Anti-Drone

28.04.2020

Computer Vision Team
Student Copter Research Organization (SRM-SCRO)
SRM Institute of Science and Technology
Kattankulathur

Overview

The fast growing technology has resulted in many systems being developed and implemented in minutes of time. As people's needs are increasing day by day and technology aids the Companies satisfy the needs of the people, many new vehicles have been developed. One such vehicle is the Drone. From being used in weddings and functions to take photographs to surveilling and fighting in a war, Drones are widely used. As these small vehicles can be built easily and can be used by everyone, restricting and keeping track of these drones and putting them down have become necessary. That is why we are building a project titled the Anti Drone System which will fulfil these requirements.

Goals

The Anti-Drone Project comprises of four major parts:

1. Detecting the Drone and tracking it
2. Predicting the next points/movement of the drone

3. Calculating the parameters such as number of Drones, speed of the Drone, distance from the camera, height from the Ground
4. Destroying the Drone or soft-killing the Drone

Specification/Procedures

1) Detection and Tracking

Detection:

Detecting the Drone is an important part in an Anti Drone System. We should be able to detect the drone first then only other conditions will follow.

Detection involves using Neural Network and Deep Learning models (Fast R-CNN or YOLO) to depict the drone when it is visible by drawing a bounding box around it.

- For now we plan to use a Fast R-CNN pretrained model available. The pretrained model takes in the image/frame of a video as input and draws a bounding box if a Drone is depicted in the frame.
- Fast R-CNN builds on previous work to efficiently classify object proposals using deep convolutional networks. Compared to previous work, Fast R-CNN employs several innovations to improve training and testing speed while also increasing detection accuracy.
- Fast R-CNN trains the very deep VGG16 network 9x faster than R-CNN, is 213x faster at test-time, and achieves a higher mAP on PASCAL VOC 2012. Compared to SPPnet, Fast R-CNN trains VGG16 3x faster, tests 10x faster, and is more accurate.

Constraints:

Haven't Implemented our Fast R-CNN model yet so not sure whether it will yield the results accurately.

Alternatives

- If the pretrained Fast R-CNN models fail to work or yield inaccurate results we will be using a pre-trained YOLO Algorithm.
- A YOLO algorithm comes with a bunch of classes already trained on it so we will be using the classes available in the YOLO Algorithm. Since YOLO algorithm doesn't have a class separately for drones it categorises Drones as Aeroplanes and aeroplanes have a class ID of 4.
- So if a particular class ID is returned we will use the values that constitute the bounding box provided by the algorithm and return the values of the bounding box (the border values) to the Tracker.

Tracking:

Both the Fast R-CNN and YOLO Algorithm may not yield an accurate result to continuously detect the Drone. That is where a Tracking Algorithm plays its role. OpenCV offers a wide range of Tracking Algorithms which can be used to track an object.

The Tracking Algorithms provided by OpenCV are:

1. KCF
2. MIL
3. TLD
4. Median-Flow
5. CSRT
6. MOSSE

The above mentioned algorithms have their own pros and cons (such as low frame rate or less accuracy), but on testing we found two algorithms which were overpowering the rest. The two algorithms are

I. **TLD:** TLD stands for Tracking, learning and detection. As the name suggests, this tracker decomposes the long term tracking task into three components: (short term) tracking, learning, and detection. From the author's paper, 'The tracker follows the object from frame to frame. The detector localises all appearances that have been observed so far and corrects the tracker if necessary. The learning estimates detector's errors and updates it to avoid these errors in the future.' This output of this tracker tends to jump around a bit.

II. **MOSSE:** Minimum Output Sum of Squared Error (MOSSE) uses adaptive correlation for object tracking which produces stable correlation filters when initialized using a single frame. MOSSE tracker is robust to variations in lighting, scale, pose, and non-rigid deformations. It also detects occlusion based upon the peak-to-sidelobe ratio, which enables the tracker to pause and resume where it left off when the object reappears. MOSSE tracker also operates at a higher fps (450 fps and even more). To add to the positives, it is also very easy to implement, is as accurate as other complex trackers and much faster.

