Project Report Stage - 2

Image Based Drone Stabilization

INDUSTRY PARTNER: InVue (Rick Terell)

Team Members:

Rohan Bhosale (801132664)
Somesh Kale (801131989)
Jugal Patel (801075521)
Divya Patel (801079381)
Hunter Pace

1. **Business Partner/Customer Interaction:**

Meeting(Face-to-Face):

Date : 10/14/2019 7:00 pm

Goal of the Project:

Taking a video feed as an input and reporting 6-axis estimated camera motion updates as the camera moves. The end goal is to be able to implement UAV stabilization using the drones on-board camera.

Key-Points:

- ❏ Discussed about the problem we will be solving with Computer vision techniques in this project.
- ❏ James presented us with his idea, How he got a thought about that idea and helped us understand what are the expectations from us.
- ❏ This project is divided into 2 steps and 1 optional step:

    1. "Video instrumentation" prototype:
       Taking a video stream as an input and estimating output motion as text.
    2. Manual stabilization prototype:
       Translate motion estimation into 6-axis motion commands given to a person holding the camera that return the camera to its starting

position. 12 possible commands to slide or rotate left/right/up/down/forward/back which the camera operator follows manually.

3. UAV stabilization prototype.
   We will be taking the manual demo and implementing it with an actual UAV. The demo would be to fly the UAV to a position and then engage the stabilization, the result being, hopefully, that the UAV holds its position and does not drift. Rick has a UAV that he can get a video stream from and can issue motion controls.

2. Literature Review:

Review of any 3 papers:

1)https://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS15/paper/download/10398/10299

This research paper is about stabilizing drone and hold the drone at fix position regardless of outside influence and inaccurate sensors. Basically old solution of using SLAM and sensors was not sufficient. It was taking too much time to process and transfer that big data. So, it was not accurate and information received was old. So, in this research paper they come up with some motion estimation and exact matching algorithm to solve this problem through computer vision instead of using mechanical techniques which are not that good. For this they used AR Drone which has USB drone flash storage and 1 GHz ARM Cortex processor running a minimalistic GNU/Linux system. They generally transfer the video by decrease the quality and send through UDP connections. They also implemented two cameras, one for taking video and one at the lower part of the drone to just see the floor and from that try to find the altitude. FIrst they start localizing drone using GPS , optical flow and acceleration-based speed estimation. Such techniques however unable to eliminate drift and localization error grows over time. After that, they come up with Extended Kalman Filter. It combines the visual system fails and no absolute pose is measured. It can predict drone's movement for a short time, which is used to balance the long network latency. Now, before localizing you have to map depth of every point of the image that we will receive. For that , you need to know your environment before hand only. Now, to do that they use FAST corner detector and a pose

estimation, which can be later updated in order to increase the precision of the pose and therefore even the precision of associated keypoints locations.
Now, to find camera pose assume that we have a calibrated pin-hole camera projection
model CamProj:

$$\begin{pmatrix} u_i \\ v_i \end{pmatrix} = CamProj \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Where x; y; z are the coordinates of a landmark relative to the current camera pose and ui; vi are the (pixel) coordinates of the landmark projection into the image plane of the camera. Let CameraPoseT rans(; pi) denote the location of the landmark pi relative to the camera pose . We can use the defined projection to express the reprojection error vector ej of the landmark with coordinate vector pj (relative to the map origin) which was observed at uj ; vj . Reprojection error is the difference between where the landmark pj should be observed according to the map, if the drone's pose is , and where it was observed using the camera.

