UNIVERSITA' DI PISA

Facoltá di Ingegneria

Laurea Magistrale in Ingegneria Robotica e dell'Automazione

Tesi di Laurea

Development of a Visual Navigation System for Multirotor Vehicles

Candidato:
Stefano Aringhieri

Relatori:
Prof. Lorenzo Pollini

Controrelatore:
Prof. Mario Innocenti

Anno accademico 2013/2014

"Un vincitore è solo un sognatore
che non si è arreso"
Nelson Mandela

Abstract

In this thesis it has been proposed a Visual Navigation System for Multirotor Vehicles, such as UAV (Unmanned Aerial Vehicle) or MAV (Micro Air Vehicle), equipped with a stereo camera and onboard IMU unit. The set of available sensors is determined by the choice to partecipate at the European contest issued by EuRoC. Cameras have infact the advantage that even small and lightweight versions can capture images of acceptable quality. Furthermore, those images can be acquired at a high enough frequency, so that the acquisition of data is surely never going to be the bottle neck of the control system.

From the IMU data has been realized a Navigagtion Inertial System, too. Finally, the attitude estimation comes from the last one Navigation System is used to correct the Visual Odometry and to implement a Integrated Visual Navigation System.

In order to test the visual navigation system proposed we have realized tests in a virtual simulation environment, using the meta-operating system ROS and Gazebo and with real data, which consist of a pair of stereo images and imu data, provided by EuRoC.

Contents

1 Introduction

In the recent years a lot of research has been directed at the field of Unmanned Aerial Vehicles (UAVs). The great success and interest for the UAVs comes from they allowing easy access to places where, for some reason or another, humans cannot go without a large effort or without exposure to hazards, or even places a human cannot reach at all. One can think of many scenarios such as search and rescue missions after a disaster or fire, inspection of either very small and narrow tunnels or very large rooms, surveillance and observation tasks and so on. Many of those tasks are most easily accomplished by small flying robots.

The UAV can either be teleoperated by a human operator, or operate completely autonomous. While the latter is obviously still a lot more challenging, even teleoperated flight poses many challenges. UAVs are highly dynamical systems which are not easily controlled by a human operator even if they are close by. Furthermore for many of the previously mentioned tasks, the operator might not be able to directly see the UAV, which only makes controlling the helicopter more difficult.

The task of this thesis results from an application where a UAV will be used to plant servicing and inspection. The given task could include some additional difficulties. For example, the surface of the industrial plant might not offer many distinctive features or patterns that can be used for visual navigation, and the inspection task might require the MAV to hover so perfectly still that a human operator will never be able to achieve this.

It will therefore be necessary to find suitable sensors to work in the given environment and to control the UAV to at least provide stable hovering.

Controlling UAVs is a difficult task, first and foremost because of the characteristics of the vehicle itself. The most important difference to any non-flying robot is, that a UAV is always in motion. Unlike wheeled robots, a helicopter cannot just stand still, wait, take some more measurements and compute the best strategy, and only then move on. Instead, while computing the next controller output, the UAV still moves and maybe even renders the control action it is currently computing invalid.

As a second main problem, UAVs have a very limited power supply on board, and most of this power is used to just keep the UAV in the air. This makes it difficult to add many sensors to the vehicle and provide sufficient computational power to perform sophisticated calculations on board. Instead, few and lightweight sensors have to be used, which usually leads to lower quality of the measurements.

Thirdly UAVs are usually underactuated, which means that the six degrees of freedom (DOF) have to be controlled by less than six independent actuators. For example, a helicopter has to pitch its nose downward in order to move forward. This also influences the sensor readings such as the field of view of cameras and therefore makes control more difficult. Most UAVs have extremely fast and underdamped dynamics, which quickly leads to oscillations unless a suitable con- troller including a derivative part is used. This however requires precise estimates of not only the current position and attitude, but also the current velocity, with a high update rate.

Because of those constraints it is interesting to investigate the use of cameras to control the UAV. Cameras have the advantage that even small and lightweight versions can capture images of acceptable quality. Furthermore, those images can be acquired at a high enough frequency, so that the acquisition of data is surely never going to be the bottle neck of the control system. Acquiring images also yields a huge amount of data that can be used to extract whatever information is needed for control. On the other hand, images also include a lot of unnecessary or even unwanted pieces of information. It may be difficult to find the relevant information within the images because of the lack of distinctive features or patterns on the walls.

1.1 Project Outline

The goal of this thesis was to investigate the usage of stereo vision to control a UAV, equipped also with onboard IMU sensor, consisting of a 3-axis linear accelerometer and a 3-axis gyroscope. In addition in order to test the visual navigation system proposed we have realized tests in a virtual simulation environment, using the meta-operating system ROS and Gazebo, and with real data, which consist of a pair of stereo images and imu data, provided by EuRoC [8].

The work consisted of the following main tasks:

1. Choose a strategy to use the images to control the UAV.
2. Integrate the Visual Odometry with the Inertial Navigation System in order to obtain a Integrated Visual Navigation System.
3. Evaluate the Navigation System proposed in the simulation enviroment and with real data.

The framework was programmed using C++ on a computer running Ubuntu linux 12.04. It makes extensive use of the meta-operating system ROS, the open source computer vision library openCV, and the Eigen library for linear algebra, matrix and vector operations, and numerical solvers.

This report is structured as follows:

- Chapter 1: prerequisities of the camera geometry, and overwiew about ROS.
- Chapter 2: presentation and evaluation of the navigation system.

1.2 Notation

Throughout this report, the following notation is used:

- 2D point vector: $\mathbf{x} = \left[\begin{array}{cc} x & y \end{array}\right]^T$or $\left[\begin{array}{ccc} x & y & 1 \end{array}\right]^T$for homogeneous coordinates.
- 3D point vector: $\mathbf{X} = \left[\begin{array}{ccc} X & Y & Z \end{array}\right]^T$ or $\left[\begin{array}{cccc} X & Y & Z & 1 \end{array}\right]^T$for homogeneous coordinates.
- Matrix: $\underline{A}$.
- Identity Matrix: $\underline{I}$

It will always be noted if homogeneous coordinates are used.

The ^ operator denotes the 3x3 skew-symmetric matrix that represents the cross- product:

$$\mathbf{X}_1 \times \mathbf{X}_2 = \hat{\mathbf{X}}_1 \cdot \mathbf{X}_2 = \left[\begin{array}{ccc} 0 & -Z_1 & Y_1 \\ Z_1 & 0 & -X_1 \\ -Y_1 & X_1 & 0 \end{array}\right] \cdot \mathbf{X}_2$$

Part I
Prerequisities

2 Camera Geometry in Computer Vision

2.1 Ideal Pinhole Camera

In order to use the cameras for control, it will be necessary to reconstruct 3D points of the scene from the 2D images. Therefore a relation that maps 2D image pixel coordinates to the 3D real world coordinates is needed. This relation can be approximated by starting at the ideal pinhole camera model and then transforming the model to the needed form.

Let we see the following figure.

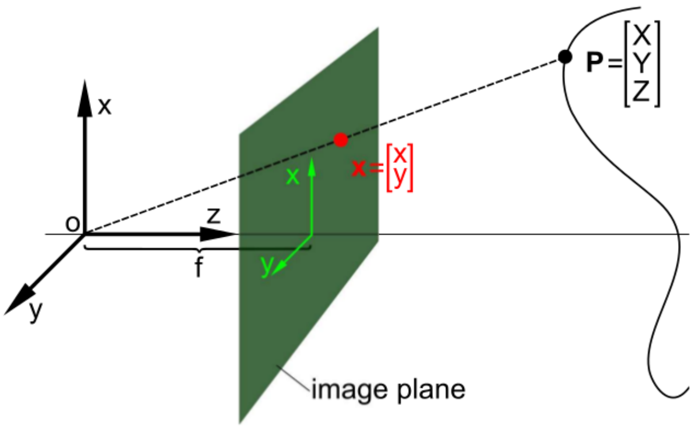


Figure 1: The Pinhole camera model depicted with the image plane placed in front of the camera.

The Figure 1 shows the pinhole camera model. Note that the image plane has been moved to the front because it is more convenient. This can be done without loss of generality. Point $\mathbf{P} = [\ X \quad Y \quad Z\ ]^T$ in the real world is projected to an image point $\mathbf{x} = [\ x \quad y\ ]^T$ according to:

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{1}$$

where f is the focal lenght of the camera. In preparation for the following transformations, the equation can be rewritten to include a camera matrix $\underline{K}_f$ and a projection matrix $\underline{P}_0$:

$$Z \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \underline{K}_f \cdot \underline{P}_0 \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

2.2 Extrinsic and Intrinsic Parameters

In the previous camera model, the origin of the world was placed at the camera center. This is often not the case for many real applications including the one under investigation in this text. Since our cameras move with the UAV, the camera coor- dinate frame will obviously not coincide with the real world fixed frame at all times .

The point $\mathbf{X}_0$ in the fixed world coordinate frame is transformed to the camera frame by the transformation $\underline{T}_{0C}$. If the transformation $\underline{T}_C$ transformed the camera frame from the world frame to its current position, then the transformation $\underline{T}_{0C}$ can be written as $\underline{T}_{0C} = \underline{T}_C^{-1}$, again using homogeneous coordinates.

Therefore, the following equation shows the relation between the point $\mathbf{X}_C$ with respect to the camera frame and the point $\mathbf{X}_0$ w.r.t. the world coordinate frame.

$$\mathbf{X}_C = \underline{T}_{0C} \cdot \mathbf{X}_0 = \begin{bmatrix} \underline{R}_C^T & \mathbf{T} \\ 0 & 1 \end{bmatrix} \cdot \mathbf{X}_0 = \begin{bmatrix} \underline{R} & \mathbf{T} \\ 0 & 1 \end{bmatrix} \cdot \mathbf{X}_0 \tag{3}$$

The 3x3 matrix $\underline{R}$ and the 3x1 vector $\mathbf{T}$ are called the extrinsic camera parameters. Inserting equation (2) into equation (3) leads to an equation that maps image plane coordinates to 3D real world coordinates. In that equation, the depth is often replaced by a constant factor λ because it is usually unknown as the pinhole camera only allows reconstruction up to a scaling factor.

