Target Localization for Autonomous Landing of Unmanned Aerial Vehicle

A Report Submitted
in Partial Fulfillment of the Requirements for the Course

B.Tech. Project (CS398)

by

Qazi Sajid Azam
(B16CS026)

Rahul Jindal
(B16CS027)

under the guidance of

Dr. Chiranjoy Chattopadhyay

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY JODHPUR

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Target Localization for Autonomous Landing of Unmanned Aerial Vehicle**" is a bonafide work of **Qazi Sajid Azam** (**Roll No. B16CS026**) and **Rahul Jindal** (**Roll No. B16CS027**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Jodhpur under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Chiranjoy Chattopadhyay**

Assistant Professor,

November, 2018 — Department of Computer Science & Engineering,

Jodhpur. — Indian Institute of Technology Jodhpur, Rajasthan.

Acknowledgements

We would like to thank our respected professors **Dr. Chiranjoy Chattopadhyay** and **Dr. C. Venkatesan** for guiding us during the course of this project.
We also acknowledge the institute for providing the necessary platform and resources for completing this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We have designed a system that recognizes the given target pattern in frames received from the camera. Using successive frames from the camera, we are able to calculate the position, velocity and acceleration of the target with respect to the camera. The detection is done by using feature matching with OpenCV in Python. Then we use some simple physics equations to calculate the acceleration needed to reach the target. This acceleration can then be achieved by the motor control system in the UAV. Due to unavailability of quadcopter, we have only made the system to output the required acceleration.
The code for the project was written in Python 3.6 using libraries Numpy, OpenCV, Pillow and Tkinter. The project is available on our GitHub repository at following link:
`https://github.com/QaziSajid/Feature-Matching`

Chapter 2

Literature Survey

2.1 Vision-based Autonomous Quadrotor Landing on a Moving Platform

In this paper, the authors present a quadrotor system capable of autonomously landing on a moving platform using only onboard computing which relies on state-of-the-art computer vision algorithms, multi-sensor fusion for localization of the robot, detection and motion estimation of the moving platform, and path planning for fully autonomous navigation.This system does not require any external infrastructure, such as motion-capture systems. No prior information about the location of the moving landing target is needed.

2.2 Template matching

Template Matching is a method for searching and finding the location of a template image in a larger image. OpenCV comes with a function for this purpose. It simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image. Since it is not rotation and scale invariant, it fails in our case.

2.3 Feature matching

There are several methods available on OpenCV for finding features as well as for matching. We have tried following methods:

2.3.1 ORB

ORB stands for Oriented FAST and Rotated BRIEF. It is the free alternative to SIFT and SURF. ORB is basically a fusion of FAST keypoint detector and BRIEF descriptor with many modifications to enhance the performance. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned.

2.3.2 Brute-force descriptor matcher

For each descriptor in the first set, this matcher finds the closest descriptor in the second set by trying each one. This descriptor matcher supports masking permissible matches of descriptor sets.

2.3.3 Flann-based descriptor matcher

This matcher trains itself on a descriptor collection and calls its nearest search methods to find the best matches. So, this matcher may be faster when matching a large train collection than the brute force matcher. However it need a larger train collection which is not suitable in our case.

Chapter 3

Work Done

To achieve our results, we have used the algorithm which is described below.

3.1 Detection of Target

This is the crucial and most challenging part of this project. Given a template, which is an image of the target, we need to find the same template in the image from camera. After detection, we need to find its centroid and linear size.

3.1.1 Template Matching

OpenCV provides a function for sliding window template matching. However this is not scale-rotation-skew invariant. It only works when we try to find a part of the image in the image itself. An example is shown in Figure 3.1.