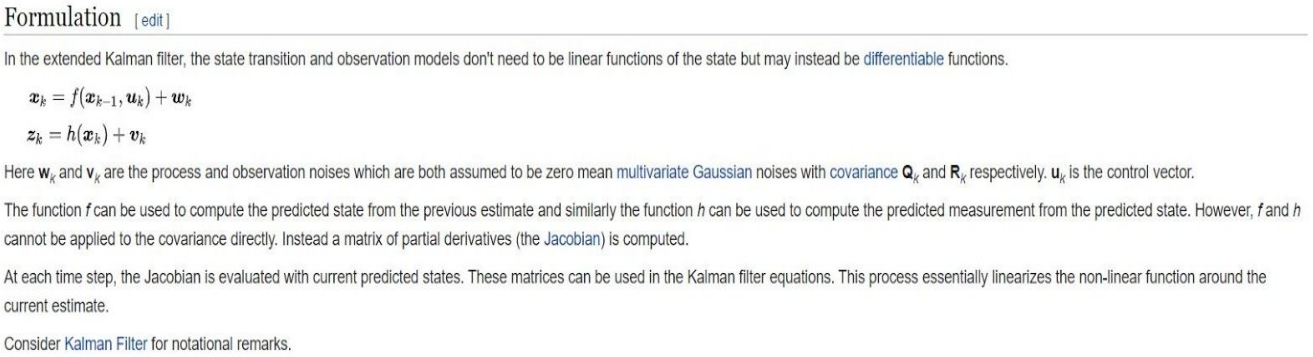
$$e_j = \begin{pmatrix} u_j \\ v_j \end{pmatrix} - CamProj(CameraPoseTrans(\mu, p_j)) \quad (2)$$

In the correct pose of the drone, the reprojection errors should be very small. Therefore we can use ej for finding the most-likely camera pose 0:

$$\mu' = \underset{\mu}{\operatorname{argmin}} \sum_{j \in S} Obj\left(\frac{e_j}{\sigma_j}, \sigma_T\right)$$

where S denotes the set of landmark observations, Obj(; T) is the Tukey biweight objective function (Hampel, Ronchetti, and Rousseeuw 1986), and T is a robust estimate of the distribution's standard deviation. Here, they meant adding newly observed landmarks into the map and updating the pose of known landmarks after future observations in order to improve the precision of their location by mapping.
They briefly summarize the extended kalman filter which they use to localize the drone. Extended Kalman filter is the nonlinear part of Kalman filter which linerizes

about an estimate of the current mean and covariance. Formula given in wikipedia is given below.



Formulation [edit]

In the extended Kalman filter, the state transition and observation models don't need to be linear functions of the state but may instead be differentiable functions.

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k$$

$$\boldsymbol{z}_k = h(\boldsymbol{x}_k) + \boldsymbol{v}_k$$

Here $\mathbf{w}_k$ and $\mathbf{v}_k$ are the process and observation noises which are both assumed to be zero mean multivariate Gaussian noises with covariance $\mathbf{Q}_k$ and $\mathbf{R}_k$ respectively. $\mathbf{u}_k$ is the control vector.

The function *f* can be used to compute the predicted state from the previous estimate and similarly the function *h* can be used to compute the predicted measurement from the predicted state. However, *f* and *h* cannot be applied to the covariance directly. Instead a matrix of partial derivatives (the Jacobian) is computed.

At each time step, the Jacobian is evaluated with current predicted states. These matrices can be used in the Kalman filter equations. This process essentially linearizes the non-linear function around the current estimate.

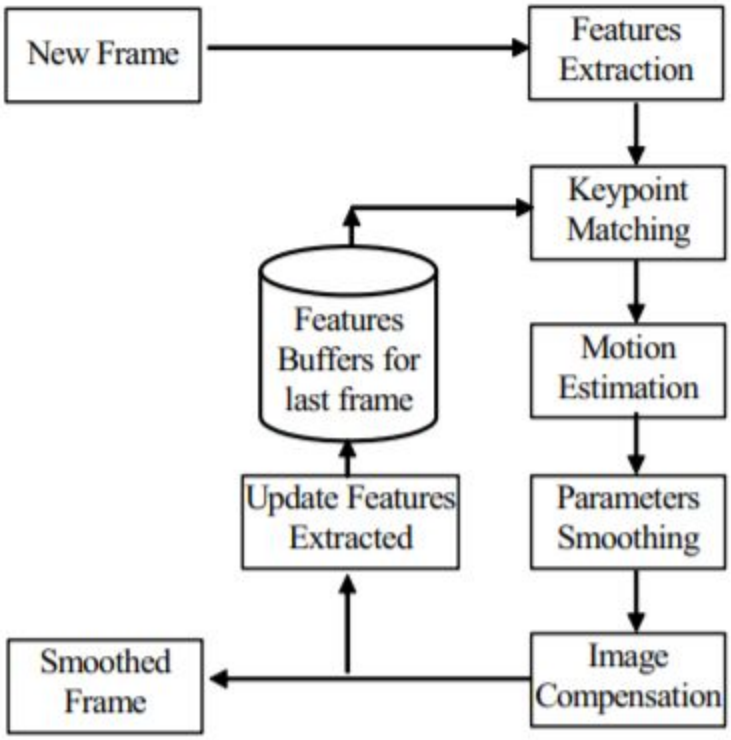
Consider Kalman Filter for notational remarks.