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \underline{R} & \mathbf{T} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{bmatrix} \tag{4}$$

$$\mathbf{x}_2'^T \underline{F} \mathbf{x}_1' = 0$$

Using the abbreviations introduced above, equation (4) can be written as :

$$\lambda \mathbf{x} = \underline{K}_f \cdot \underline{P}_0 \cdot \underline{T}_{0C} \cdot \mathbf{X}_0 \tag{5}$$

In order to get usable equations, the current image plane coordinates x and y have to be mapped to image pixel coordinates, because the camera is built up of many small photo sensors of a given size and shape, each yielding the image brightness at the corresponding location. This compensation has to account for three effects :

- Pixels might not be square. The coordinates in x-direction are scaled by s_x and the coordinates in y-direction are scaled by s_y.
- Pixels might also not be rectangular, which is compensated by the shearing factor $s_\theta = cot(\theta)$.
- For cameras with chips that are read line by line from left to right, the image principal point has to be shifted from the top left corner to the image center. Other transformations might be possible for other builds, but they can all be summarized by adding an offset $\begin{bmatrix} o_x & o_y \end{bmatrix}^T$ in both x- and y-directions respectively.

Summarizing all of the above, one can write the mapping from image plane coordinates to pixel coordinates as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & s_\theta \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} o_x \\ o_y \end{bmatrix} \tag{6}$$

With equation (6) and by using homogeneous coordinates again, the camera matrix $\underline{K}_f$ can be extended to the intrinsic camera matrix $\underline{K}$:

on

$$\underline{K} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} fs_x & fs_\theta & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

Using the intrinsic camera matrix $\underline{K}$, equation (5) can be updated to map from pixel coordinates to 3D real world coordinates:

$$\lambda \mathbf{x}' = \underline{K} \cdot \underline{P}_0 \cdot \underline{T}_{0C} \cdot \mathbf{X}_0 \tag{8}$$

which can be simplified to

$$\lambda \mathbf{x}' = \underline{P} \cdot \mathbf{X}_0 \tag{9}$$

with $\underline{P} = \underline{K} \cdot \underline{P}_0 \cdot \underline{T}_{0C} = [\ \underline{K}\underline{R}\ \ \underline{K}\mathbf{T}\ ]$

Equation (9) fully describes the mapping from a 3D real world point to a 2D pixel coordinate point for the ideal pinhole camera model. However, in a real camera, lenses have to be used. Depending on the design and quality of the lenses, the resulting image will be more or less distorted. For example, narrow field of view lenses tend to impose smaller distortions than wide field of view lenses. In order to use the images for 3D reconstruction, it is absolutely necessary to com- pensate for those distortions, because otherwise finding valid correspondences will be more of a gamble rather than a well defines process. Therefore, the projected image plane coordinates of a real world point have to be transformed before multiplying them with the intrinsic camera matrix. Usually, a distortion model including radial and tangential distortions is used, and the transformation from distorted image plane coordinates x to undistorted image plane coordinates xd is given by (see [1] page 392):

$$\mathbf{x}_d = \mathbf{x} \cdot (1 + d_1 r^2 + d_2 r^4 [+d_5 r^6]) + \begin{bmatrix} 2d_3 xy + d_4(r^2 + 2x^2) \\ d_3(r^2 + 2y^2) + 2d_4 xy \end{bmatrix} \tag{10}$$

with $r^2 = x^2 + y^2$. The first contrinute is teh radial distorsion, the second one the tangential distorsion.

Usually, the distortion parameters are summarized to a vector $\mathbf{D}$. d_5 is sometimes omitted, depending on how much accuracy is needed. In order to use the model given in equation (10) to undistort an image, nonlinear equations have to be solved. Typically, this cannot be done fast and efficient enough to run the undistortion at real time, however, it is possible to precompute a lookup table that can then be used to remap the images in real time. Using equations (10) and (9) thus allows to go from pixel coordinates to 3D real world coordinates while compensating for lens distortions.

2.3 Stereo Camera : Epipolar Geometry

Geometrical relations between two camera views can be described by epipolar geometry. Using epipolar relations will later help to find correspondences, remove outliers and test for accuracy. The geometrical relations are true for either two different views of one single camera, or for two cameras of a stereo vision setup. For the following explanation, the first view will be named camera1 and the second view camera2, with the corresponding intrinsic camera matrices $\underline{K}_1$ and $\underline{K}_2$ respectively. In this thesis, camera1 was the left camera and camera2 the right camera of the stereo setup.

The figure 2 shows the basic setup with two camera views. In both views, the real world point $\mathbf{P}$ is captured in the image at positions $\mathbf{x}_1$ and $\mathbf{x}_2$ respectively. From the drawing in the figure it can be seen that the projection $\mathbf{x}_2$of $\mathbf{P}$ in the second view can only lie on the line l_2 if the points $\mathbf{x}_1$ and $\mathbf{x}_2$ indeed are both projections of the same point $\mathbf{P}$ - and vice versa for point $\mathbf{x}_1$. The lines l_1 and l_2 are called the epipolar lines and the points e_1 and e_2 are called epipoles. The epipoles are located at the two points where the image plane intersects with the baseline between the two views. Note that if the optical axes of the two views are parallel, this intersection

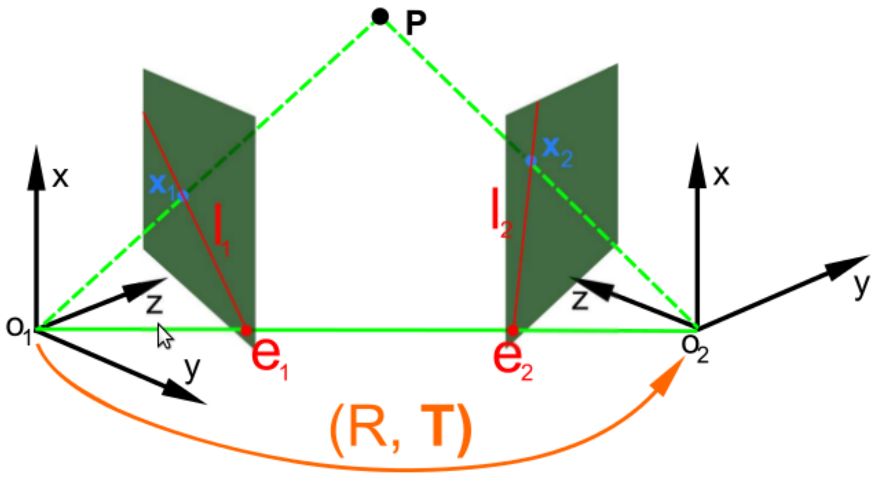


Figure 2: Setup of two camera views with epipolar geometry.

will lie at infinity and the epipolar lines will be parallel.

The geometric constraints shown in the image lead, after some rewriting, too the well known epipolar constraint, which must be satisfied for all corresponding $\mathbf{x}_1$,$\mathbf{x}_2$ and $\mathbf{P}$:

$$\mathbf{x}_2^T \underline{E} \mathbf{x}_1 = 0 \tag{11}$$

with $\underline{E} = \underline{\widehat{\mathbf{T}} R}$.

$\underline{E}$ is called the essential matrix, and it encodes the relative pose of the camera(s) from one view to the other. The previous equation (11) uses image plane coordintes, but it can be transformed to use pixel coordinates by using the previously introduced intrinsic camera matrices $\underline{K}_1$and $\underline{K}_2$:

$$\mathbf{x'}_2^T \underline{F} \mathbf{x}_1' = 0 \tag{12}$$

with $\underline{F} = \underline{K}_2^{-T} \hat{\mathbf{T}} \underline{R} \underline{K}_1^{-1} = \underline{K}_2^{-T} \underline{E} \underline{K}_1^{-1}$.

$\underline{F}$ is called the fundamental matrix and it contains the intrinsic camera properties.

2.4 Rectification

tereo image rectification refers to a process where the two images of a stereo camera pair are undistorted, projected to the same plane and transformed such that their epipolar lines are parallel and exactly row-aligned. Mathematical details about the process can be found in [1] starting at page 430. Figure 3 shows the four steps needed to rectify a stereo image pair:

1. Two images of the scene are taken by the right and left cameras.
2. The images are undistorted using the distortion approximation described above in section 2.2.
3. Now the transformation ($\underline{R}$, $\mathbf{T}$) that describes the relative pose of one camera to the other, is used to rectify the images. That is of course, if that relation is known. In this thesis, we use calibrated cameras, and therefore it is indeed known. There is also a solution to the problem in case this transformation is unknown.
4. The image is cropped because the warping creates curved edges that might interfere with image segmentation. After the cropping, only the parts of the image remain where no artifacts of the warping process are visible.