3.1.2 Feature Matching

This is a more robust approach as it is rotation-scale-skew invariant. For this, we are using OpenCV in Python. we detect the keypoints in the image using ORB (Oriented FAST and Rotated BRIEF). This is a scaling and rotation invariant keypoint and descriptor detector. After finding keypoints in both images, we match them using the descriptor matcher in

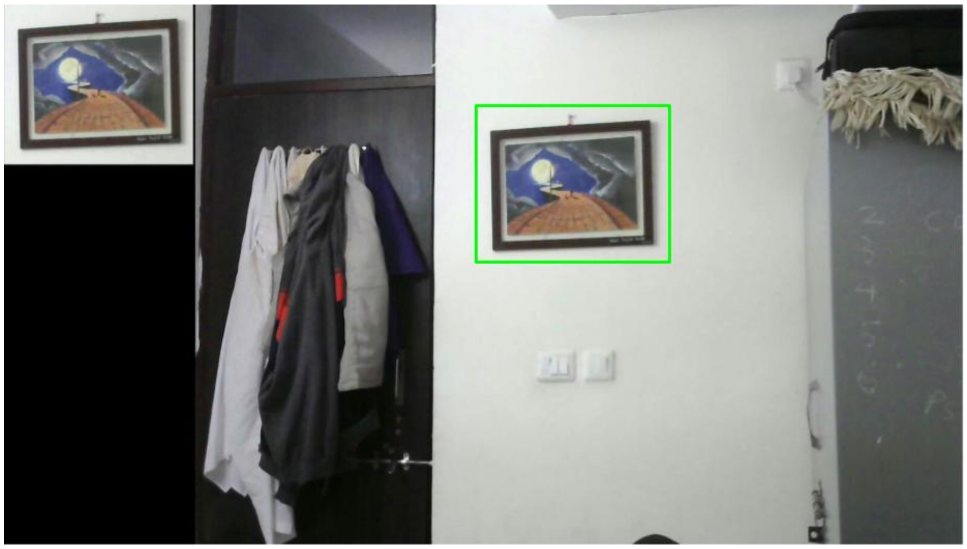

Fig. 3.1 Sliding window template matching

OpenCV and find the homography between good matched features. Good matched featured are found using top 25% of matches and then RANSAC to remove outliers. Using the homography, we are able to draw a bounding box on the detected area, and output the centroid and size for further calculation. Detected keypoints are shown in Figure 3.2 and matched features are shown in Figure 3.3. Due to limitations in the OpenCV libraries we have used for detection, the results are accurate only when the image from camera is not cluttered, as shown in Figure 3.3.

3.2 Finding the Relative Motion

We run the template detection module continuously which provides us with the centroid and linear size. From two consecutive frames, we can find the relative velocity. Using two consecutive velocities, we can find the acceleration. These are calculated in terms of pixels, and can be converted into real quantities by using ratio formulas.

