peer** connection. It is established and maintained without the need of any intermediate and supervising router or server.

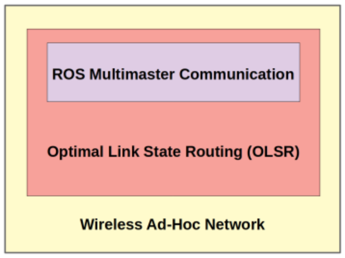

ROS Multimaster Communication
Optimal Link State Routing (OLSR)
Wireless Ad-Hoc Network

Fig. 3 Swarm feature stack

In missions like these, all drones must be able to exchange real-time information reliably via a resilient communication system that allows them to connect indirectly to distant drones that are beyond the **direct signal reach**, leveraging other drones in the vicinity. This is achieved by using **dynamic routing protocols** that are installed on all the drones to provide indirect communication links and dynamic routing information. Over this mesh network, the drones will keep broadcasting status messages that allow them to coordinate their activities. This is done by running an application on each drone that broadcasts the local status and fetches the status information of other drones.

Implementation and Working

a. Ad-Hoc communication

Each drone houses an Odroid with a wireless adapter setup in Wi-Fi Ad-Hoc mode. The wireless interfaces are then assigned the same Wireless channel, Broadcast SSIDs and subnet mask. The IP addresses are then assigned under the same subnet. Having the **same wireless channel** is an essential requirement. This can be identified by the Cell-IDs assumed by the interfaces. The **Cell-ID** acts as a unique identifier for the combined network. Once all the drones are inside the same cell, they can communicate with each other.

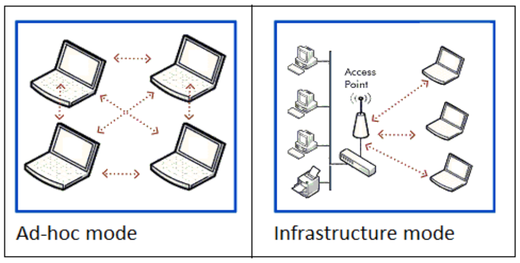


Fig. 4 Comparison between infrastructure and ad-hoc

b. Dynamic Routing using Optimal Link State Routing (OLSR)[#]

OLSR is a routing protocol specifically intended for Mobile Ad-Hoc Networks (MANETs). It allows us to create a mesh topology of drones within the ad-hoc network and provides **deterministic routing** information. The protocol is an optimization of the classical link state algorithm tailored to the requirements of a mobile wireless LAN. Once a network interface is initiated into OLSR routing, it can discover other OLSR enabled nodes that are present in the Ad-Hoc network.
In OLSR, link-state information *(topology control)* is generated only by nodes elected as **multi-point relays** (MPRs). At any given time, OLSR sticks to a specific topological structure, which is updated dynamically with time by the topology control as the drones keep moving. This updating of topology is only possible due to *topology control* of OLSR. There are other link-state routing protocols like OSPF and BATMAN. However, OLSR gains preference because it prevents wireless network flooding by using the optimal amount of topology control messages, and it is tailor-made considering the mobile nature of the network.

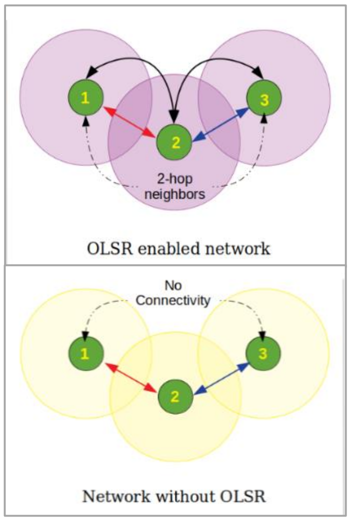


Fig. 5 OLSR Network

c. Application Level Communication

We need an application-level communication to allow applications on different drones to communicate. Decentralizing the ownership of the topics by having a master on each drone ensures that the topics are shared even when the 'central' drone becomes unavailable, and also reduces latency between the local applications. But ROS does not naturally support multiple masters so, we employ external methods to facilitate the exchange of contents published under the topics of different masters. ROS **multi-master-fkie*** is a readily available package that bridges the topics published under multiple masters. Using this package is more preferred as compared to the primitive method of creating direct UDP sockets, mainly due to the ease of implementation and scalability.