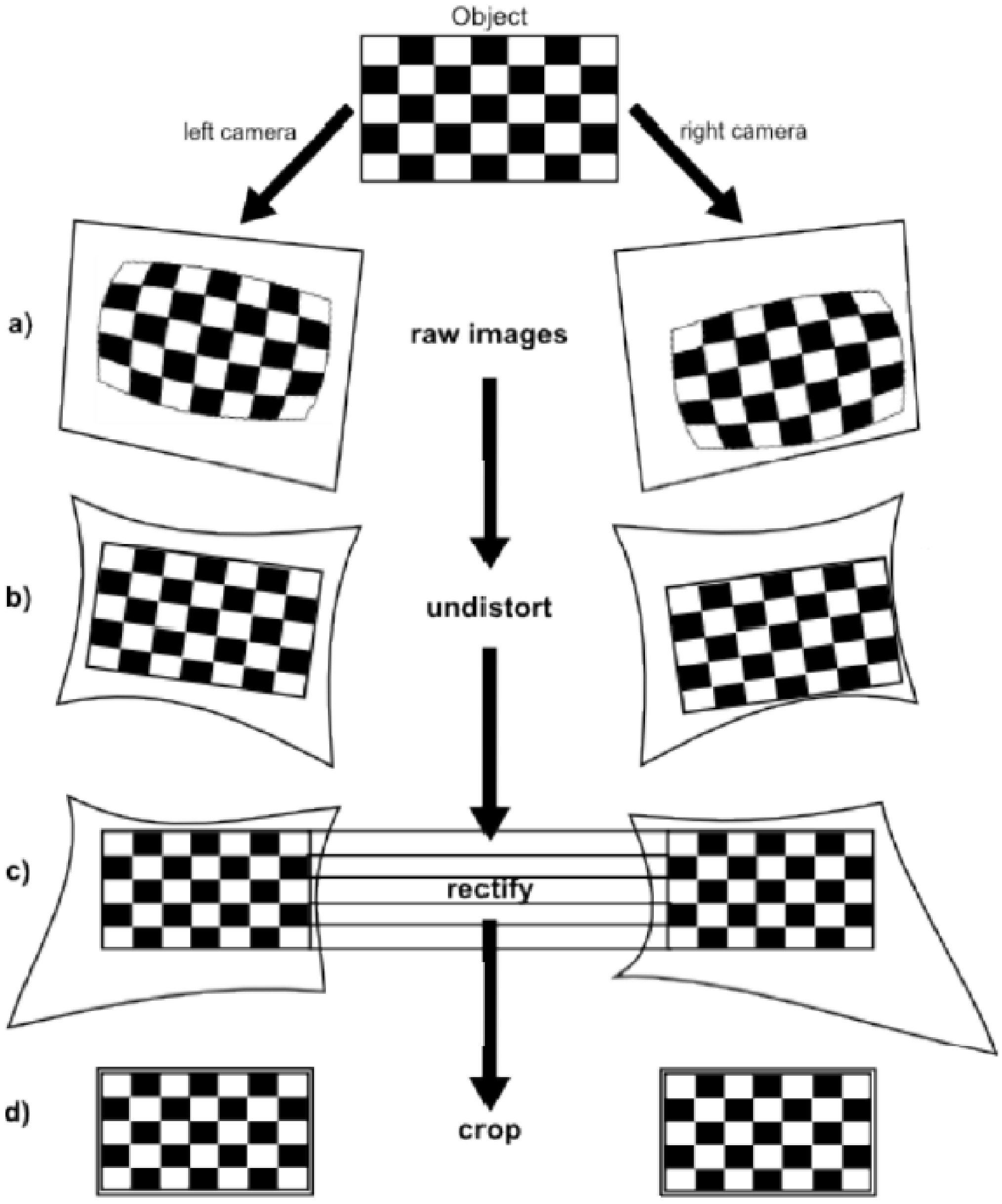


Figure 3: Schematic of the stereo rectification process: a) take images b) undistort the images c) rectify to row-aligned images projected to the same plane d) crop .

Now we have all the prerequisites for implementing a pose estimation using a stereo camera.

3 Robot Operating System(ROS)

One of the key components of the framework is the meta-operating system ROS [5]. ROS is, in the words of its developers, *"an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management."*

The main advantage of using ROS is the much improved modularity. ROS allows the user to write so-called nodes, which are programs written in either C/C++ or Python. Those nodes can send and receive messages, and thereby communicate with one another. One key advantage is, that a node can be coded to receive mes- sages, but it will also run when no such message is there to be received. On the other hand, a node can send messages without the need to know if another node is actually receiving them. This allows the user to add and remove nodes to the current setup at will, which makes testing, plotting, visualizing and verifying data a lot more comfortable and fast.

A second advantage of using ROS are the many built-in nodes that come with the basic installation. A lot of very useful tools for robotics have been created by the ROS community, which can significantly speed up testing and/or developing new ideas. For example, the framework currently uses the ROS package "stereo image proc" to modify the camera images. The same thing can of course be achieved by writing code oneself - in this case using openCV functionality - but the implementation is already available in high quality.

3.1 Framework overwiew

In this thesis we use ROS to test the visual estimation algorithm on a dataset provided by EuRoC Project, and to realize a simulation enviroment using Gazebo too. Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, and perform regression testing using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. At your fingertips is a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces.

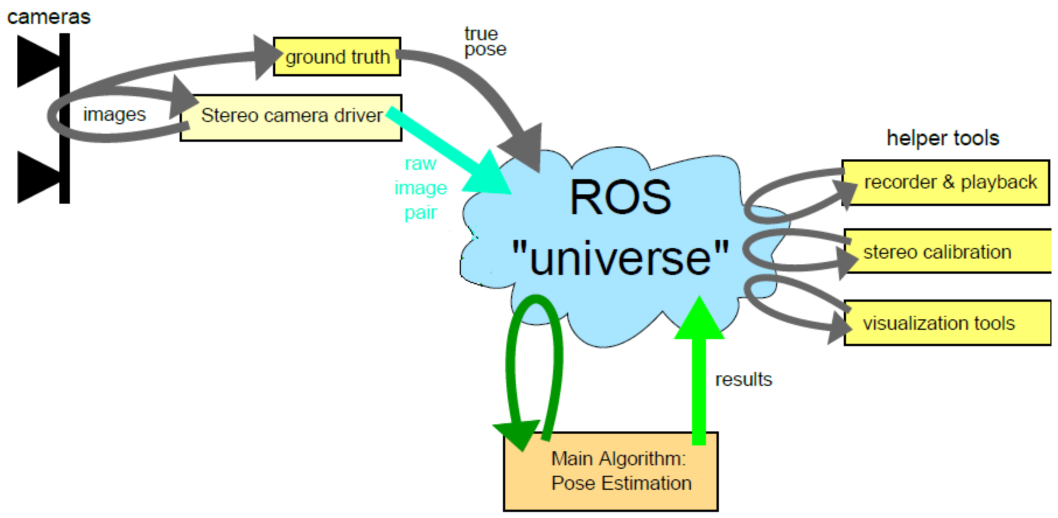


Figure 4: Framework: using ROS to connect the necessary and mandatory components depending on the current task

Figure 4 shows how ROS is used to connect two or more nodes. This connection is called the framework in this thesis, because this is the main structure that allows modular testing. It will always include one node which feeds images into the ROS 'universe' and the node running the main pose estimation algorithm. In addition, there are many helper nodes to save images, feed saved images back to simulate the cameras, visualize data by either plotting it, saving it to files . For testing, in the simulation enviroment a node running a program to determine ground truth can be added, so that the performance of the pose estimation can be verified.

Part II
Navigation System

4 Introduction

Navigation systems are used for land, sea, airborne, and space vehicles. These systems provide an operator and/or control system with the necessary information to effect some action in response to data provided by these systems. For example, this action can be a course correction indication for an aircraft pilot or a feedback control signal to guide an autonomous vehicle. These systems incorporate onboard sensors coupled with a computer, permitting self-contained operations with little or no assistance required from sources external to the vehicle.

The core of the navigation system is a set of sensors combined with a computer that can provide a relatively stable and accurate source of navigation. These systems output navigation state data, which usually include position, velocity, and attitude. As a result of imperfections in navigation sensors and computational errors, errors develop in the navigation state data and grow in time. Tht: host vehicle's operating environment also influences the error growth rate. Long-term error growth is minimized by including other sensors that provide independent redun- dant navigation data, i.e., position, in an integrated system that optimally combines this independent data source with the core navigation system. These independent sensors, referred to as navigation aids, are characterized by long-term error stability, which can compliment the short-term error stability of the navigation system's sensors. When combined within a computer algorithm, such as a Kalman filter, errors from both the core sensors and navigation aids can be estimated to reduce the integrated navigation system's errors. The resulting navigation system will exhibit improved performance, even if independent data are used intermittently or are not available for a short timespan.

The majority of navigation systems are mechanized with accelerometer and gyro inertial sensors. These inertial sensors provide sensed accelerations (velocity changes over a time interval) and rates (attitude changes over a time interval). Accelerometers and gyros are mounted in orthogonal triad clusters and enclosed within an inertial measurement unit (IMU) to provide three components of acceleration and rate outputs. These outputs are provided to a computer-implemented numerical integration process that computes a navigation solution yielding a complete set of navigation state data, i.e., position, velocity, and attitude. These mechanizations are generally referred to as an inertial navigation unit when enclosed within a case that can be easily removed and replaced. Implementations that include the inertial sensors, computer, and navigation aids are referred to as an inertial navigation system (INS).

4.1 Frame of References

The navigation system's core sensors provide information as a result of movement. These sensors are fixed to the vehicle, i.e., a strapdown IMU. Therefore, these sensors provide information about the movement of the vehicle as reference to the vehicle's frame of reference, called body frame F_b. The information in this reference frame may not be useful to a pilot or guidance system. Therefore, a navigation solution is established in a reference frame, the navigation frame F_n , that allows its data to be used conveniently and allows data from other sources, i.e., navigation aids, to be easily incorporated. This is accomplished by establishing a navigation frame that is relatively fixed.

The literature indicates a variety of navigation system reference frames that have been used in integrated navigation systems. Examples include inertial (stellar referenced), Earth-referenced (north-referenced azimuth), and wander azimuth (free azimuth movement). Even within these examples, there are additional levels of definition, e.g., which axis is aligned with what direction and what order one axis is rotated with respect to another. The navigation frame selection can be arbitrary, at the discretion of the designer, or the frame's definition may be specified.