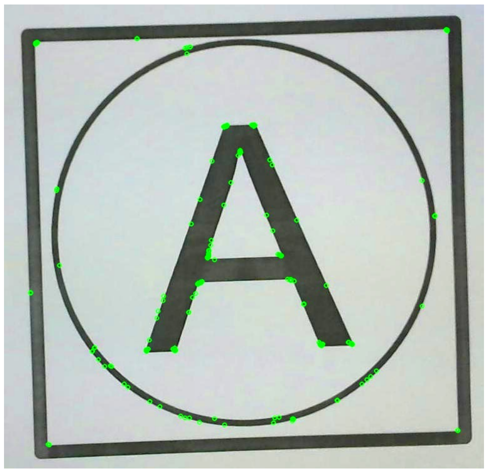

Fig. 3.2 Detected Keypoints

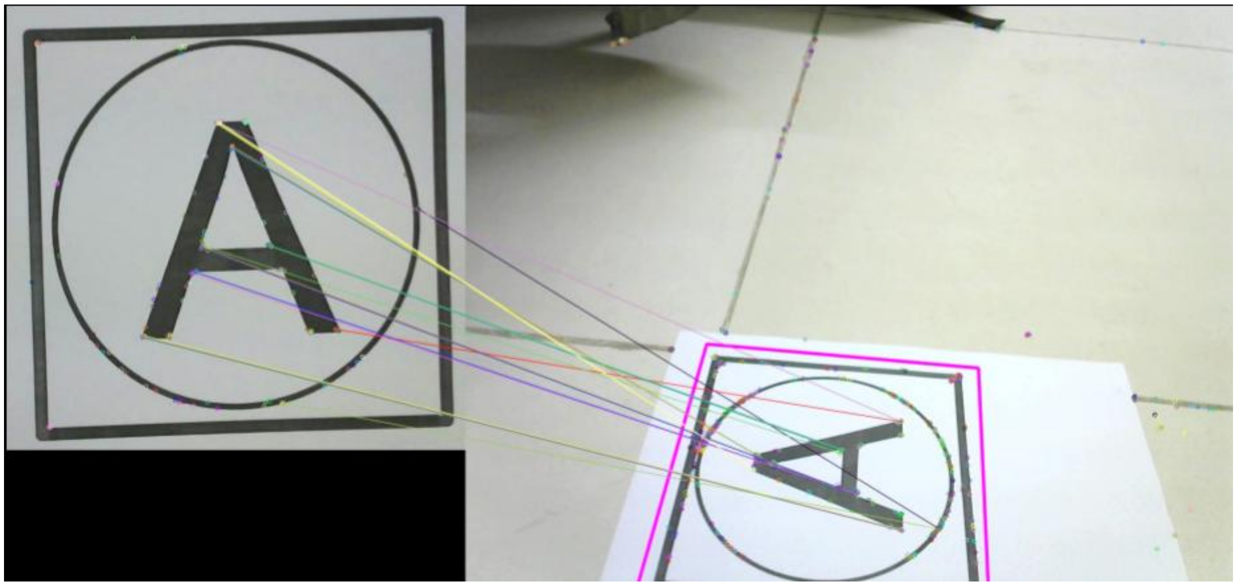

Fig. 3.3 Matching

3.3 Calculating Required Acceleration

Using the physics equations of motion, we calculate the acceleration required for reaching the target. This calculation is done continuously, and the required acceleration is also updated continuously so that controlling can be done accordingly. But due to unavailability of quadcopter in the institute, we are unable to implement it.

3.4 Improving Accuracy

In practical situations, the feature matching is not seamless and involves lot of flickering. Occasionally, it fails to detect the target correctly even if it is clearly visible. We have used some methods to improve this.

3.4.1 Validation of Bounding Box

We check that the bounding box is a closed convex polygon, and use normalized standard deviation of side lengths to eliminate potentially incorrect matches. This removes incorrect matches such as concave, complex, or overly skewed boxes.

3.4.2 Buffering

Whenever we find a successful match, we will store the coordinates. If a failure occurs less than 10 frames after the last buffered content, it will use the buffered data. This significantly improves the performance and accuracy of the system.

3.5 Calculation of Accuracy

Whenever we detect a valid match, we count it as a success. Also, if we get an invalid match, but there is still buffered content, it will count as a success. If we find an invalid match, and the buffer has expired, we will count that as a failure. If the is not visible in the frame and the buffer has expired, it means the target is not in the field of view, and we do not account for that frame in our calculation of accuracy.

Chapter 4

Results

The template matching method we are using gives an accuracy of around 80%. This is the case when we have the following assumptions:-

a) The background is uncluttered

b) The image is not blurry

c) The target is visible clearly

After that, we can show a live stream of what features are being matched and the detected target. The frame rate of capturing camera is around 25 FPS, and when we do the processing simultaneously, it runs at around 8 FPS. The accuracy from all the tests combined is 77.78%. These are the results on the laptop on which we tested the system.

4.1 Conclusion

This project can be used as a part of an automated UAV control system. Our system is invariant to scaling and rotation, and also skewing to some extent. It is also fairly accurate considering the limitations we had to deal with.