Both these trackers work by getting the ROI of the image and then using the algorithm on which it is built. It tracks the drone and depicts the status whether the algorithm is tracking the drone or not. Once the Drone is not in the frame a status depicting 'lost' is displayed and as soon as the drone is back into the frame the tracking algorithm begins tracking the Drone.

Overall Workflow

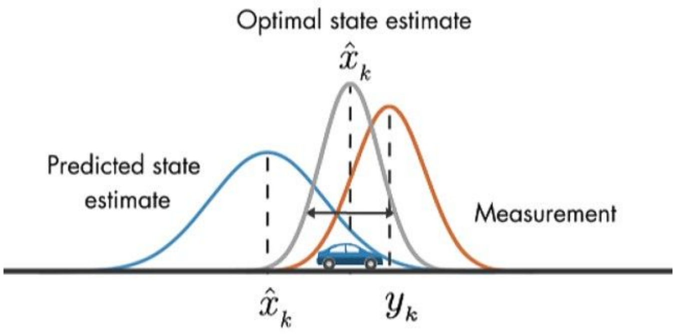
1. A detecting algorithm to detect the presence of a drone and return the bounding box values.

2. Tracking algorithm to use the values provided by bounding box values as Region of Image (ROI) and resume tracking the drone.
3. Tracking algorithm depicting the status whether the drone is available or not by indicating whether it is Tracking or Lost.
4. Detection algorithm again to resume if the status returned by tracking algorithm is 'Lost' and return the bounding box values to tracking algorithm after detecting the drone.

2) Predicting the next points/movements of the drone

The Kalman Filter

Kalman Filters are very popular for tracking obstacles and predicting current and future positions. It is used in all sort of robots, drones, self-flying planes, self-driving cars, multi-sensor fusion, ...



A Kalman Filter is used on every bounding box, so it comes after a box has been matched. When the association is made, predict and update functions are called. These functions implement the math of Kalman Filters composed of formulas for determining state mean and covariance.

State Mean and Covariance

Mean and Covariance are what we want to estimate. Mean is the coordinates of the bounding box, Covariance is our uncertainty on this bounding box having these coordinates.

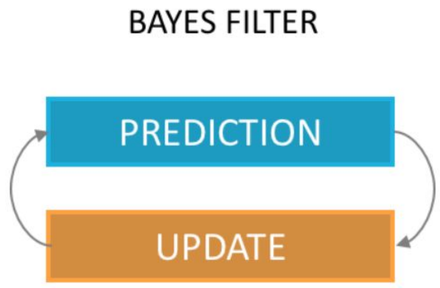
Mean (x) is a state vector. It is composed by coordinates of the centre of the bounding box (cx, cy), size of the box (width, height) and the change of each of these parameters, velocities.

When we initialize this parameter, we set velocities to 0. They will then be estimated by the Kalman Filter

Covariance (P) is our uncertainty matrix in the estimation. We will set it to an arbitrary number and tweak it to see results. A larger number means a larger uncertainty.

That's all we need to estimate : a state and an uncertainty ! There is two steps for a Kalman Filter to work : prediction and update. Prediction will predict future positions, update will correct them and enhance the way we predict by changing uncertainty. With time, a Kalman Filter gets better and better to converge.

How to use ?



Kalman Filter cycle

At time t=0, we have a measurement of 3 bounding boxes. The Hungarian Algorithm defines them at 3 new detections. We therefore only have 3 detections in our system. For each box, we initialize Kalman Matrices with coordinates of the bounding boxes.

At time t=1, we have 3 bounding boxes, of the same object. The Hungarian Algorithm matches them with the 3 former boxes and we can start calling predict and update. We predict the actual bounding boxes at time t from the bounding boxes at time t-1 and then update our prediction with the measurement at time t.

Prediction

Prediction phase is matrix multiplication that will tell us the position of our bounding box at time t based on its position at time t-1.

$$x' = Fx + u$$

$$P' = FPF^T + Q$$

Prediction equations

So imagine we have a function called predict() in a class Kalman Filter that implements these maths. All we need to do is to have correct matrices F, Q and u. We will not use the u vector as it is used to estimate external forces, which we can't really do easily here.