In this thesis we introduce the following reference systems:

- A Local Geodetic Frame, or Navigation frame F_n.
- A body frame, F_b
- A frame associated with the IMU sensor, F_{imu}.
- A camera frame, F_{cam}.

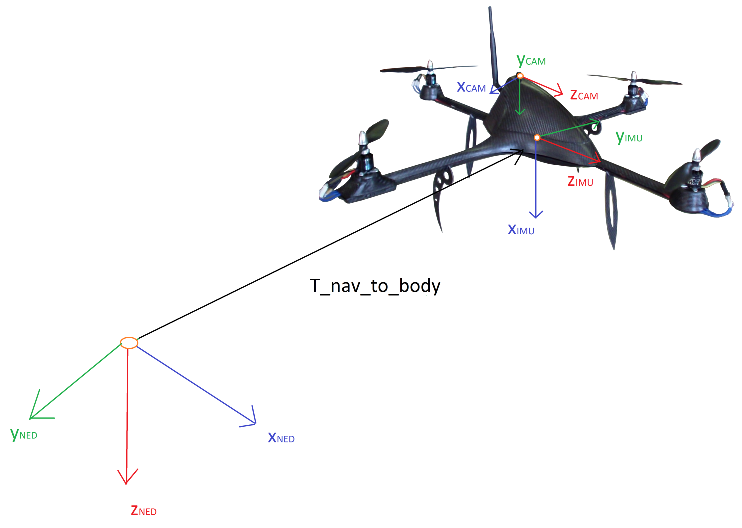


Figure 5: Schematic rapresentations of reference systems used (NED frame, IMU frame, Camera Frame).

For the Navigation system, we use the NED (North-East-Down) Frame.

The body frame is rigidly attached to and defined within the vehicle carrying the navigation system. In this thesis, in particular, the body frame is chosen coincident with the IMU-frame F_{imu}, so we have $\underline{C}_b^{imu} = \underline{I}$.

The rotation matrix between the IMU-frame and the Camera-frame ,C_{imu}^{cam}, comes from the calibration tests.

The following figure shows the reference systems used in this thesis, and the vectors between the origin of a reference system and the next one.

4.2 Coordinate Trasformations

The coordinate transformations, used to express the navigation vector variables in one reference system or another , chosen in this thesis is the DCM (director cosines matrix) based on Euler angles. Here we use the standard aerospace (and SNAME) standard for Roll Pitch and Yaw angles.

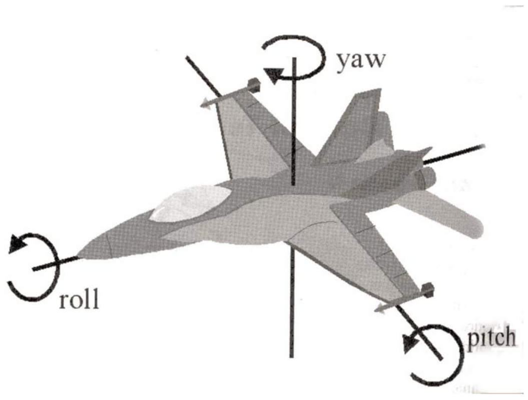


Figure 6: Standard SNAME.

In particular, if we define the frame F_{cam0} as the stereo camera frame taken in the initializazion (see section 6.2), and the frame F_{cam} attached to tehe camera, the visual navigation system estimates the rotation matrix between these frames, $\underline{C}_{cam}^{cam0}$. Starting from this matrix we must compute the pose of the quadrotor in the navigation frame F_n, $\underline{C}_{vn}^{\ b}$, where v stands for navigation.

So we have:

$$\underline{C}_n^{cam} = \underline{C}_{cam0}^{cam} \cdot \underline{C}_n^{cam0} \tag{13}$$

$$\underline{C}_{vn}^{\ b} = \underline{C}_{imu}^{b} \cdot \underline{C}_{cam}^{imu} \cdot \underline{C}_n^{cam} = \underline{C}_{cam}^{b} \cdot \underline{C}_n^{cam} \tag{14}$$

where $\underline{C}_n^{cam0}$, is the rotation matrix between the frame $F_{cam0}$and the navigation frame F_n. This rotation matrix is defined at the inizialization, when the first keyframe is taken, and the TRIAD algorithm is computed (see section 5.1) as:

$$\underline{C}_n^{cam0} = \underline{C}_{imu}^{cam} \cdot \underline{C}_b^{imu} \cdot \underline{C}_{0n}^{\ b} \tag{15}$$

the matrix $\underline{C}_{0n}^{\ b}$ comes from the TRIAD algorithm, it is the inizial attitude of the vehicle respect to the navigazion frame.

5 Attitude Estimation from IMU sensor

The quadrotor used for flight testing in an industrial plant has a stereo camera and an IMU sensor, consisting of a 3-axis linear accelerometer and 3-axis gyroscope.

Figure 7: An example of IMU unit.

The algorithm implemented consists of two parts:

1. inizialiation, and we compute the TRIAD algorithm.
2. main loop, executed each time a new measurement is available from the IMU (accelerometers and gyroscopes).

The main loop is executed each time a new measurement is available from the IMU (accelerometers and gyroscopes) and it provides a 3D orientation by integrating a gyroscope inertial, so the rotation matrix $\underline{C}_n^b$. It is obtained implementing a version of the AHRS filter proposed by Mahony in [7], although it is not available the measurements of the magnetometer to correct heading error. In general a AHRS (Attitude Heading Reference System) is a system for the measurement of attitude and heading, which calculates the 3D orientation with the aid of gyroscopes and reference sensors (such as magnetometers and accelerometers). Using accelerometers and magnetometers, the drift of integration is compensated by reference vectors, namely gravity and earth magnetic field. This results is a drift-free orientation. However in this thesis the UAV does not have the magnetometer, and it will not be possible to calculate corrections for heading.

5.1 Three-axis attitude determination (TRIAD) Algorithm

When the first meseaurement from accelerometers and gyroscopes is available, the first problem is the attitude determination. Determining the attitude of a vehicle is equivalent to determining the rotation matrix describing the orientation of the vehicle-fixed reference frame (called body frame), F_b , with respect to a known reference frame, say navigation frame, F_n . That is, attitude determination is equivalent to determining C_b^n . Although there are nine numbers in this direction cosine matrix, it only takes three numbers to determine the matrix completely . So it takes at least two dfferent measurements to determine the attitude.

We begin with two measurement vectors, such as in this case the UAV's acceleration and the direction of the Earth's magnetic field. We denote the actual vectors by $\hat{a}$ and $\hat{m}$, respectively . The measured components of the vectors, with respect to the body frame, are denoted a_b and m_b, respectively . The known components of the vectors in the navigation frame are a_n and m_n . Ideally , the rotation matrix, or attitude matrix, C_b^n , satisfies:

$$a_b = C_n^b a_n \qquad and \qquad m_b = C_n^b s_b \tag{16}$$

Unfortunately , since the problem is overdetermined, it is not generally possible to find suc an C_b^n. The simplest deterministic attitude determination algorithm is based on discarding one piece of this information; however, this approach does not simply amount to throwing away one of the components of one of the measured directions.

The algorithm is known as the Triad algorithm (Three-axis attitude determination).

It is based on constructing two triads of orthonormal unit vectors using the vector information that we have. The two triads are the components of the same reference frame , denoted F_t, expressed in the body and inertial frames.

This reference frame is constructed by assuming that one of the body/inertial vector pairs is correct. For example, we could assume that the acceleration vector measurement is exact, so that when we find the attitude matrix, the first of the equation (13) is satisfied exactly. We use this direction as the first base vector of F_t. That is,

$$\hat{t_1} = \hat{a} \tag{17}$$

$$t_{1b} = a_b \tag{18}$$

$$t_{1n} = a_n \tag{19}$$

We then construct the second base vector of F_t as a unit vector in the direction perpendicular to the two observations. That is,

$$\hat{t_2} = \hat{a} \times \hat{m} \tag{20}$$

$$t_{2b} = \frac{a_b \times m_b}{|a_b \times m_b|} \tag{21}$$

$$t_{2n} = \frac{a_n \times m_n}{|a_n \times m_n|} \tag{22}$$

The third base vector of F_t is chosen to complete the triad:

$$\hat{t_3} = \hat{t_1} \times \hat{t_2} \tag{23}$$

$$t_{3b} = t_{1b} \times t_{2b} \tag{24}$$

$$t_{3n} = t_{1n} \times t_{2n} \tag{25}$$

Now, we construct two rotation matrices by putting the **t** vector components in to the columns of two 3 x 3 matrices. The two matrices are:

$$\left[\begin{array}{ccc} t_{1b} & t_{2b} & t_{3b} \end{array}\right] \qquad and \qquad \left[\begin{array}{ccc} t_{1n} & t_{2n} & t_{3n} \end{array}\right] \tag{26}$$

It is evident that the two matrices are C_t^b and C_n^t, respectively . Now, to obtain the desired attitude matrix, C_n^b , we simply form:

$$C_n^b = C_t^b C_n^t = \left[\begin{array}{ccc} t_{1b} & t_{2b} & t_{3b} \end{array}\right] \left[\begin{array}{ccc} t_{1n} & t_{2n} & t_{3n} \end{array}\right]^T \tag{27}$$

So, in this thesis the TRIAD algorithm is used at the inizialization to defined the inizial attitude of the vehicle, the rotation matrix $\underline{C}_{0n}^{\,b}$ and the navigation frame F_n with the gravity aligned with the Z-axis, and the X-axis pointing the North.

6 Pose Estimation With Stereo Vision

In literature there is no a single optimal algorithm to calculate the camera pose estimation of the quadrotor. In this section will therefore be presented the algorithm chosen and implemented in this thesis for this purpose.