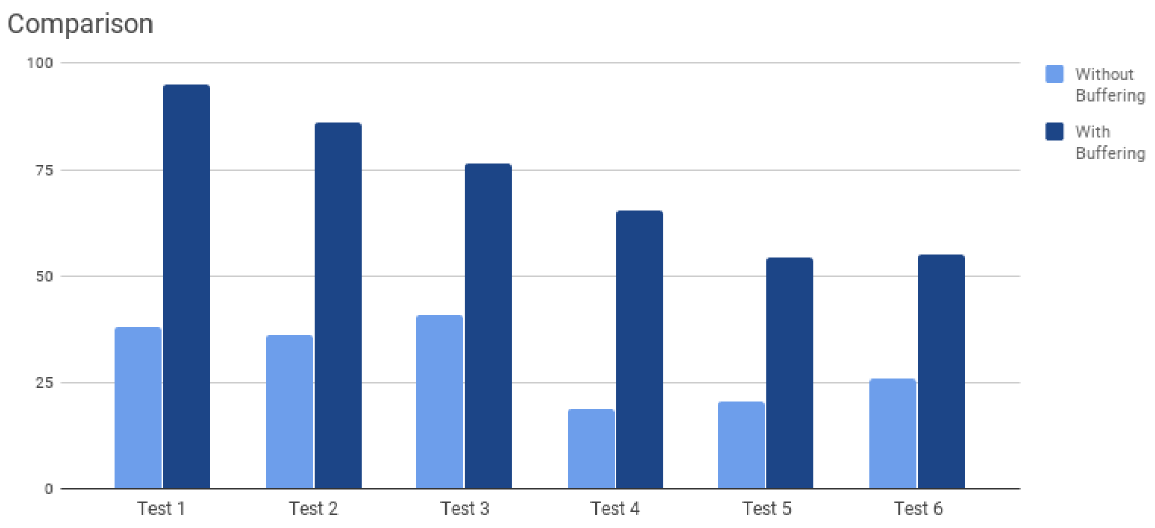

Fig. 4.1 Benefits of Buffering

S.No.	Frames	Accuracy%	Detected	Accurate	Buffered	Invalid
1	529	88	515	456	265	59
2	680	96	663	642	440	21
3	869	96	855	826	497	29
4	352	89	343	306	174	37
5	428	58	409	241	130	168
6	459	78	414	325	163	89
7	412	74	395	294	156	101
8	356	93	344	320	216	24
9	192	64	190	122	81	68
10	137	48	134	65	49	69
11	340	73	330	242	133	88
12	403	67	390	264	189	126
13	538	55	525	292	181	233
14	1231	71	1145	814	530	331
15	1147	78	1115	877	409	238

Table 4.1 Quantitative analysis of the results

Chapter 5

Future Work

The algorithm presented in this report is not so accurate, especially when the target is small compared to the background, or the image from the camera is blurry. We present some improvements that can be done by someone wanting to continue our work.

5.1 Deep Learning

The most challenging part of our project was to detect the target in the image. Deep learning can be used to solve this problem. It is a technology with huge potential, and can be applied here to improve the results. However, a dataset would be needed, which will most likely have to be prepared manually.

5.2 Energy-Efficient Trajectory Prediction

Currently we are first aligning the UAV with target so that landing can be done. A more efficient method would be descend and align simultaneously in the most efficient trajectory allowed within the physical constraints of our vehicle and while avoiding obstacles.

References

1. OpenCV Documentation (OpenCV version 3.4.3)

2. Python 3.6 Documentation

3. pyimagesearch.com

4. stackoverflow.com

5. Falanga, Davide; Zanchettin, Alessio; Simovic, Alessandro; Delmerico, Jeffrey; Scaramuzza, Davide (2017). Vision-based Autonomous Quadrotor Landing on a Moving Platform. In: IEEE/RSJ International Symposium on Safety, Security and Rescue Robotics, Shanghai, 11 October 2017 - 13 October 2017, 1-8.