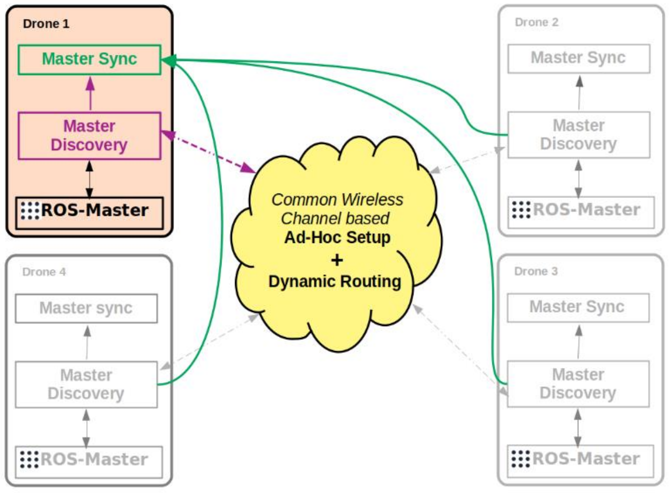


Fig. 6 ROS multi-master working with respect to Drone 1

Specifications and performance

1. Wireless adapter specifications: generic adapter with *802.11b/g/n/ac* support
2. Line-of-sight data rate (upto *100 metres*) minimum (*30kB/s*) and *upto 2MB/s*
3. Latency - worst case *100 ms* (nominally *15 ms*)

*ROS: Multimaster-fkie [http://wiki.ros.org/multimaster_fkie]
#RFC 3626, OLSR Protocol [https://tools.ietf.org/html/rfc3626]

Strategy and Planning

Search strategy plays a vital role in effective search of target positions. A simulation world was setup using ROS Gazebo and various strategies like Random walks, Information gain-based approach, pre-determined waypoints were tried and tested to determine the best approach for the given scenario. Following conclusions were made:

1. **Random walks**, though asymptotically complete, lead to many ineffective searches as they can closely pass from a nearby target position without detecting it. Besides, it leads to a lot of overlapping coverage. Its operation can only be justified in completely unknown environments and thus theoretically eliminated due to its inefficiencies in a constrained environment.
2. **Information gain-based** strategies are generally used for exploration in environments which are rich in features and the waypoints are generated in areas where the maximum amount of information may be inferred. Thus, the use of this strategy was inappropriate for the given scenario due to low availability of features in an open environment and high computation costs.
3. Different methods for the **pre-determined waypoint approach** were tried. Initially, trial runs were conducted wherein each of the UAV would be assigned a rectangular strip with a pre-determined U shaped trajectory (fig 8). The search time in this approach was highly dependent on the spawn location of the UAVs. This led to the idea of dividing the arena into four quadrants wherein each of the drones would be assigned an individual quadrant for search (fig 9). This was inefficient since the UAV was forced to search the entire quadrant from a lower height and hence increasing the search time.