6.1 Introduction

The use of stereo vision is not the only possible choice. In general, it is possible to estimate motion with monocular vision. The advantage of this kind of setup is primarily the potential to save weight and power by having only one camera. Using monocular vision, it is possible to compute translation and rotation of the camera pose, though the translation can only be found up to a scaling factor. In general, a unique solution can be found, but in order to get a robust recovery algorithm, the baseline between consecutive images has to be sufficiently large and the images have to be taken with two distinctive vantage points. Both conditions are usually not achievable for cameras mounted on a UAV, because UAVs have very fast dynamics and require control algorithm with high update rates. Therefore, compared to the update rate, the UAV moves rather slowly and the baseline remains relatively short. It would be possible to simply skip frames until the baseline is large enough, but this would again violate the constraint of a high update rate.

Because of the drawbacks listed above a stereo vision setup was chosen for this thesis. Using two cameras allows to recover accurate estimates of depth and motion. Futhermore the recovery is done in only one timestep, instead of having to use two consecutive images. However there are some drawbacks of using stereo vision as well, in addition to the increased weight and power consumption, the computational burden is also increased significantly. Additionally, the cameras have to be synchronized, which is not a trivial task given the required accuracy.

6.2 Visual Odometry Algorithm

In this section we present the algorithm used to compute a pose estimation of the UAV from a pair of stereo images. With the image input running, the main algorithm can be started too, and it will first go through a keyframe initialization, and then enter loop where the pose estimation is actually done.

It consists of following part:

- inizialiation, computed only when arrives the first couple of stereo images.
- the main loop, always performed except with the first pair of images.

The initializazion consist of the following steps:

1. Prepare all variables for storage.
2. Read in an image pair, and rectified the images.
3. Detect features in the image pair and match the features between the left and the right pair.
4. Do a refinement of the features' match, and their classification in inilier or outlier.
5. Triangulate the valid matches and get their 3D real world coordinates. Save the matched features, their properties, and the resulting 3D pointcloud. This will be the keyframe reference. Until a new keyframe is taken, all movement is calculated with reference to this frame.

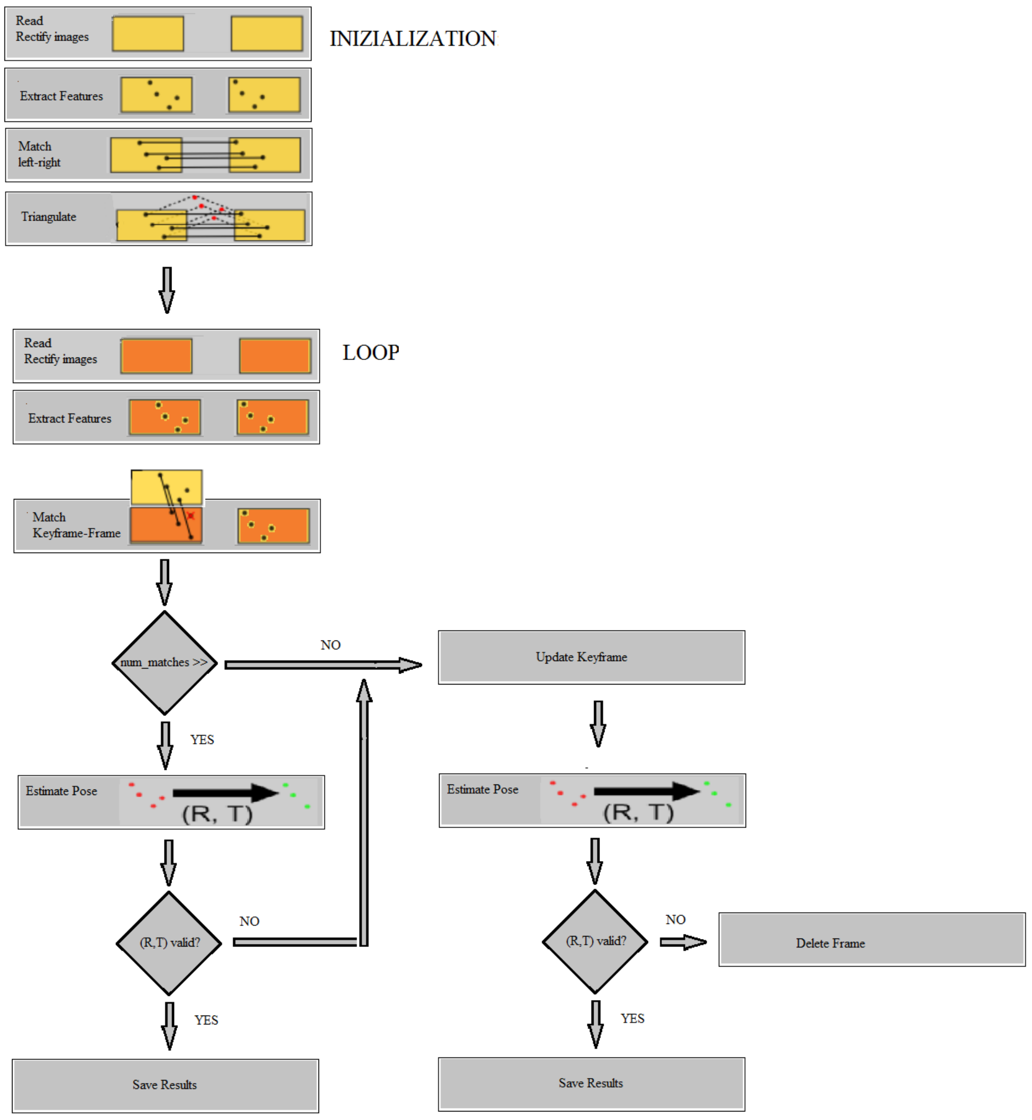


Figure 8: Schematic of the pose estimation algorithm. Top: Initialization to get a keyframe and store the properties. Bottom: Loop to read images, find keypoints, find correspondences in the keyframe, then estimate the pose .

After acquiring the keyframe and the reference data, the main loop runs until a new set of image pair is available. It contains the followin steps:

1. Read in a new pair of images, and rectified the images.
2. Extract features and match features between the current left image and the saved keypoints of the keyframe left image.
3. Do a refinement of the features' match, and their classification in inlier or outlier.
4. If there is enough inlier, run the estimation routine to get the rotation matrix $\underline{R}$ and the translation vector $\mathbf{T}$ which describe the current position w.r.t. the keyframe, and then check if the current pose estimation is valid. If there isn't enough inlier or the current pose estimation w.r.t. the keyframe is not valid, the keyframe will be changed, and a new pose estimation w.r.t. the new keyframe chosen is computed.
5. Update the current position with the new pose estimation, and insert the current frame with its properties in a list, that contains all the previous frame. This list is important because the new keyframe will be chosen among the element of this list.
6. Finally start over with next frame.

In the following sections will present all details for each step in the in the algorithm.

6.2.1 Rectification

When a new set of image pair is available, the images taken by the cameras are each within their own camera coordinate frame and both are distorted according to each cameras' distortion coefficients. For this thesis, the raw images are to be rectified in order to speed up the pose estimation algorithm. This is not mandatory. All the following steps can be performed on raw images. However, rectification not only leads to increased speed, but also allows for a a good and very fast outlier removal step for the left to right feature matching.

The open source library OpenCV [4] already offers a routine, *stereoRectify()* , that computes the operations described in section 2.4. In particular, given:

- the camera matrix $\underline{K}_1$and $\underline{K}_2$.
- The camera distortion parameters of the two cameras.
- The rotation matrix between the coordinate systems of the first and the second cameras, $\underline{R}$.
- The translation vector between coordinate systems of the cameras, $\mathbf{T}$.

the function computes the rotation matrices for each camera that (virtually) make both camera image planes the same plane. Consequently, this makes all the epipolar lines parallel and thus simplifies the dense stereo correspondence problem.

The output, as previously stated, are two images that have been transformed into the same coordinate frame (only the baseline distance in x-direction is still there), and have parallel epipolar lines.

6.2.2 Feature Detection : SIFT

The first thing to do after reading in a pair of images is to find points of interest within them. This is a very important step as it will lay the groundwork for all the computer vision steps later in the algorithm. While a human eye can easily recognize objects in an image, even if they are partially occluded, distorted, rotated or skewed, a computer is still incapable of achieving much more than very simple object recognition. There exist algorithms were a computer can learn the shape and characteristics of an object and then find it within images, but science is nowhere near the goal of being able to find whatever is considered 'relevant data' in arbitrary images.

Feature detectors might have some or all of the following desirable properties though it is usually better to select those that are needed and not generally include 'as many as possible'. It is also quite obvious the first property of low computational cost might be in conflict with the other properties:

- Fast computation (low computational cost).
- Distinctiveness: Features should be unique and distinguishable.
- Repeatability: If the algorithm is applied to the same image twice, the same features should be detected.
- Invariance to scale: if the image is viewed at different scale, the same features should be detected.
- Invariance to rotation: if the image is viewed in a rotated version, the same features should be detected.
- Invariance to illumination: if the image is viewed under different illuminations, the same features should be detected.

In literarure exist much feature detectors, like SURF, FAST, Harris, with some of the previous properties. In this thesis we have chosen to use the Scale Invariant Features Transform (SIFT) as feature detector.

The main difference between the others feature detectors is that SIFT is scale invariant. Some corner detectors, like Harris and so on, are rotation-invariant, which means, even if the image is rotated, we can find the same corners. It is obvious because corners remain corners in rotated image also. But what about scaling? A corner may not be a corner if the image is scaled. For example, check a simple image below (Figure 9). A corner in a small image within a small window is flat when it is zoomed in the same window. So Harris corner is not scale invariant.