How to set up the matrices correctly ?

State Transition : F

F is the core implementation of what we will define. What we put here is important because when we will multiply x by F, we will change our x and have a new x, called x'.

$$\underbrace{\begin{pmatrix} cx\\ cy\\ w\\ h\\ vx\\ vy\\ vw\\ vh \end{pmatrix}_{t+1}}_{x'} = \underbrace{\begin{pmatrix} 1&0&0&0&dt&0&0&0\\ 0&1&0&0&0&dt&0&0\\ 0&0&1&0&0&0&dt&0\\ 0&0&0&1&0&0&0&dt\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{pmatrix}}_{F} \cdot \underbrace{\begin{pmatrix} cx\\ cy\\ w\\ h\\ vx\\ vy\\ vw\\ vh \end{pmatrix}_{t}}_{x}$$

x Predict formula

As you can see, F [8x8] matrix contains a time value : dt is the difference between current frame and previous frame timestamp. We will end up having cx' = cx + dt*vx for the first line, cy' = cy + dt*vy, and so on...

In this case, it means that we consider the velocity to be constant. We therefore set up F to implement the Constant Velocity model. There are a lot of models we can use depending on the problem we want to solve.

Process Covariance Matrix : Q

Q [8x8] is our noise matrix. It is how much confidence we give in the system. Its definition is very important and can change a lot of things. Q will be added to our covariance and will then define our global uncertainty. We can put very small values (0.01) and change it with time.

Based on these matrices, and our measurement, we can now make a prediction that will give us x' and P'. This can be used to predict future or actual positions. Having a matrix of confidence is useful for our filter that can model uncertainty of the world and of the algorithm to get better results.

Update

Update phase is a correction step. It includes the new measurement (z) and helps improve our filter.

$$\begin{aligned} y &= z - Hx' \\ S &= HP'H^T + R \\ K &= P'H^TS^{-1} \end{aligned}$$

$$\begin{aligned} x &= x' + Ky \\ P &= (I - KH)P' \end{aligned}$$

Update equations

The process of update is to start by measuring an error between measurement (z) and predicted mean. The second step is to calculate a Kalman Gain (K). The Kalman gain is used to estimate the importance of our error. We use it as a multiplication factor in the final formula to estimate a new x.

Again, How to set up the matrices correctly ?

Measurement Vector: Z

z is the measurement at time t. We don't input velocities here as it is not measured, simply measured values.

Measurement Matrix : H

H [4x8] is our measurement matrix, it simply makes the math work between all or different matrices. We put the ones according to how we defined our state, and its dimension highly depends on how we define our state.

Measurement Noise : R

R is our measurement noise; it's the noise from the sensor. For a LiDAR or RADAR, it's usually given by the constructor. Here, we need to define a noise for YOLO algorithm, in terms of pixels. It will be arbitrary, we can say that the noise in terms of the centre is about 1 or 2 pixels while the noise in the width and height can be bigger, let's say 10 pixels.

Matrix multiplications are made; and we have a prediction and update loop that gets us better results than the yolo algorithm. The filter can also be used to predict at time t+1 (prediction with no update) from time t. For that, it needs to be good enough and have a low uncertainty.

Conclusion

To predict future bounding box positions or actual positions, Kalman Filters are used. From that, we could use Machine Learning to predict future behaviour or trajectories, we could be able to estimate what an obstacle has been doing the last 10 seconds. This tool is powerful and tracking become not only possible, but also very accurate. Velocities can be estimated and a huge set of possibilities becomes available.

Alternative Method :-

Motion of the moving drone in a video stream, captured by a camera, is studied and its velocity is detected and its future position is predicted. The process consists of the following three stages. In the first stage, event scenes are captured and processed using computer vision techniques. Second stage is recognizing and following the event. These two steps have already been explained above. In the last stage, future position of the object and its velocity are predicted. The centroid of object is computed to use in the analyses of the position of the moving object image.