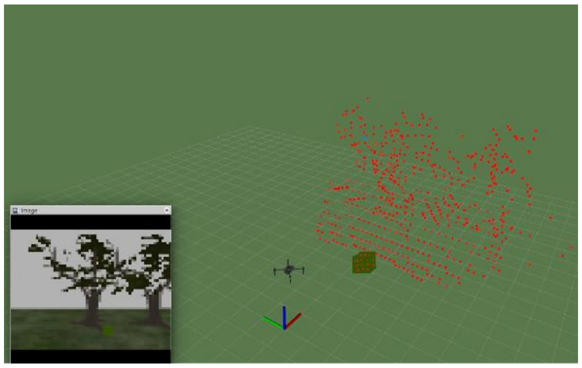


Fig. 7 Simulation World Setup

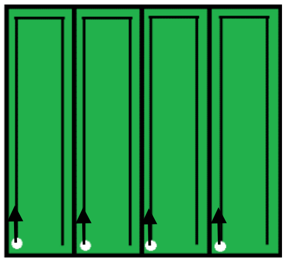


Fig. 8

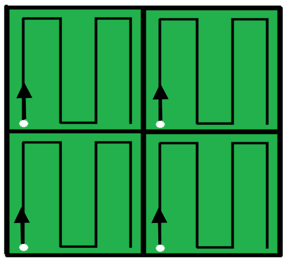


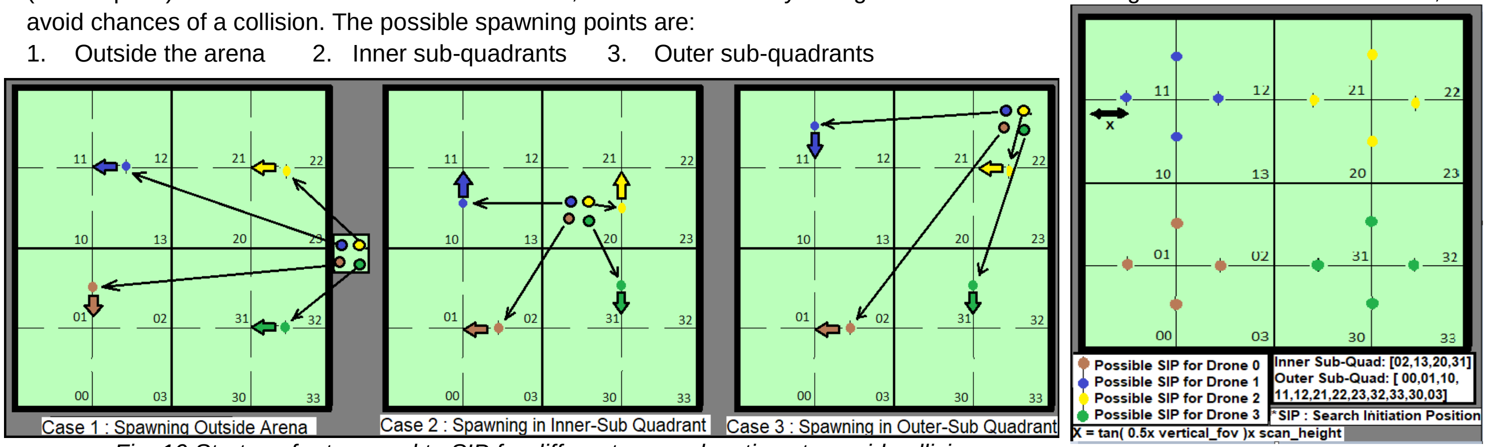
Fig. 9

After conducting multiple simulations, we came up with a combination of **Region of Interest (ROI) & quadrant based strategy** which was an intermediate solution that would justify the present problem statement and would also work in any general scenario. The proposed strategy is described below.

Allocation of Search Initiation Position (SIP)

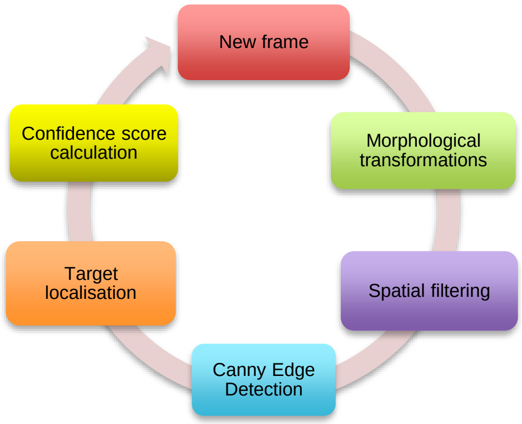
The arena has been divided into 4 equal quadrants. Each drone is assigned one of the 4 quadrants for the detection of the target. We have defined 4 possible SIPs for commencing the detection process in each of the quadrants. Each drone from the **spawning position** (takeoff point) travels to the closest of these 4 SIPs, while autonomously taking off in order of decreasing distance to the closest SIP, to avoid chances of a collision. The possible spawning points are:

