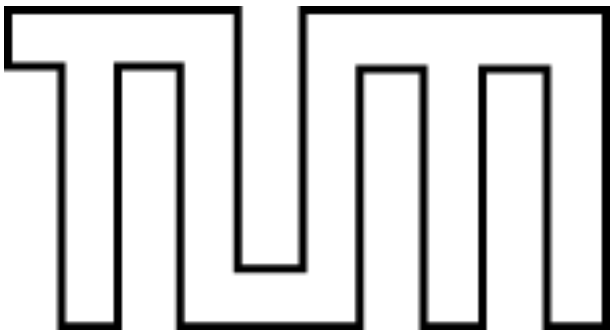

Technische Universität München

Fakultät für Informatik

Chair of Computer Vision

Photo Goal Navigation

Master-Praktikum

Amil George
Bharti Munjal
Neeraj Sujan

Advisor: Lukas von Stumberg

Date of Delivery: 9. March 2016

Contents

1 Introduction

The aim of this project is to build a vision based target navigation system for a quadcopter. Given a target image, the quadcopter has to navigate autonomously to the location from which the target image was taken. This project can be a part of larger use cases like autonomous delivery of products when combined with more information from GPS.

This report can be divided into three sections. First, we talk about the assumptions made regarding the inputs and the environment. Then, we discuss about the several approaches we took to solve the photo goal navigation problem. Finally, we describe the architecture of our application. The application is built using C++ on ROS and works with tum-ardrone package. The tum-ardrone package provides the quadcopter with monocular camera based state estimation and navigation ability. We make use of the api provided in tum-ardrone package to control the drone and send navigation commands. The application has been tested extensively in the visual navigation lab and the results look promising. The application can be further improved as it makes various assumptions discussed in the report.

2 Assumptions

We make the following assumptions in our project

1. We assume that a part of the given photo is in the view of the camera.
2. We further assume that the intrinsics of the camera from which the photo was taken and that of the quadcopter's camera are same.

The assumptions that we make are big and might not be the case in general but they provide a good starting point to our project. We can drop these assumptions in future, for example for case 1 the quadcopter can move around to bring the photo in its view.

3 Approach

The approach we took can be summarised in the following steps:

1. Estimation of point correspondences between current view and target view
2. Estimation of relative camera pose
3. Transformation of relative camera pose to pose in world coordinates
4. Sending commands to the quadcopter
5. Determine if the drone has reached the target else repeat from step 1.

3.1 Estimation of Point correspondences

Given the target view and the quadcopters view we have to find the corresponding points in the two images. Finding good correspondences is a very crucial step as the accuracy of subsequent steps in the method are dependent on this step. Before we attempt to find the correspondences we have to undistort the images. The distortion parameters were computed using camera calibration

utility provided in ROS. We initially used various feature extraction methods available from default opencv of ROS to find the correspondences namely FAST, ORB, BRIEF. In order to make this initial step better, we moved to a more robust feature detector and descriptors - SIFT using custom opencv-3 with ROS. This gave us better point correspondences and a good initial point for our algorithm.

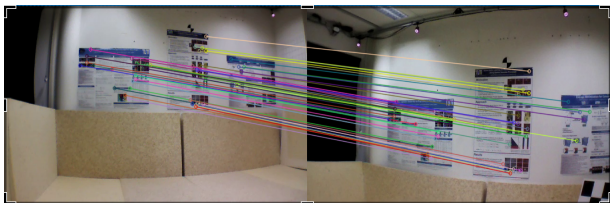

Figure 1: Point Correspondences

3.2 Estimation of Relative Camera Pose

In this step we try to estimate the target camera pose with respect to the current camera pose. The target camera pose represents the target location to which the quadcopter has to move and the current camera pose represents the current location of the quadcopter. In order to estimate the relative camera pose we have used two separate methods depending upon the drone's target environment. If the view of the quadcopter is non planar we use the fundamental matrix based estimation technique and if the view is planar we use the homography based estimation technique. This is because the fundamental matrix is degenerate when the scenes are planar.

3.2.1 Non Planar Scenes

We used epipolar geometry in order to find out the transformations between two camera frames of non planar scenes. In epipolar geometry, we can relate two images of a scene via a 3*3 rank 2 matrix called Fundamental Matrix. With homogeneous image coordinates x_L and x_R of corresponding points, F*x_L (shown as red line in Fig 2) describes the line on which the corresponding point x_R lies on the other image.