Only problem is if the initial location given by us has error then it will diverge very fast.

Now, this method they evaluate in different conditions. It is working fine and latency is also great in most places but in some dark light rooms, it was hard to detect corner points and drone tends to drift with high error. They also have one interesting video https://vimeo.com/102528129 which really helps you understand how things are working.

2)http://www.mva-org.jp/Proceedings/2011CD/papers/09-27.pdf

This paper presents a new real-time video stabilization method for Unmanned Aerial Vehicles (UAV). It mainly consists of three steps. Firstly, the keypoints are located based on FAST corner detection and preliminarily matched. Secondly, the matched keypoints are then involved for estimation of affine transform to reduce false matching keypoints. Finally, motion estimation is performed based on affine transform model and the compensation for vibration is conducted based on Spline smoothing.The flowchart of our method is illustrated in . First of all, corner based features are extracted by FAST corner detector [7] and matching pairs are determined. Next, motion between two consecutive frames is estimated based on an affine transform model. Subsequently, the estimated motion parameters are accumulated and smoothed by a spline model. Finally, the frames are compensated based on smoothed parameters and form a stable video. The details are explained as follows.



And the last step include experimental results which includes keypoint detection and matching and motion estimation and video stabilization.A video stabilization algorithm has been presented in this paper, which consists of three steps. Firstly, the keypoints are detected and matched. Then, the motion between two consecutive frames is estimated. And finally, the cumulative distortion of a frame relative to the first frame is generated and the parameters are smoothed by cubic spline smoothing. The proposed method was tested for video stabilization in the real video captured by camera installed on UAV, the experimental results show the efficiency and accuracy of the proposed algorithm

3) https://www.sciencedirect.com/science/article/pii/S1474667016438188

This research paper is regarding how one company named Parrot which has the same idea as what we are working on completing the project and started selling AR drone which user can remotely controlled through a user-friendly graphical interface running on any Apple devices.The idea of company is to go way beyond conventional usage commonly considered in both civilian and military applications. Main problem they want to solve is that system must be easy to fly

and safe. So, users just have to give high level command and drone should follow that safely without getting unstable or start drifting. They wanted to make a robust system which they can use indoor and outdoor,too. So, in GPS and sensors estimation sometimes data fusion happens or sometimes signal got lost. So, it was not that robust. Here also they use two cameras, vertical for estimating the speed of the vehicle. First algorithm they used is optical flow for the finding velocity. This will also deduct altitude problem. They also used corner tracking as their second algorithm for finding all kind of corners of all the objects available in the image. Which helps to compare two frames and estimate drift. They also collaborate with Inertial sensor to make systems more reliable. Consider a sensor (e.g. a triaxial accelerometer) and note Y_m its measurement and Y_v the true value of the sensed variable, the following model is considered

$$Y_m = \alpha R Y_v + \beta \text{ with } \beta = [\beta_1, \beta_2, \beta_3]^T,$$

$$R = \begin{bmatrix} 1 & \psi & -\theta \\ -\psi & 1 & \phi \\ \theta & -\phi & 1 \end{bmatrix}, \quad \alpha = \text{diag}(\alpha_1, \alpha_2, \alpha_3)$$

Where R represents a first order misalignment matrix (micro-rotator) between the frame of the camera to the frame of the sensor board, α stands for scale factors and β for accelerometer bias. So after comparing two frames they are tracking displacement of objects and for that they are considering related to the linear velocity, angular velocity and altitude of the vehicle. They also mention how they use aerodynamics model for velocity estimation and at last for achieving the main goal of stabilizing the drone. They have given summary of that part as 'liner drag term exists from interaction between the grid body and the rotors and this term us reinforced by tilt phenomenon which changes a lift force component in drag'.

This was their user interface for remote user.

So, the result of this experiment is one estimating architecture which is a complex combination of several principles, used to determine, over distinct time-horizons, the biases and other defects of each sensor. Right now, the company is focusing on making video games which will use this type of drones platform in interactive AR gameplays.