1. Outside the arena 2. Inner sub-quadrants 3. Outer sub-quadrants


Case 1 : Spawning Outside Arena
Case 2 : Spawning in Inner-Sub Quadrant
Case 3 : Spawning in Outer-Sub Quadrant
Possible SIP for Drone 0
Possible SIP for Drone 1
Possible SIP for Drone 2
Possible SIP for Drone 3
Inner Sub-Quad: [02,13,20,31]
Outer Sub-Quad: [00,01,10, 11,12,21,22,23,32,33,30,03]
*SIP : Search Initiation Position
X = tan(0.5x vertical_fov)x scan_height
Fig. 10 Strategy for traversal to SIP for different spawn locations to avoid collision

Efficient detection of probable target positions

1. Upon reaching the respective SIP, the drone ascends to a height enough to have the field of view to cover the entire width of the quadrant. Next, they move towards the opposite side, **scanning** the arena for the presence of the target.
2. By virtue of the scanning height drones, the image stream obtained does not have any features of the grass. The only source of image gradients is cluttered objects and our targets. To make our approach even more robust and to suppress any remaining features of the grass, we apply various **morphological transformations** on the image.
3. Spatial filtering and canny edge detection are used for **feature extraction**.



4. The last section of our algorithm deals with **localising the targets** based on the features extracted. For this, the contour detection algorithm is used, which clusters these features into sets of continuous points which could only be interpreted as probable targets.
5. The algorithm is re-iterated for several frames. It assigns a **confidence score** to targets based on its shape colour and other properties. For decreasing false positives, targets below a certain threshold are not considered during path planning.

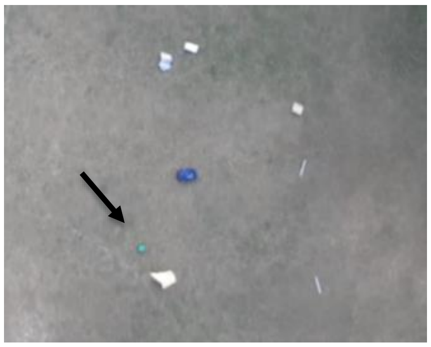


Fig. 11(a) Green box can be seen in the image(arrow pointed)

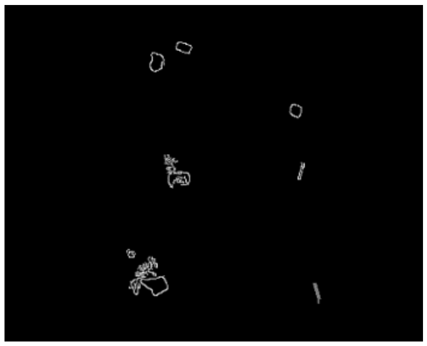


Fig. 11(b) Feature Map

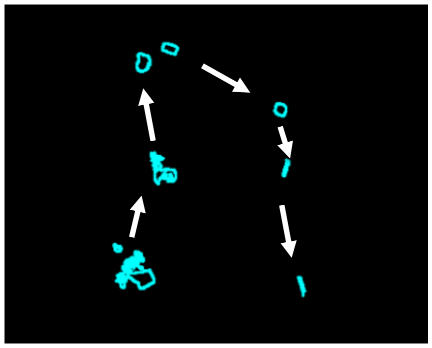


Fig. 11(c) Planned Path along probable target positions

Multi-Goal Path Planner

Once the drone has identified all the probable target positions, it computes a near-optimal path starting from its current position to traverse all the probable target positions only once. This is the **minimum Hamiltonian path problem**[#]. It falls under the category of NP-hard problems, and there exists no polynomial-time algorithm for this. Our algorithm looks for an approximate result for the problem involving the techniques observed in the **Kruskal** Algorithm. The time complexity of this algorithm is **$O(N^2 \log N)$**, where N is total number of probable target positions.