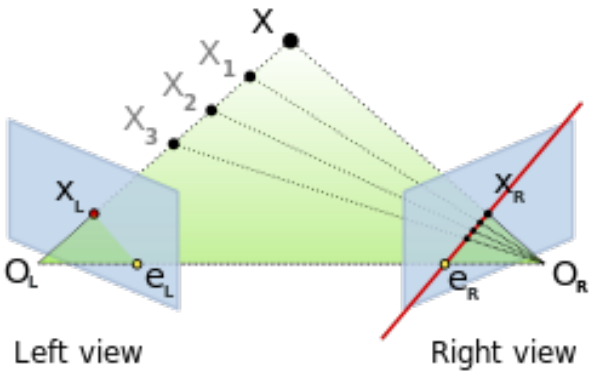

Figure 2: Epipolar

That means, for all pairs of corresponding points holds [HZ03]

$$x'_R * F * x_L = 0$$

When camera intrinsics are known, the more generic Fundamental Matrix can be used to find another rank 2 matrix called Essential Matrix.

$$E = K_1 * F * K_2$$

where K_1 and K_2 are intrinsic parameters of two cameras corresponding to both images.

3.2.2 Decomposition of Essential Matrix

To find rotation and translation from Essential Matrix, we first do an SVD of the matrix to obtain orthogonal matrices U and V [MSKS03].

$$E = U * diag\{\sigma_1, \sigma_2, \sigma_3\} * V$$

Since the reconstruction E is only defined upto scale, we project E into normalised essential space by replacing the singular values σ_1, σ_2, σ_3 with 1,1,0. The four possible options of Rotation and Translation are then computed as

$$R = UR_Z^T(\pm\frac{\pi}{2})V^T$$

$$T = UR_Z(\pm\frac{\pi}{2})\Sigma U^T$$

where R_Z is the rotation around z axis by $\pm\frac{\pi}{2}$

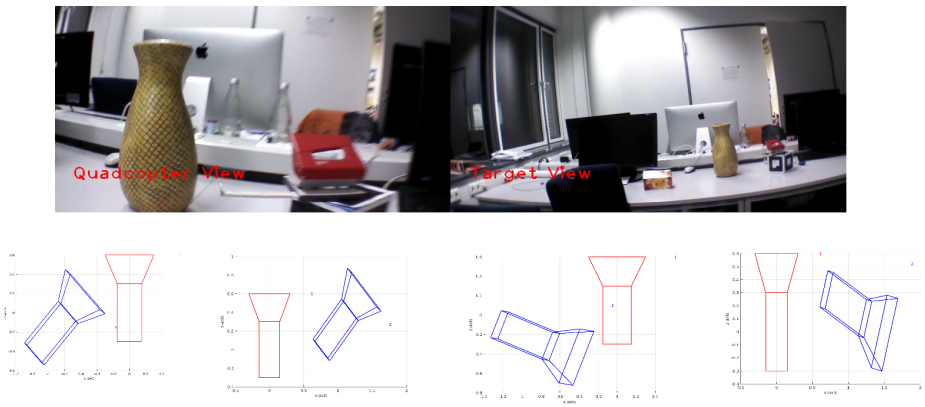

Figure 3: R and T possible options from Essential Matrix

3.2.3 Choosing the Correct Rotation and Translation

Essential matrix decomposition gives us four possible options of R and T as shown in Fig 3. We used triangulation of points to find a valid option. In triangulation, the corresponding points in images are triangulated to find their depths for each possible rotation and translation. We chose the rotation and translation with MINIMUM negative depths as the valid ones.

3.2.4 Planar Scenes

If the observed scene is planar, Fundamental Matrix can not be used to determine Rotation and Translation [TZM98]. For such scenes we can use Homography matrix that relates corresponding points p_1 and p_2 in two images. Homography can be thought of as a special case of Fundamental Matrix wherein points must lie on a plane.

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} H_{00} & H_{01} & H_{02} \\ H_{10} & H_{11} & H_{12} \\ H_{20} & H_{21} & H_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \iff p_2 = Hp_1$$

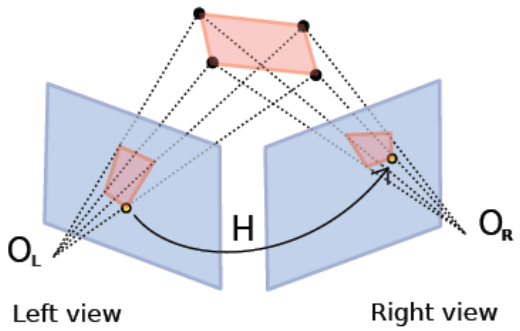

Figure 4: Homography

3.2.5 Decomposition of Homography Matrix

The homography matrix contains information for transformation between the two images as

$$H_{ba} = R - \frac{t * n^T}{d}$$

where R is the rotation matrix by which camera b is rotated in relation to a, t is translation vector from a to b, n is the normal vector of the plane and d is the distance to the plane. We used the opencv-3 decomposeHomographyMat(..) method which implements an analytical method for homography decomposition [MV07]. This gives us 4 possible solutions for rotation and translation along with their surface normals.