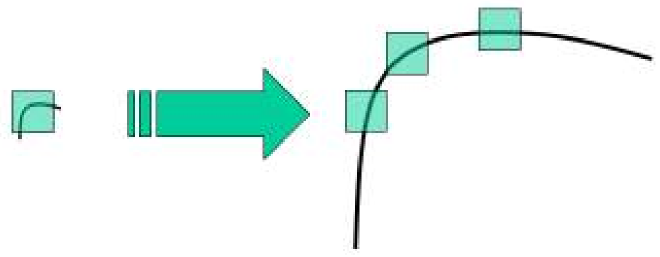


Figure 9: Feature for corner at different scale

So, in 2004, D.Lowe, University of British Columbia, came up with a new algorithm, Scale Invariant Feature Transform (SIFT) in his paper [6], which extract keypoints and compute its descriptors. There are mainly four steps involved in SIFT algorithm, and we present just a short summary of this paper.

Scale-space Extrema Detection

From the Figure 9, it is obvious that we can't use the same window to detect keypoints with different scale. It is OK with small corner. But to detect larger corners we need larger windows. For this, scale-space filtering is used. In it, Laplacian of Gaussian is found for the image with various σ values. LoG acts as a blob detector which detects blobs in various sizes due to change in σ. In short,σ acts as a scaling parameter. For example, in the above image, gaussian kernel with low σ gives high value for small corner while guassian kernel with high σ fits well for larger corner. So, we can find the local maxima across the scale and space which gives us a list of (x,y,σ) values which means there is a potential keypoint at (x,y) at σ scale.

But this LoG is a little costly, so SIFT algorithm uses Difference of Gaussians which is an approximation of LoG. Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different σ, let it beσ and $k \cdot \sigma$. This process is done for different octaves of the image in Gaussian Pyramid. It is represented in below image:

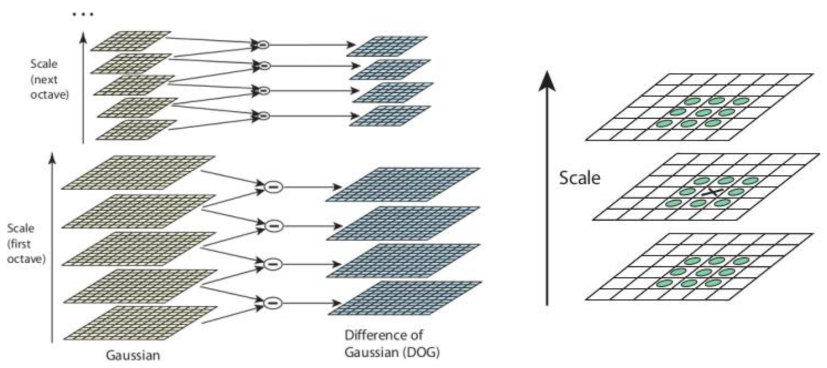


Figure 10: The Gaussian Pyramid

Once this DoG are found, images are searched for local extrema over scale and space. For eg, one pixel in an image is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale. It is shown in below image:

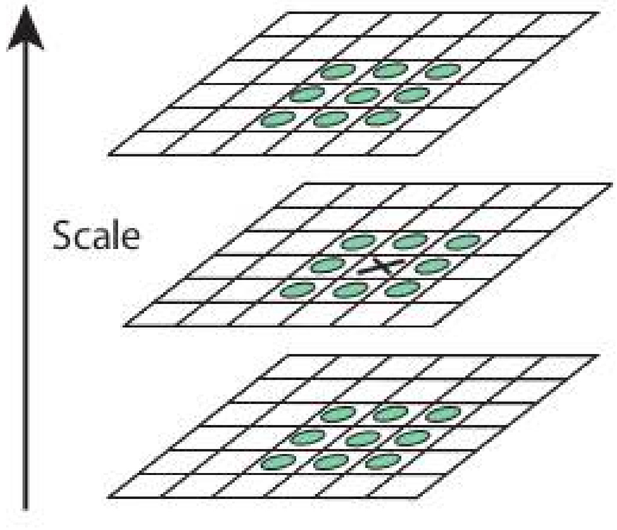


Figure 11: Images are searched for local extrema over scale and space

Regarding different parameters, the paper gives some empirical data which can be summarized as, number of octaves = 4, number of scale levels = 5, initial σ=1.6, $k = \sqrt{2}$ etc as optimal values.

Keypoint Localization

Once potential keypoints locations are found, they have to be refined to get more accurate results. They used Taylor series expansion of scale space to get more accurate location of extrema, and if the intensity at this extrema is less than a threshold value (0.03 as per the paper), it is rejected.

DoG has higher response for edges, so edges also need to be removed. For this, a concept similar to Harris corner detector is used. They used a

2x2 Hessian matrix ($\underline{H}$) to compute the pricipal curvature.
If this ratio is greater than a threshold that keypoint is discarded. It is given as 10 in paper.

So it eliminates any low-contrast keypoints and edge keypoints and what remains is strong interest point.

Orientation Assignment

Now an orientation is assigned to each keypoint to achieve invariance to image rotation. A neigbourhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. (It is weighted by gradient magnitude and gaussian-weighted circular window with σ equal to 1.5 times the scale of keypoint. The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. It creates keypoints with same location and scale, but different directions. It contribute to stability of matching.

Keypoint Descriptors and Matching

Now keypoint descriptor is created. A 16x16 neighbourhood around the keypoint is taken. It is devided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc.

Keypoints between two images are matched by identifying their nearest neighbours. But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other reasons. In that case, ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected. It eliminaters around 90% of false matches while discards only 5% correct matches, as per the paper.

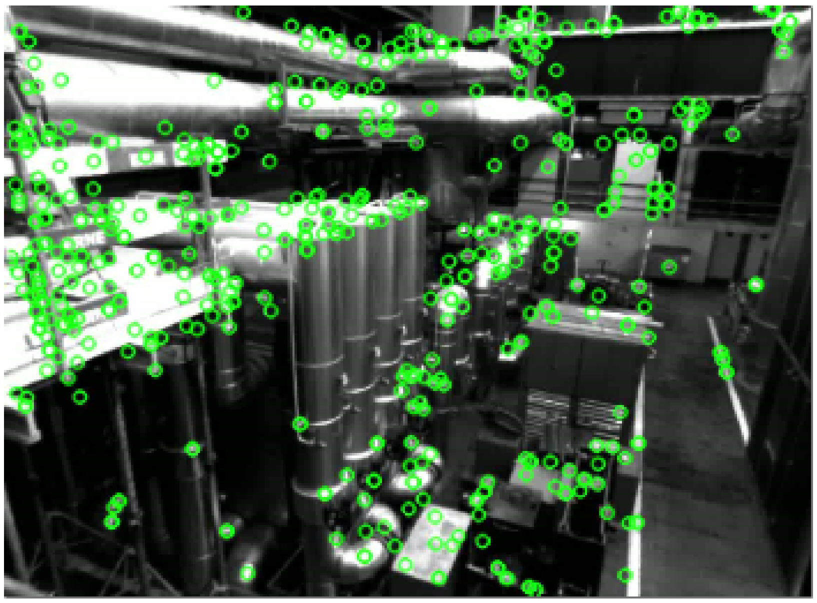
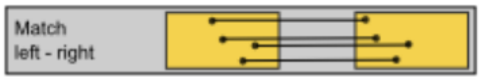


Figure 12: Example of feature extraction

So this is a summary of SIFT algorithm. For more details and understanding, reading the original paper is highly. The open source library OpenCV offers some routines that compute the feature detection with SIFT and then extract descriptors from the keypoints.

6.2.3 Finding Correspondences and Doing Matches Refinement



After extracting features in the images, correspondences between two sets of such features have to be detected. This has to be done between the left image of the current frame and the keyframe. There are two main categories of algorithms to find correspondences: those relying on descriptors (matching), and those that do no use any descriptors (tracking).

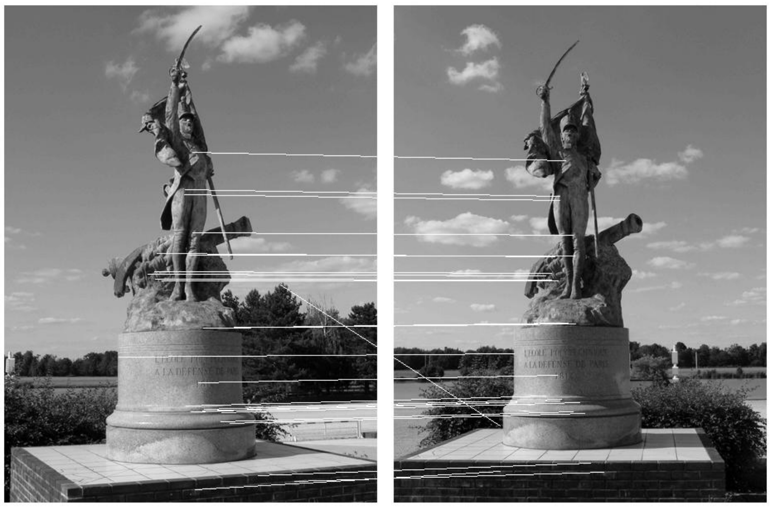


Figure 13: Correspondes between features in the left image and in the right one.

Regardless of which approach is used, finding correspondences in this thesis may consists of:

- when a new keyframe is taken, find correspondences between the left and the right image of the new keyframe. The correspondences found in this step are used then to get their 3D real world coordinates.
- when a new a set of pair image is available, find correspondences between the left image of the current frame and the left image of the keyframe. The correspondences found in this step are later used in the pose estimation step, where the estimator needs to know which 2D

point of the current frame belongs to which 3D point of the keyframe. Only searching correspondences in the left frames is an arbitrary choice. It would also be useful to do it for both the left and right images, but the gain in performance would not justify the additional computational effort.

In this thesis we choice to use descriptor based matching . It uses, as its name suggests, descriptors in order to find the corresponding features in two sets.