Algorithmic flow of Strategy and Planning:

1. Each drone starts moving from the spawn point to their individually calculated SIPs in each quadrant. Next, the drones move towards the opposite side to scan the quadrant.
2. This process develops a feature map to locate the probable targets. If the number of probable targets falls short of a certain threshold, the drone is set into recovery mode[1].
3. Otherwise, an optimised multi-goal path is planned through probable target positions, eliminating previously identified targets.
4. This path is implemented for searching at a lower height. Each target is explored and verified using detection algorithm. The targets with confidence lesser than the lower threshold are identified as false positives, and those with confidence higher than the upper threshold are identified as true positives.
5. Each drone broadcasts its progress using swarm communication and also keeps track of the overall progress.
6. If the number of true positives identified by all the drones combined becomes equal to the defined number of **search targets (N_T)**, the drone terminates the detection process and lands; else it moves to verify the next probable target. If the iteration ends with number of true positive examples less than N_T, then all drones go into recovery mode[2].

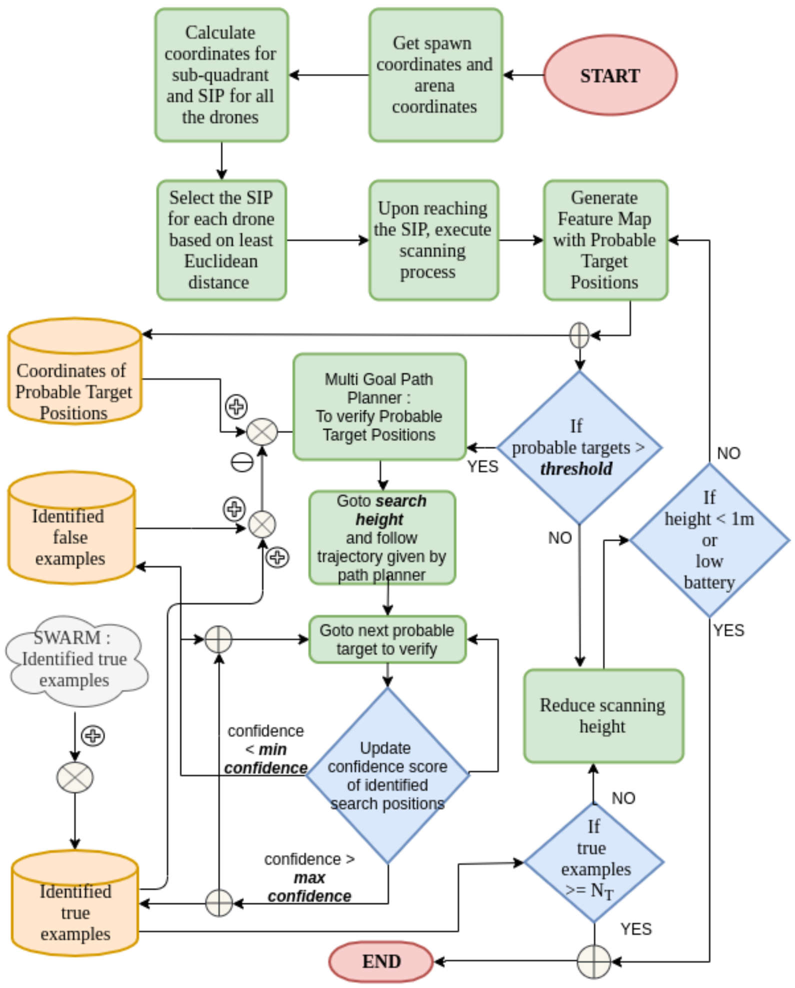

START
Get spawn coordinates and arena coordinates
Calculate coordinates for sub-quadrant and SIP for all the drones
Select the SIP for each drone based on least Euclidean distance
Upon reaching the SIP, execute scanning process
Generate Feature Map with Probable Target Positions
Coordinates of Probable Target Positions
Multi Goal Path Planner : To verify Probable Target Positions
If probable targets > threshold
YES
NO
NO
If height < 1m or low battery
YES
Identified false examples
Goto search height and follow trajectory given by path planner
SWARM : Identified true examples
Goto next probable target to verify
confidence < min confidence
Update confidence score of identified search positions
Reduce scanning height
NO
If true examples >= N_T
YES
confidence > max confidence
Identified true examples
END