3.2.6 Choosing the Correct Rotation and Translation

Choosing the correct R and T from the 4 solutions obtained during the decomposition of homography matrix is not trivial as in the case of essential matrix.

In order to select the correct value of R and T from the decomposition of homography matrix usually additional sensors or assumptions regarding the environment are made use of. We assumed that the camera normal with respect to the first image i.e z-axis and the surface normals that are obtained during the decomposition of homography matrix are close to each other. The assumption would hold true for cases where the quadcopter is not at a large yaw angle to the planar surface considered.

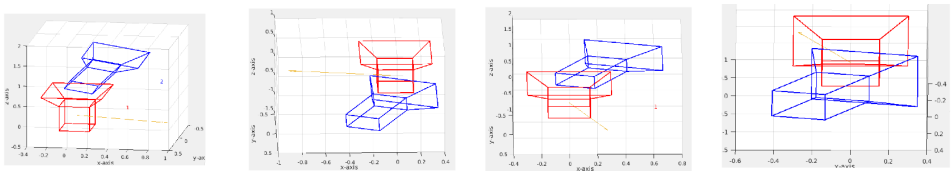

Figure 5: R and T options from Homography Decomposition. Red is the initial camera, blue is the relative transformation of the second camera and yellow line is the surface normal

3.3 Coordinate Transformation

R and T that we obtained so far are in a frame (vision frame) different from the quadcopter's co-ordinate frame. We have to convert them to the correct quadcopter's frame and then to the world frame.

We have the following coordinate systems.

1. World Coordinate System: z -"up", x -"right", y -"front"
2. Drone Coordinate System (Origin is at drone center): z -"up", x -"right", y -"front"
3. Vision Coordinate System (from Homography or Essential Matrix): z -"front", x -"right", y -"down"

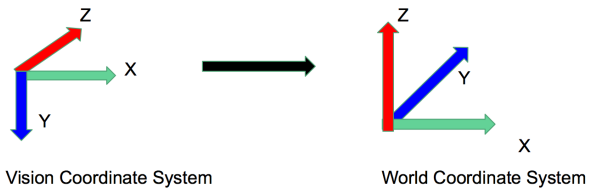

Figure 6: Coordinate Transformation

The transformation from drone coordinates to global coordinates $T_{droneToglobal}$ is the predicted pose obtained from tum-ardrone.The transformation from global to drone can be defined as

$$T_{globalTodrone} = T^{-1}_{droneToglobal}$$

The transformation $T_{droneTovision}$ from drone to vision is rotation around x axis by $\frac{\pi}{2}$. Finally the transformation from global to vision can be found as

$$T_{globalTovision} = T_{droneTovision} * T_{globalTodrone}$$

$$T_{visionToglobal} = T^{-1}_{globalTovision}$$

In order to get the target camera position in world coordinates, we first compute the pose of the target camera with respect to the current vison coordinate system of the drone and then multiply it with the transformation $T_{visionToglobal}$.

3.4 Interaction with tum-ardrone

After determining the final position of the quadcopter from the above mentioned transformations, we send these commands to quadcopter via tum-ardrone. tum-ardrone provides APIs to move the quadcopter absolute or relative (goto, moveBy, moveByRel). We used the goto command and published the obtained position and yaw to the required topic.

3.5 Momentum Control

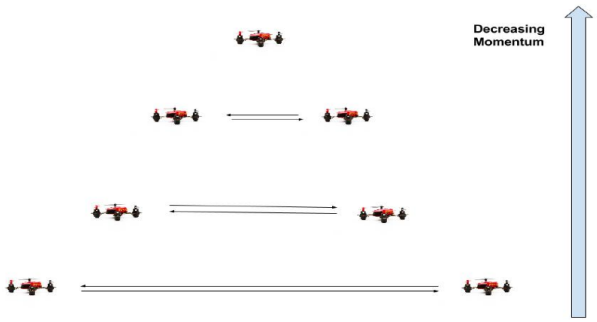

Figure 7: Momentum Control

The rotation and translation estimated using the above methods are scale independent. Since we do not have the scale we will not be able to reach the final target location in just one iteration. For example, we may obtain the same translation [1 0 0] if we are 10 units or .001 units away from the target. Moreover, this problem makes it difficult for us to determine if the quadcopter has reached the final target location.

In our approach we normalize the translation obtained which is then multiplied with an empirically chosen initial momentum. Momentum is chosen as a three vector along x,y and z directions. Now if the quadcopter moves in the same direction in consecutive steps we increase the momentum in the corresponding direction by a constant factor. If the quadcopter moves in opposite direction in consecutive steps we decrease the momentum and continue again to estimate the rotation and translation. Following this process when the quadcopter is near to the target we assume that the quadcopter is going to move in a zig zag manner which will reduce the momentum vector to a small value. When the momentum is sufficiently small we assume that the quadcopter has reached the final location.