The computation of the centroid :

The centroid of the object has been used for target following. The x-y curve of the object will be obtained using this centroid. The ‘centroid’ is a good parameter for specifying the location of an object. It is the point having coordinates x', y’ such that the sum of the square of the distance from it to all other points within the object is a minimum. The formula used in centroid computation is given below. In this equation, n refers to the number of “0” valued pixels in the edge detected image.

$$\text{Centroid} = \left(\frac{1}{n} \sum_{x} x, \frac{1}{n} \sum_{y} y \right)$$

The detection of object movements and target :

The video occurs by many sequential images changing very rapidly in a frame. The area and centroid of the object is computed in each frame. The area of object is an important component in the computation of the object. If the object is in the image work area, the area value will not change. In this situation, the centroid is real centroid of object and can be used in computations. If the area of object is changing time by time, the whole object is not in the work area. The digitalization of the image area is very easy because it is fixed. So, the centroid of the object can be digitalized on x and y-axis easily. Using these obtained values, the movement analysis of the object is done. Here, it is obtained that the object is moving as linear or regular circular. The road taken on x and y-axis, and speed, compound speed and road is found in x and y direction. Using this obtained speed, the position of the object is computed at time “t”.

The computation of the object speed :

In the computation of the speed, t1 and t2 image frames are used in the followed image sequences. The system clock of computer is used to decide the time. The position change of object is calculated using x-y curve. By the assumption of linear motion for objects movement, the average speed can be calculated by using the equation below –

$$v = \frac{\Delta x}{\Delta t} = \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}{t_2 - t_1}$$

If the object is moving regularly circular –

$$v = \frac{2\Pi r}{(t_2 - t_1)} \frac{\alpha}{360}.$$

On this situation, firstly, we need to find r and α. r curve is computed by the assumption of that it is a part of any circle. Here, all pixels are controlled and the distances from circle are computed by using an algorithm. When all the distances are equal, the centroid of the circle is found, and the distance gives r value. Then α can be found. Then, the speed of the object is computed.

The decision of the target :

The position of the object at any time in the future can be computed by the assumption that the object is steady moving due to the computed speed. The coordinate of centroid at any time can be computed by using equation x = y, t. The coordinate can be found using this computed x value.

3)Calculating the Parameters

Parameters for the drone include

- Speed of the Drone,
- Number of Drones,
- Distance from the camera,
- Height from the Ground

Speed Estimation:

Speed estimation of drones is crucial for (i) the detection of intruder drones in protected environments; (ii) sense and avoid purposes on drones or on other aerial vehicles and (iii) multi-drones control scenarios, such as environmental monitoring, surveillance and exploration. For an anti drone system, speed estimation helps in prediction of the drone movement which would give us the best point to release the blocking mechanism.

- Our speed formula is

speed = distance / time (Equation 1.1).

- We need to know the distance constant for the frame.

- Meters per pixel are calculated by dividing the distance constant by the frame width in pixels (Equation 1.2).

- Distance in pixels is calculated as the difference between the centroids as they pass by the columns for the zone (Equation 1.3). Distance in meters is then calculated for the particular zone (Equation 1.4).

- Four timestamps (t) will be collected as the drone moves through the FOV past four waypoint columns of the video frame.

- Three pairs of the four timestamps will be used to determine three delta t values.

- We will calculate three speed values (as shown in the numerator of Equation 1.5) for each of the pairs of timestamps and estimated distances.

- The three speed estimates will be averaged for an overall speed (Equation 1.5).

The speed is converted into speedMPH or speedKMPH .

The following equations represent our algorithm:

$$meters\ per\ pixel = mpp = \frac{distance\ constant}{frame\ width} \tag{1.2}$$

$$distance\ in\ pixels = p_{ab} = \mid col_B - col_A \mid \tag{1.3}$$

$$distance\ in\ meters\ zone_{ab} = d_{ab} = p_{ab} * mpp \tag{1.4}$$

$$average\ speed = \frac{\frac{\triangle t_{ab}}{d_{ab}} + \frac{\triangle t_{bc}}{d_{bc}} + \frac{\triangle t_{cd}}{d_{cd}}}{3} \tag{1.5}$$

Number of drones:

For a multiple-drones situation, we need to count the number of drones. An accurate count of the drones will help us to deal with the situation. A better blocking mechanism can be formed. On detection of drones, we create bounding boxes. On counting the number of bounding boxes, we get the number of drones in the frame.