Fig. 12 Flow Diagram for Strategy and Planning

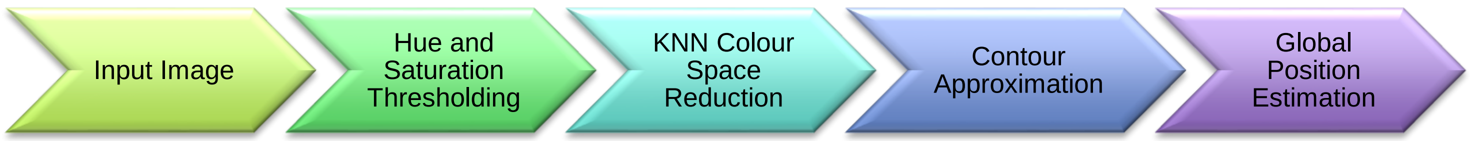
Recovery Mode:

1. If a particular drone enters recovery mode due to shortage of probable targets, then the scanning process is conducted at half of the previous scanning height, whereas the search height is preserved for the drone.
2. If the number of true positives is less than N_T, then all drones starts scanning at half of the previous scanning height.

In the event that the scanning height gets reduced below a threshold value or the battery level falls to a critical point, the drone terminates the detection process and lands.

[#] Rubin, Frank (1974), "A Search Procedure for Hamilton Paths and Circuits", Journal of the ACM, 21 (4): 576–80, doi:10.1145/321850.321854

Target Detection and Localisation



Input Image
Hue and Saturation Thresholding
KNN Colour Space Reduction
Contour Approximation
Global Position Estimation

The algorithm for the detection of the green box is a three-fold process. The first part of the algorithm deals with the preprocessing of images; the second part is getting a mask for the target object. The last part of the algorithm deals with a post-processing step through which we can extract out the rectangle from the mask of the image. Here we are showing the algorithm used to extract the possible masks and approximations done to fit the required specifications. Finally, we are getting the local position of the target using the required transformations.
The example shown to the right is a cropped image of the actual video shot using a 1080x720-resolution camera from a height of 7 meters. We further tested our algorithm from various heights, including 2m, 5m, 7m, and 10m.

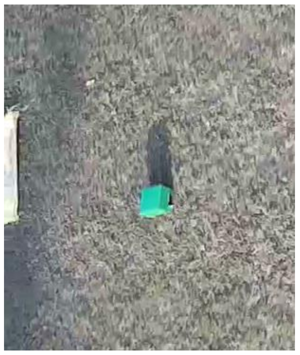


Fig. 13(a) Input Image

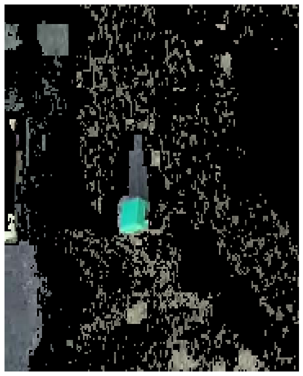


Fig. 13(b) Thresholded Image

The preprocessing step removes unnecessary objects in the image based on their **saturation** and **hue** value. Through this, we can remove the clutters present in the environment (the objects which are not the background, neither the target object) like the chair, tables as mentioned in the problem statement. Through this, we can obtain a mask of the image consisting of the background and target box. However, it might still contain some random objects that follow the same colour distribution as the target or background.

The second part of the algorithm for detection has three major steps. Broadly, these are: **reduction of colour space**, conversion to a binary image, and finally, detection and classification of contours. The first stage, reduction of colour space, works by applying **K-Nearest Neighbours(KNN)**# across the image's pixels. Instead of applying KNN to find the most precise colour space at once, we first split the images into a smaller 5x5 grid and apply KNN to each of these grids independently.