Other Related Paper that we looked into :
https://ieeexplore.ieee.org/abstract/document/4177606

https://patents.google.com/patent/US8498447B2/en

https://link.springer.com/chapter/10.1007/978-3-319-26327-4_11

http://www.robots.ox.ac.uk/~vgg/hzbook/hzbook2/HZepipolar.pdf

3. **Review of Available Open Source Code:**

Determining change in camera from 2 pictures of a Rubek's Cube (Needs distance to object in question):

```
1. Run Feature Detection and Detector Extractor on both images.
2. Match Features.
3. Use F = cv::findFundamentalMatrix with Ransac.
4. E = K.t() * F * K. // K needs to be found beforehand.
5. Do SingularValueDecomposition of E such that E = U * S * V.t()
6. R = U * W.inv() * V.t() // W = [[0, -1, 0], [1, 0, 0], [0, 0, 1]]
7. Tx = V * Z *  V.t() // Z = [[0, -1, 0], [1, 0, 0], [0, 0, 0]]
8. get t from Tx (matrix version of cross product)
9. Find the correct solution. R.t() and -t are possiblities.
10. Get overall scale by knowing the length of the size of the Rubrik's cube.
```

1D Kalman Filter:

Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe. The filter is named after Rudolf E. Kálmán, one of the primary developers of its theory.

After implementing this filter, you should see that you can go from a very uncertain location Gaussian to a more and more certain Gaussian, as pictured below. This code is really just a simplified version of the Kalman filter that runs in the Google self-driving car that is used to track surrounding vehicles and other objects.

```
# import math functions
from math import *
import matplotlib.pyplot as plt
import numpy as np

# gaussian function
def f(mu, sigma2, x):
    ''' f takes in a mean and squared variance, and an input x
       and returns the gaussian value.'''
    coefficient = 1.0 / sqrt(2.0 * pi *sigma2)
    exponential = exp(-0.5 * (x-mu) ** 2 / sigma2)
    return coefficient * exponential
```

You've also been given the complete `update` code that performs a parameter update when an initial belief and new measurement information are merged. And the complete `predict` code that performs an update to a Gasuuain after a motion is incorporated.

```
# the update function
def update(mean1, var1, mean2, var2):
    ''' This function takes in two means and two squared variance terms,
        and returns updated gaussian parameters.'''
    # Calculate the new parameters
    new_mean = (var2*mean1 + var1*mean2)/(var2+var1)
    new_var = 1/(1/var2 + 1/var1)

    return [new_mean, new_var]


# the motion update/predict function
def predict(mean1, var1, mean2, var2):
    ''' This function takes in two means and two squared variance terms,
        and returns updated gaussian parameters, after motion.'''
    # Calculate the new parameters
    new_mean = mean1 + mean2
    new_var = var1 + var2

    return [new_mean, new_var]
```

```
update(20,9,30,3)
```

```
[27.5, 2.25]
```

You'll see that you are given initial parameters below, and this includes and nitial location estimation, `mu` and squared variance, `sig`. Note that the initial estimate is set to the location 0, and the variance is extremely large; this is a state of high confusion much like the *uniform* distribution we used in the histogram filter. There are also values given for the squared variance associated with the sensor measurements and the motion, since neither of those readings are perfect, either.

You should see that even though the initial estimate for location (the initial `mu`) is far from the first measurement, it should catch up fairly quickly as you cycle through measurements and motions.

]:
```
# measurements for mu and motions, U
measurements = [5., 6., 7., 9., 10.]
motions = [1., 1., 2., 1., 1.]

# initial parameters
measurement_sig = 4.
motion_sig = 2.
mu = 0.
sig = 10000.


## TODO: Loop through all measurements/motions
# this code assumes measurements and motions have the same length
# so their updates can be performed in pairs
for n in range(len(measurements)):
    # measurement update, with uncertainty
    mu, sig = update(mu, sig, measurements[n], measurement_sig)
    print('Update: [{}, {}]'.format(mu, sig))
    # motion update, with uncertainty
    mu, sig = predict(mu, sig, motions[n], motion_sig)
    print('Predict: [{}, {}]'.format(mu, sig))


# print the final, resultant mu, sig
print('\n')
print('Final result: [{}, {}]'.format(mu, sig))
```

```
Update: [4.998000799680128, 3.9984006397441023]
Predict: [5.998000799680128, 5.998400639744102]
Update: [5.999200191953932, 2.399744061425258]
Predict: [6.999200191953932, 4.399744061425258]
Update: [6.999619127420922, 2.0951800575117594]
Predict: [8.999619127420921, 4.09518005751176]
Update: [8.999811802788143, 2.0235152416216957]
Predict: [9.999811802788143, 4.023515241621696]
Update: [9.999906177177365, 2.0058615808441944]
```