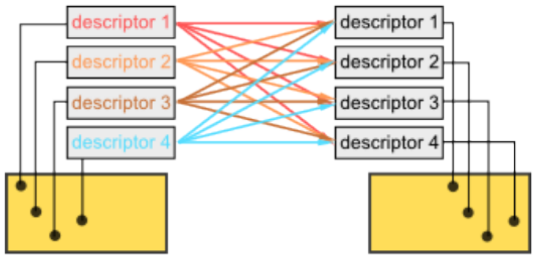


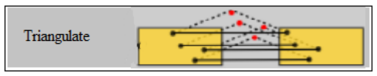
Figure 14: Schematic of descriptor based matching.

For every feature a descriptor is computed. This happens in both images independently, which requires a feature detector with good repeatability. The advantage of using descriptors is that choosing a smart descriptor can have a positive effect on speed and accuracy of the matching process, might allow to include invariances and can enable the use of smart matching techniques. The drawbacks of using descriptors is, that it takes some computational effort to compute them, in some cases this is way too much for practical use on a UAV. Figure 14 shows the straightforward way of matching four descriptors: just test them all in a brute-force approach and select the best match for each. The open source library OpenCV offers some routines that compute the feature detection with SIFT.

After the features matching, one of the main problem is the presence of a certain percentage of outliers. In order to identify the valid match, given the pixel coordinates of the corrispondences between the two frames, we estimate the fundamental matrix (see section 2.3) and those points that violate the equation (13) , the epipolar costraint, are classified as outliers.

The fundamental matrix is estimate through an OpenCV routine, *findFundamentalMat()*. And then we stored only the matches that are classified as inliers.

6.2.4 Triangulation



Once two corresponding points in the two images have been located, the 3D real world position of them can be reconstructed through triangulation. This operation is computed when a keyframe is taken, with a set of features in the left image and a set of features in the right image including a correspondence map have been computed.

Given the pixel coordinates, $\mathbf{x}_l$ and $\mathbf{x}_r$, of the projection , of a 3D real world point, $\mathbf{X}$,in the left image and in the right one respectively, the folowing equation holds:

$$\mathbf{x}_l = \underline{K}_l \mathbf{X} \tag{28}$$

$$\mathbf{x}_r = \underline{K}_r \mathbf{X} \tag{29}$$

$$\mathbf{x}_l - \underline{R}\mathbf{x}_r = \mathbf{T} \tag{30}$$

where $\underline{R}$ and $\mathbf{T}$ are the rotation matrix and the vector between camera left and camera right.

From equations (25) and (26), we obtain:

$$\mathbf{X} = \underline{K}_l^{-1}\mathbf{x}_l = \begin{bmatrix} \frac{x_l}{z_l} \\ \frac{y_l}{z_l} \\ 1 \end{bmatrix} = \mathbf{x}'_l \cdot z_l \tag{31}$$

$$\mathbf{X} = \underline{K}_r^{-1}\mathbf{x}_r = \begin{bmatrix} \frac{x_r}{z_r} \\ \frac{y_r}{z_r} \\ 1 \end{bmatrix} = \mathbf{x}'_r \cdot z_r \tag{32}$$

where $\mathbf{x}'_l$ and $\mathbf{x}'_r$ are the pixel coordianates of the decalibrated points.

From equation (27) we obtain:

$$[\mathbf{x}'_l - \underline{R}\mathbf{x}'_r] \begin{bmatrix} z_l \\ z_r \end{bmatrix} = \underline{H} \begin{bmatrix} z_l \\ z_r \end{bmatrix} = \mathbf{T} \tag{33}$$

So , finally:

$$\begin{bmatrix} z_l \\ z_r \end{bmatrix} = \underline{H}^{+}\mathbf{T} \tag{34}$$

where $\underline{H}^{+}$is the pseudo-inversa of the matrix $\underline{H}$.

Once obtained z_l and z_r the triangulated point is obtained by equation (28) or (29):

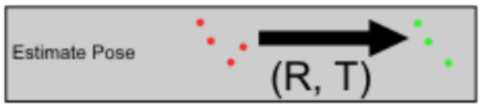
$$\mathbf{X} = \mathbf{x}'_l \cdot z_l \tag{35}$$

After triangulating, some plausibility checks are usually useful to get rid of outliers. It is often possible to define rules to limit the 3D real world space in which the features should be located. For example it is already helpful to exclude every triangulated point with a negative z-coordinate, since in reality all points need to lie in front of the camera. The acceptable space for points can be further specified if the setup allows it, for example using knowledge about the cameras' fields of view and the baseline between them allows to define an acceptable volume. Additional information about the task can also help to further refine and minimize the volume. All points triangulated to locations outside this volume are then considered outliers and discarded.

In this this thesis we have implemented the previous equation in C++ code. The inputs of the triangulation function are the two vectors containing the features and the correspondence map specifying which features have to be paired up, and the camera calibration data for both cameras. The function simply outputs the 3D coordinates, w.r.t the left camera frame, of all the pairs of features. The triangulate points are finally saved in a structure with the other properties of the keyframe.

Compared to the computationally very expensive feature extraction and matching steps, triangulation is computationally cheap and is basically a non-factor for the overall looptime.

6.2.5 Pose Estimation: DLT



The last step of the pose estimation algorithm is actually estimate the current pose with respect to the keyframe pose. A large number of different markerless pose estimation algorithms already exist in literature. Usually the camera pose for a given image is estimated solely using a set of correspondence, a match of an object's natural feature (3D) detected on the image (2D). In particular given an homogeneous representation $\mathbf{X}_i \in \mathbb{R}^4$ of a 3D point in world coordinates, is mapped by the camera to the point $\mathbf{x}_i \in \mathbb{R}^3$. $\mathbf{x}_i$ itself is a homogeneous representation of the corresponding 2D point on the image. This mapping is defined by:

$$\mathbf{x}_i = \underline{K} \cdot [\ \underline{R} \quad \mathbf{T}\ ] \cdot \mathbf{X}_i \tag{36}$$

So estimation of the camera pose from correspondences then refers to searching the camera pose $[\ \underline{R} \quad \mathbf{T}\ ]$ which best relates a set of given correspondences $C = \{\mathbf{X}_i \leftrightarrow \mathbf{x}_i\}$ using the previous equation.

In this thesis we'll use the Direct Linear Transformation (DLT) algorithm. These algorithms intend to directly estimate the camera pose ignoring certain restrictions regarding the solution space. However the main problem of this algorithm is the correspondences will also contain a certain percentage of outliers . These outliers are problematic because they potentially can have a huge negative impact on the quality of the solution. Therefore, methods such as RANSAC, used in this thesis, need to be used to identify the inliers in order to perform the camera pose estimation only using this set.

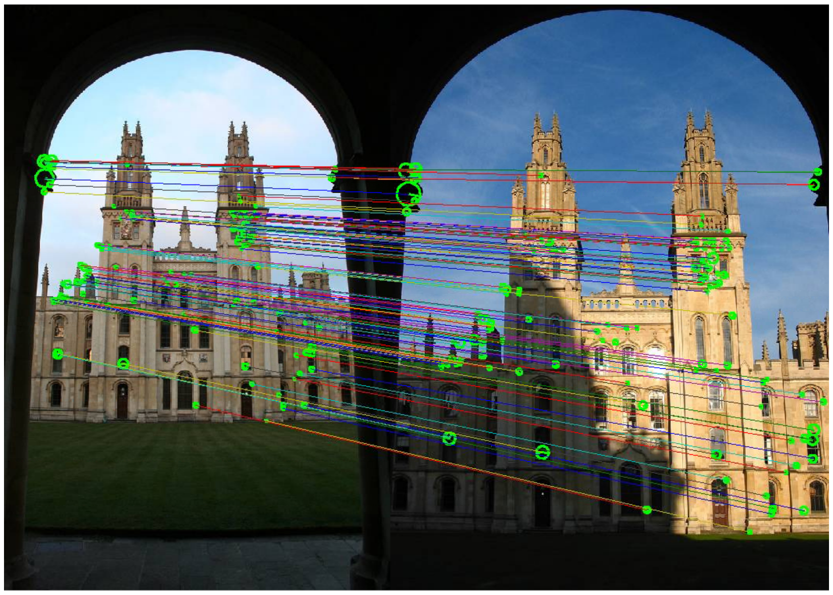


Figure 15: Correspondes between features of the keyframe and of the frame.

Given the following inputs :

- the 3D triangulate points of the keyframe.
- the 2D points in the left image of the current frame.
- the correspondences between the left image of the current frame and the left image of the keyframe.

the algorithm, that provides an estimation of the current pose w.t.r. the keyframe pose, consists of:

- exclusion of outliers.
- performing DLT with only inliers.

RANSAC uses a stochastic approach in order to identify the inliers. Therefore this algorithm is of non-deterministic nature in the sense that it produces a reasonable result only with a certain probability. This probability increases as more iterations are allowed. The algorithm operates as follows:

- A random subset C_S of the provided correspondences C is selected. It contains only the minimal number of correspondences required for camera pose estimation, chosen equal to six in this thesis.
- A camera pose Q is estimated using the DLT algorithm along with C_S.
- All remaining correspondences$C \backslash C_S$ are then checked for integrity with Q . Therefore the 3D points of the keyframe are projected using the known camera matrix $\underline{K}$ and $\underline{Q}$ to the 2D points $\mathbf{x}_i^{'}$. If $\|\mathbf{x}_i^{'} - \mathbf{x}_i\| \leqslant \tau$, then the correspondence $\mathbf{X}_i \leftrightarrow \mathbf{x}_i$ fits well to the estimated camera pose and will therefore also be considered as a hypothetical inlier.

This procedure is repeated a fixed number of times, each time producing either a camera pose which is rejected because too few points are classified as inliers or a refined pose together with a corresponding error measure. In the second case the refined pose is kept if its error is lower than the last saved pose.

$\underline{Q}$ is reasonably good if a sufficient number of correspondences has bg (poor quality of the frame, not enough inlier, and son on). It's better delete wrong estimations, because only one wrong estimation could affect the next ones. It may happen, infact, that the frame een classified as hypothetical inliers.