Calculating distance of detected drone from camera:

In order to determine the distance from our camera to a detected drone, we are going to utilize triangle similarity.

Triangle similarity:

The triangle similarity goes something like this: Let's say we have a marker or object with a known width W. We then place this marker some distance D from our camera. We take a picture of our object using our camera and then measure the apparent width in pixels P. This allows us to derive the perceived focal length F of our camera:

F = (P x D) / W

For example, let’s say I place a standard piece of 8.5 x 11in piece of paper (horizontally; W = 11) D = 24 inches in front of my camera and take a photo. When I measure the width of the piece of paper in the image, I notice that the perceived width of the paper is P = 248 pixels.

My focal length F is then:

F = (248px x 24in) / 11in = 543.45

As I continue to move my camera both closer and farther away from the object/marker, I can apply the triangle similarity to determine the distance of the object to the camera:

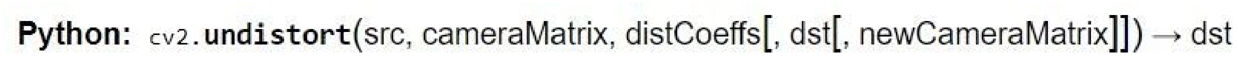
D’ = (W x F) / P

Again, to make this more concrete, let’s say I move my camera 3 ft (or 36 inches) away from my marker and take a photo of the same piece of paper. Through automatic image processing I am able to determine that the perceived width of the piece of paper is now 170 pixels. Plugging this into the equation we now get:

D’ = (11in x 543.45) / 170 = 35in

Or roughly 36 inches, which is 3 feet.

ALTITUDE MEASUREMENT: - To measure the altitude of a drone in flight we are going to use a set of Simple Formulas based on trigonometry along with correction for Lens distortion/Barrel distortion. This will be used to correct for non-linear increase of distortion when the drone approaches close to the camera. Also, to note unless using a very wide-angle lens as the drone moves further from camera tiny errors in the angle of view calculation will stack exponentially so there will be a function for accuracy calculation as well.



```
Python: cv2.undistort(src, cameraMatrix, distCoeffs[, dst[, newCameraMatrix]]) → dst
```

STEP 1: Correct for Barrel Distortion: - For this we will use a function present in opencv2 known as undistort.

1)SRC- Input Image

2)cameraMatrix- it is a 3X3 matrix that looks like

fx 0 cx

0 fy cy

0 0 1

And can be created by

```
Mat cameraMatrix = (Mat1d(3, 3) << fx, 0, cx, 0, fy, cy, 0, 0, 1);
```

3)distCoeffs- is a vector or 4, 5, or 8 parameters, that looks like

k1, k2, p1, p2 [, k3 [, k4, k5, k6]]

and can be created by

```
Mat distortionCoefficients = (Mat1d(1, 4) << k1, k2, p1, p2);
Mat distortionCoefficients = (Mat1d(1, 5) << k1, k2, p1, p2, k3);
Mat distortionCoefficients = (Mat1d(1, 8) << k1, k2, p1, p2, k3, k4, k5, k6);
```

These parameters can be obtained and calculated from the image data stored in the specific camera and can be extracted by reading the settings .xml file. The meaning of these parameters are explained in opencv documentation here: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

4)dst- Output image(must be same size and dimensions as input image)

5)newCameraMatrix- Camera matrix of the distorted image. By default, it is the same as cameraMatrix but you may additionally scale and shift the result by using a different matrix.

Step 2: Now we will find the angle of view of the Camera(A):-

$$\alpha = 2 \arctan \frac{d}{2f}$$

d- size of film/sensor, for eg a 36mm*24mm, inputting 36mm will give you horizontal angle while inputting 24mm will give you vertical angle.