Automating the implementation of Kalman Filter Algorithms:

Autofilter is a tool that generates implementations that solve estimation problems using kalman filters. From a high-level, mathematics-based description of a state estimation problem, Autofilter automatically generates code that computes a statistically optimal estimate using one or more of a number of well_known variants of the kalman filter algorithm.

https://ti.arc.nasa.gov/m/tech/rse/publications/papers/AIKFA04/autofilter.pdf

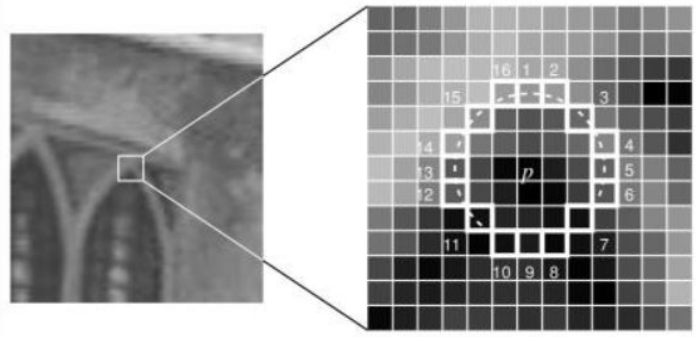
Fast Corner Detection Algorithm:

We saw several feature detectors and many of them are really good. But when looking from a real-time application point of view, they are not fast enough. One best example would be SLAM (Simultaneous Localization and Mapping) mobile robot which have limited computational resources.

As a solution to this, FAST (Features from Accelerated Segment Test) algorithm was proposed by Edward Rosten and Tom Drummond in their paper “Machine learning for high-speed corner detection” in 2006 (Later revised it in 2010). A basic summary of the algorithm is presented below.

Feature Detection using FAST

1. Select a pixel p in the image which is to be identified as an interest point or not. Let its intensity be I_p.
2. Select appropriate threshold value t.
3. Consider a circle of 16 pixels around the pixel under test. (See the image below)



4. Now the pixel p is a corner if there exists a set of n contiguous pixels in the circle (of 16 pixels) which are all brighter than $I_p + t$, or all darker than $I_p - t$. (Shown as white dash lines in the above image). n was chosen to be 12.
5. A **high-speed test** was proposed to exclude a large number of non-corners. This test examines only the four pixels at 1, 9, 5 and 13 (First 1 and 9 are tested if they are too brighter or darker. If so, then checks 5 and 13). If p is a corner, then at least three of these must all be brighter than $I_p + t$ or darker than $I_p - t$. If neither of these is the case, then p cannot be a corner. The full segment test criterion can then be applied to the passed candidates by examining all pixels in the circle. This detector in itself exhibits high performance, but there are several weaknesses:

    - It does not reject as many candidates for n < 12.
    - The choice of pixels is not optimal because its efficiency depends on ordering of the questions and distribution of corner appearances.
    - Results of high-speed tests are thrown away.
    - Multiple features are detected adjacent to one another.

Machine Learning a Corner Detector

1. Select a set of images for training (preferably from the target application domain)
2. Run FAST algorithm in every images to find feature points.
3. For every feature point, store the 16 pixels around it as a vector. Do it for all the images to get feature vector P.
4. Each pixel (say x) in these 16 pixels can have one of the following three states:

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \le I_p - t & \text{(darker)} \\ s, & I_p - t < I_{p\to x} < I_p + t & \text{(similar)} \\ b, & I_p + t \le I_{p\to x} & \text{(brighter)} \end{cases}$$

5. Depending on these states, the feature vector P is subdivided into 3 subsets, P_d, P_s, P_b.
6. Define a new boolean variable, K_p, which is true if p is a corner and false otherwise.
7. Use the ID3 algorithm (decision tree classifier) to query each subset using the variable K_p for the knowledge about the true class. It selects the x which yields the most information about whether the candidate pixel is a corner, measured by the entropy of K_p.
8. This is recursively applied to all the subsets until its entropy is zero.
9. The decision tree so created is used for fast detection in other images.