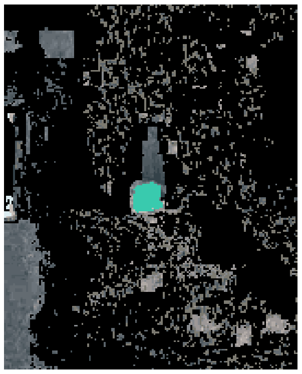


Fig. 13(c) Colour Space Reduced

Because KNN is an O(N^2) algorithm, this gives us a massive speed improvement. Finally, we take this composite image and apply KNN to it entirely. This step now proceeds faster than a usual KNN because the error is already very low, and the threshold error value is obtained very quickly.

There are 2 parameters to be controlled in this stage that are several clusters(L) in step 1 of KNN and the number of parameters(M) in the second step. Now that our colour space for the entire image has been reduced to L, we form binary images for each of the L different colours. In this binary image, we apply some dilations and erosions to get cleaner image. This process gives us M binary images.

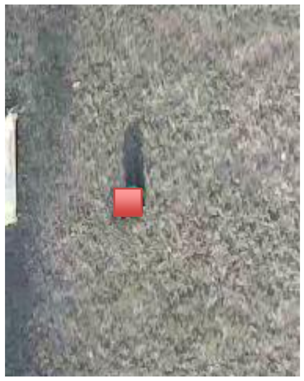
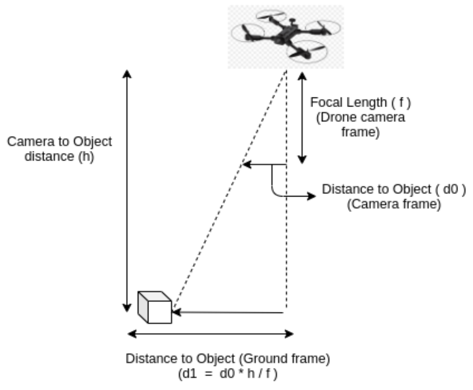


Fig. 13(d) Output Image

In the final part of our algorithm, we find contours in all of the M different binary images. These contours are then replaced by an **approximate contour** that has an arc length of some (1+ε) times original. At this point, we reject all approximate contours that do not have exactly 4 points. Next, we apply specific thresholding criterion to filter out noisy contours such as the ratio of area to original contour, the ratio of diagonals, minimum and maximum contour area. If any contour passes all of these tests, we declare it our intended object.

Now, having obtained the relative position of the object in the image frame, the position could be found out in the real world using the pin-hole model and the height, which can be estimated using the **barometer/rangefinder**. Thus, we can know the relative position of the target with respect to the current position of the quadcopter.



Focal Length (f)
(Drone camera frame)
Camera to Object distance (h)
Distance to Object (d0)
(Camera frame)
Distance to Object (Ground frame)
(d1 = d0 * h / f)

Fig. 14 Pin-Hole Model

Having obtained the GPS location of the drone, the **tf-ROS*** package is used to transform the target position from the local frame to the world frame. The swarm of drones continuously publish coordinates of all the newly detected objects. A subscriber is created using **roslibjs**, which is the core JavaScript library for interacting with ROS from the browser. This subscriber listens to these GPS coordinates and then updates them in a stack. This stack is employed by the mapping script, which uses the **OpenLayers API** to output a geographical map of the field onto the browser with the detected coordinates tagged. Using the above system, we obtain the position of the detected object in real-time on the ground station.

Thus, using the above proposed hardware setup and algorithms we will be able to efficiently solve the given problem statement and the approach can be easily extended to other generalised use cases such as exploration, and search-rescue missions.

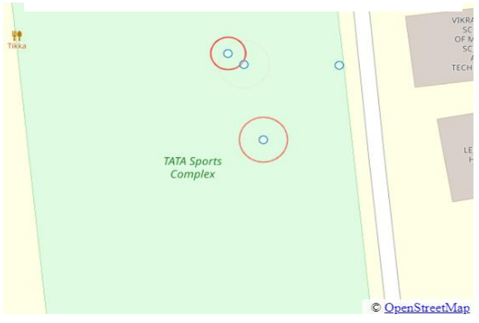


Fig. 15 Geo-tagged points displayed

** TF-ROS* *#KNN Model-Based Approach in Classification*