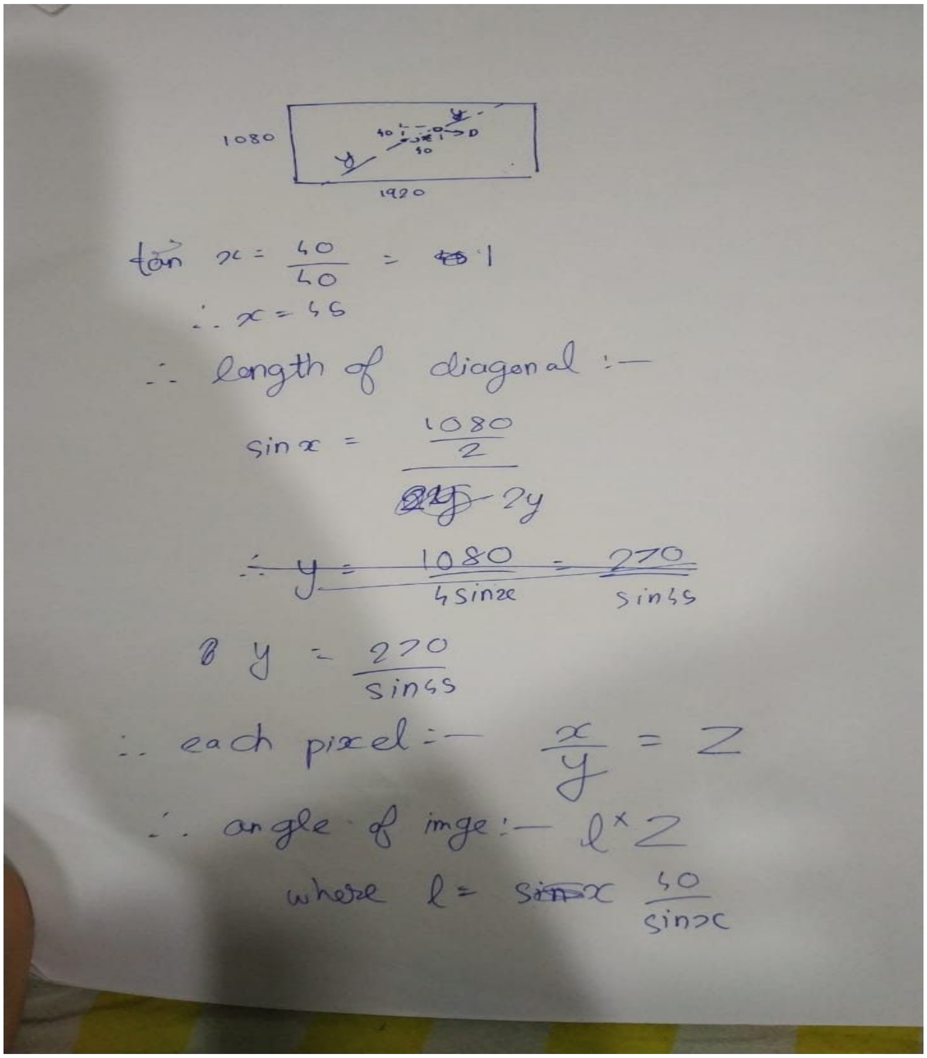
f- this is just the effective focal length which is just focal length *crop factor. Note for lenses with variable focal length, this will vary from image to image.

*NOTE- since this is a trigonometric calculation, it will not give exact angle but for a non-wide angle lens but the angle will then approximately be equal to:

$$\frac{180d}{\pi f}$$

Step 3: Pixel Degree: -

This will require us to know the resolution of the input image and angle of view of camera, so for an example, let's take resolution as (1920*1080). Suppose image is detected at (1000,580)