Non-maximal Suppression

Detecting multiple interest points in adjacent locations is another problem. It is solved by using Non-maximum Suppression.

1. Compute a score function, V for all the detected feature points. V is the sum of absolute difference between p and 16 surrounding pixels values.
2. Consider two adjacent keypoints and compute their V values.
3. Discard the one with lower V value.

FAST Feature Detector in OpenCV

It is called as any other feature detector in OpenCV. If you want, you can specify the threshold, whether non-maximum suppression to be applied or not, the neighborhood to be used etc.

For the neighborhood, three flags are defined, `cv2.FAST_FEATURE_DETECTOR_TYPE_5_8`, `cv2.FAST_FEATURE_DETECTOR_TYPE_7_12` and `cv2.FAST_FEATURE_DETECTOR_TYPE_9_16`. Below is a simple code on how to detect and draw the FAST feature points.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('simple.jpg',0)

# Initiate FAST object with default values
fast = cv2.FastFeatureDetector()

# find and draw the keypoints
kp = fast.detect(img,None)
img2 = cv2.drawKeypoints(img, kp, color=(255,0,0))

# Print all default params
print "Threshold: ", fast.getInt('threshold')
print "nonmaxSuppression: ", fast.getBool('nonmaxSuppression')
print "neighborhood: ", fast.getInt('type')
print "Total Keypoints with nonmaxSuppression: ", len(kp)

cv2.imwrite('fast_true.png',img2)

# Disable nonmaxSuppression
fast.setBool('nonmaxSuppression',0)
kp = fast.detect(img,None)

print "Total Keypoints without nonmaxSuppression: ", len(kp)

img3 = cv2.drawKeypoints(img, kp, color=(255,0,0))

cv2.imwrite('fast_false.png',img3)
```

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_fast/py_fast.html#fast

4. **Review on Existing Industry Solutions:**
   What companies are solving similar problems to yours?

   1.DJI a drone manufacturing company, is one who is extensively working on image stabilization and also Drone stabilization. In drone DJI Inspire1, DJI worked on this problem and were successful to stabilize the drone as they were able to fly it indoors.

   2.Parrot also a drone manufacturing company were working on the same problem in the drone parrot bebop where they successfully stabilized the drone on 3 axes.

   3.The Skydio 2 shoots video with a 20mm-equivalent camera stabilized with a three-axis gimbal, but it's also got six fisheye camera eyes solely for navigation. They feed data into the Nvidia Tegra X2 processor that runs the drone's AI scene-processing software. By understanding its full surroundings, the drone can plot a course around buildings and through a forest's trees as it tracks its designated subject.

   https://www.cnet.com/news/skydios-second-gen-drone-a-1000-self-flying-action-cam-sells-out-for-2019/

5. **Dataset Collection and Visualization**

   Our dataset is just a video made by us in our phone. There are no label as we do not need to predict anything. Just from every frame, we have to just give output as how drifted our drone is. Means we have to give 6 axis value. So, we can stable the drone by giving reverse command.

What problem will your Computer Vision solution solve, and for whom?

It is simple drone stabilization problem. So, we can remove drifting effect of drone and try to keep drone stable and also give command remotely. This idea can be used by companies which are using drone for indoor surveillance. They just want to keep the drone stable at given point facing wall or anything till no command is given. Also, this can be used for weeding, security/military purpose.

How big is the potential market?

Now a days we see an increase in the use of drones in various areas so potential market for this problem of drone stabilization is very big. Also many big companies like DJI and parrot are currently trying to solve this problem and are partially successful. So getting a working algorithm for image based drone stabilization can create a good market as well.

How are potential customers dealing with these issues now?

Customers handle this issue of stabilization by controlling the drone manually to get a better image as well as to keep the drone stabilized, where as our aim is to keep the drone stabilized with the help of video feed provided to the drone by the mounted camera.

What data is available for testing and/or training algorithms?

We can generate as much data as we want.

Is labeled data available? How much? How is the data licensed? Is it under copyright protection?

We do not need label data. We can make our own data by just taking video using our phone.

If you have access to the product, what can you learn from using it?

If we get access to these products we can observe the working and stability of the drones with respect to camera in these products and get a fair idea of how with the help of a camera can a drone be stabilized also we can practically see the change in angle of drone with respect to change in angle of camera.