4 Architecture

The architecture of our application mainly consist of a controller that is responsible for interacting with the GUI Manager and tum-ardrone.

The controller interacts (subscribes/publishes) with the following ros-topics of tum-ardrone

1. */ardrone/front/image_raw* to receive quadcopter's current pose
2. */ardrone/predictedPose* to receive the current pose of the quadcopter
3. */tum_ardrone/com* to send goto command to tum-ardrone

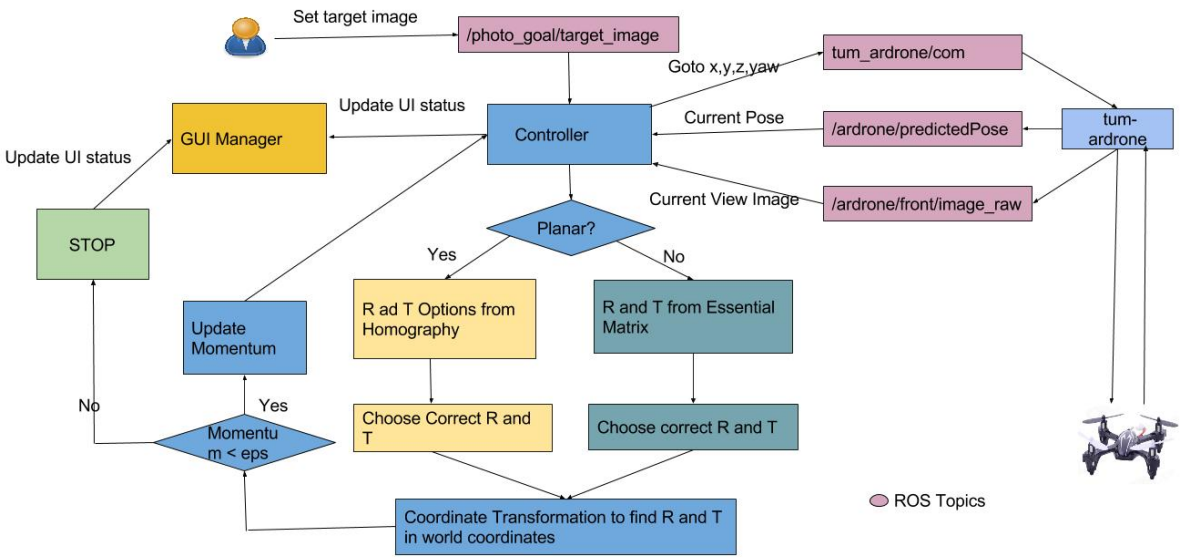

Figure 8: Application Architecture

The controller receives the current pose and view of the quadcopter continuously from tum-ardrone. The user inputs the target image to the controller via topic /photo_goal/target_image. Once the controller receives the target image, it tries to find out the relative transformation from the quadcopter's current pose to target view using one of the methods as shown in the Figure 8. Once the controller estimates the new position and yaw of the quadcopter in world coordinates, it checks the current momentum to verify if the related command has to be sent to tum-ardrone. If the momentum is below a fix value, the command is not sent and the target is achieved otherwise the command is sent. The controller then waits for the command sent to be executed in the tum-ardrone queue. The tum-ardrone itself does not provide any api to know the current status of a command. To verify if the command has been executed or not the controller compares the current pose received with the last command sent. If both values are close enough, command is assumed to executed and the controller can run next iteration for the current view.

5 Graphical User Interface

In order to display important information to the user we made a simple interface that is controlled by GUI Manager that receives input from the application controller. The interface displays the following (Fig 9)

1. Target View and Current View
2. Last goto command Status (Sent/Executed)
3. Target Achieved or not

Figure 9: GUI

6 Code

The implementation of the project can be found at the repository of our group https://github.com/amilgeorge/PhotoGoal

7 Future Work

1. Ability to support target images taken with a camera other than the quadcopter camera. In this case we have separate intrinsic parameters K_1 and K_2 for the current quadcopter view and target quadcopter view.

2. Automatic detection of planar scenes and non planar scenes. An easy way to implement this is to compute the homography assuming the scenes are planar and use the reprojection error to determine if the scene is planar or non planar.

3. Ability to navigate to target scenes outside the field of view of the quadcopter

References

[HZ03] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[MSKS03] Yi Ma, Stefano Soatto, Jana Kosecka, and S. Shankar Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. SpringerVerlag, 2003.

[MV07] Ezio Malis and Manuel Vargas. Deeper understanding of the homography decomposition for vision-based control. 2007.

[TZM98] Philip HS Torr, Andrew Zisserman, and Stephen J Maybank. Robust detection of degenerate configurations while estimating the fundamental matrix. *Computer Vision and Image Understanding*, 71(3):312–333, 1998.