1080
1920
40
40
x
y
y
D

$\tan x = \frac{40}{40} = \cancel{45}\ 1$

$\therefore x = 45$

∴ length of diagonal :-

$$\sin x = \frac{\frac{1080}{2}}{\cancel{2y}\ 2y}$$

~~$\therefore y = \frac{1080}{4\sin x} = \frac{270}{\sin 45}$~~

$\cancel{8}\ y = \frac{270}{\sin 45}$

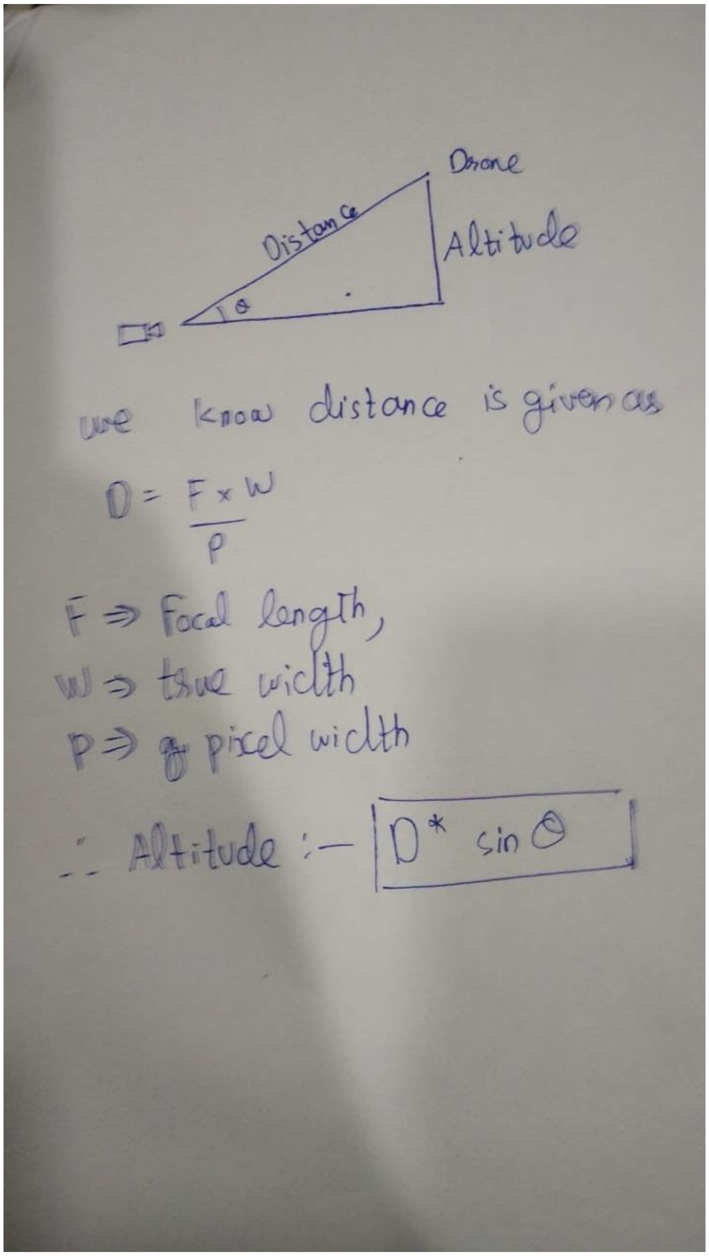
∴ each pixel :- $\frac{x}{y} = Z$

∴ angle of imge :- $l \times Z$

where $l = \cancel{\sin}x\ \frac{40}{\sin x}$

So from here we have found the angle of image, we will have to implement this algorithm for all four quadrants separately.

Step 4: Measuring altitude:-


Drone
Distance
Altitude
θ

we know distance is given as

$$D = \frac{F \times W}{P}$$

F ⇒ Focal length,
W ⇒ true width
P ⇒ ~~g~~ pixel width

$$\therefore \text{Altitude} :- \boxed{D^{*} \sin \theta}$$

This will simply give us the altitude from the altitude of the camera, assuming the camera is pointed horizontally across a plane surface, the extra altitude has to be added in manually. Note that theta is same as L*Z. also note step 3 is given to quicken the process when the drone center lies on those 3 axes.

4) Destroying the Drone

The destruction of the Drone part is slightly unclear upon discussion. After completion of the primary three parts a careful look will be given to this part.

Milestones

I. Detection and Tracking

Drone

II. Predicting

Drone

III. Calculating the Parameters

Drone