Finally Q is then reestimated from all hypothetical inliers using DLT .

6.2.6 Direct Linear Transformation (DLT) Algorithm

For the Direct Linear Transformation (DLT) algorithm [3] it is assumed that the set of points $\mathbf{X}_i$ spans a real 3D space which means that those points are arbitrarily distributed in space an thus do not lie on a single point, line or plane. Let $\mathbf{X}_i = (\ X_i \ \ Y_i \ \ Z_i \ \ W_i\ )^T \leftrightarrow \mathbf{x}_i = (\ u_i \ \ v_i \ \ w_i\ )^T$ be n correspondences. A DLT $\underline{F} \in \mathbb{R}^{3x4}$ now is a linear function $\underline{F}$: $\mathbb{R}^4 \longmapsto \mathbb{R}^3$ which maps the points $\mathbf{X}_i$ to the points $\mathbf{x}_i$. This can be expressed in the homogeneous context as

$$\underline{F}\mathbf{X}_i \sim \mathbf{x}_i \Leftrightarrow \mathbf{x}_i \times \underline{F}\mathbf{X}_i = 0 \tag{37}$$

F has 12 unknows:

$$\underline{F} = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1^T \\ \mathbf{f}_2^T \\ \mathbf{f}_3^T \end{bmatrix} \tag{38}$$

Rewriting equation using the fact $\mathbf{f}_i^T \mathbf{X}_i = X_i^T \mathbf{f}_i$ that yields

$$\mathbf{x}_i \times \underline{F}\mathbf{X}_i = \begin{pmatrix} v_i \mathbf{f}_3^T \mathbf{X}_i - w_i \mathbf{f}_2^T \mathbf{X}_i \\ w_i \mathbf{f}_3^T \mathbf{X}_i - u_i \mathbf{f}_2^T \mathbf{X}_i \\ u_i \mathbf{f}_3^T \mathbf{X}_i - v_i \mathbf{f}_2^T \mathbf{X}_i \end{pmatrix} = \begin{bmatrix} \mathbf{0}^T & -w_i \mathbf{X}_i^T & v_i \mathbf{X}_i^T \\ w_i \mathbf{X}_i^T & \mathbf{0}^T & -u_i \mathbf{X}_i^T \\ -v_i \mathbf{X}_i^T & u_i \mathbf{X}_i^T & \mathbf{0}^T \end{bmatrix} \begin{pmatrix} \mathbf{f}_1^T \\ \mathbf{f}_2^T \\ \mathbf{f}_3^T \end{pmatrix} \tag{39}$$

However each correspondence will only derive two linearly independent equations. The linearly independent system now reads

$$\begin{bmatrix} \mathbf{0}^T & -w_i \mathbf{X}_i^T & v_i \mathbf{X}_i^T \\ w_i \mathbf{X}_i^T & \mathbf{0}^T & -u_i \mathbf{X}_i^T \end{bmatrix} \mathbf{f} = 0 \tag{40}$$

For $n \geq 6$ the system is overdetermined and hence F can be estimated by a Singular Value Decomposition (SVD). The camera pose can now be extracted from F :

$$\underline{F}\mathbf{X}_i \sim \mathbf{x}_i \sim \underline{K}[\ \underline{R} \quad \mathbf{t}\ ]\mathbf{X}_i \Rightarrow \underline{F} \sim \underline{K}[\ \underline{R} \quad \mathbf{t}\ ] \Rightarrow [\ \underline{R} \quad \mathbf{t}\ ] \sim \underline{K}^{-1}\underline{F} \qquad (41)$$

Since by the previous equation the pose is defined.

6.2.7 Validation of the Camera Pose Estimation

When new camera pose estimation is computed (see section 5.2.5) it could be wrong (poor quality of the frame, not enough inlier, and son on). It's better delete wrong estimations, because only one wrong estimation could affect the next ones. It may happen, infact, that the frame that it is associated to a wrong estimate, inserted in the list of old frame, is chosen as the new keyframe. In this case all the next estimates will be affected by mistake. So it's better not consider the current pose estimation.

In this thesis we chose to consider not valid a pose estimation if:

- the number of inlier used by the DLT is low.
- the projection error computed in the DLT algorithm is greater.

6.2.8 Update Keyframe

In the inizialization of the algorithm, when the first couple of image is arrived, the first keyframe is taken. However it could be necessary to update the keyframe. Note infact that all the camera pose estimation is computetd respect the keyframe, through the correspondences between the left image of the current frame, and the left one of the keyframe.

In this thesis we choose to update the keyframe when :

- if at the step 4 of the algorithm's loop, the the number of refined matches between the left image of the current frame and the keyframe is not enough large. In this case infact the inliers in the DLT algorithm could be very small and the camera pose estimation pose could be wrong.
  There are multiple reasons why this happens, for examole if the frame does not have enough features, or if the UAV has moved a lot w.r.t. where it has been taken the last keyframe, and so there aren't enough features in common. Note that it is not possible to know if it is, therefore, the poor quality of the frame or the relative motion of the UAV due to the few inlier. In this thesis regardless of what they are owed the few refined matches, we decide to update the keyframe.
- the camera pose estimation using DLT is computed with a few number of inliers or the projection error is greater than a certain threshold.

In both cases we chose a new keyframe.

The new keyframe is chosen searching in the list containing the old frames and their properties (keypoints and descriptors) and solving this problem of maximum:

$$NewKeyframe = \underset{i}{argmax}(Matcher(x_{currentFrame}, x_{frame_i})$$

So we make the match between these and the current frame. The the frame with a number of match greater than the others becomes the new keyframe.

Then we re-match the left image of the current frame with the left one of the new keyframe, and a new camera pose estimation using DLT is computed. Note that if the new camera estimation computed is considered again not valid, the frame is deleted and not inserted in the list.

The list of old frames is also update when the keyframe is changed: we chose to delete the oldest frame and to leave in the list no more than the 20 recent frames. This is the better choice respect to delete all the list, as the following figures shown, because choosing the new keyframe as written before, allows to correct possible error in pose estimation.

For example suppose the current keyframe is the frame 9 and we'll chose the frame 12 as new keyframe. If the the rotation matrix $\underline{C}_{key_{12}}^{key_9}$ is affected by an error the current attitude estimation is also wrong :

$$\underline{C}_{cam}^{cam0} = \underline{C}_{key_9}^{cam0} \cdot \underline{C}_{key_{12}}^{key_9} \cdot \underline{C}_{cam}^{key_{12}}$$

However if in the next keyframe updating we choose the frame 11 , the next attitude estimation is: $\underline{C}_{cam}^{cam0} = \underline{C}_{key_9}^{cam0} \cdot \underline{C}_{key_{11}}^{key_9} \cdot \underline{C}_{cam}^{key_{11}}$and the error on the attiude estimation could be recovered.

The figures 16,17,18,19,20,21 are obtained in the simulation enviroment using ROS and Gazebo and they show the difference between three cases:

1. the new keyframe chosen as the last frame available .
2. the new keyframe chosen in a list of old frames and the list is deleted when the keyframe is updated .
3. similar to the case 2 but the list is not deleted .

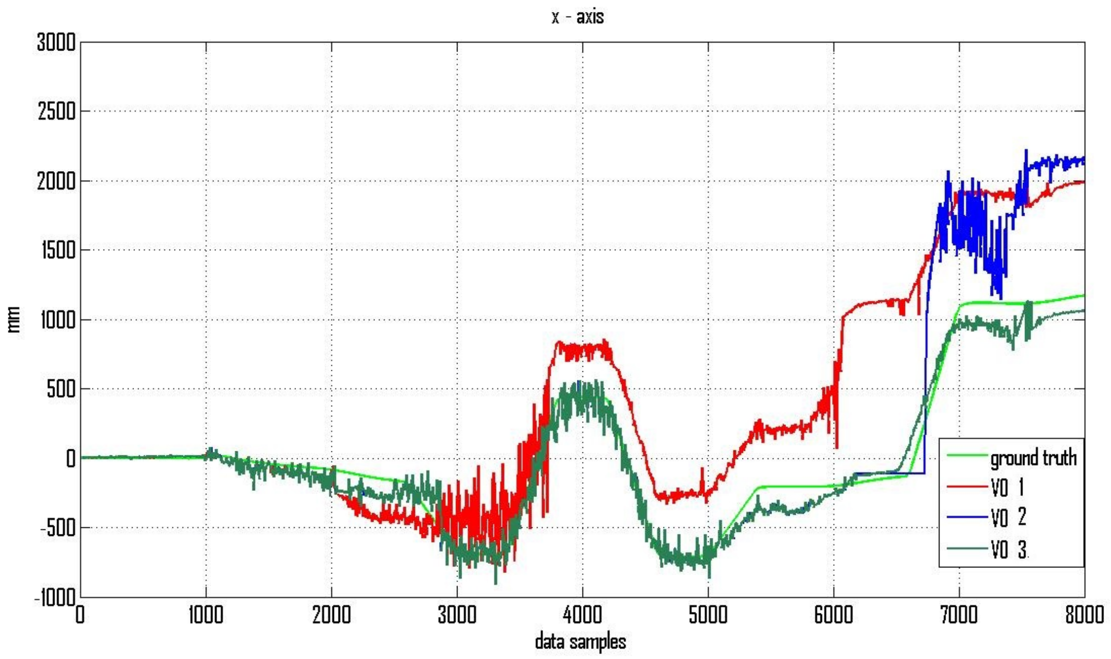


Figure 16: The Figures shows the difference between three cases about the pose estimation on x-axis: the new keyframe chosen as the last frame available (case 1), the new keyframe chosen in a list of old frames and the list is deleted when the keyframe is updated (case 2), similar to the case 2 but the list is not deleted (case 3).

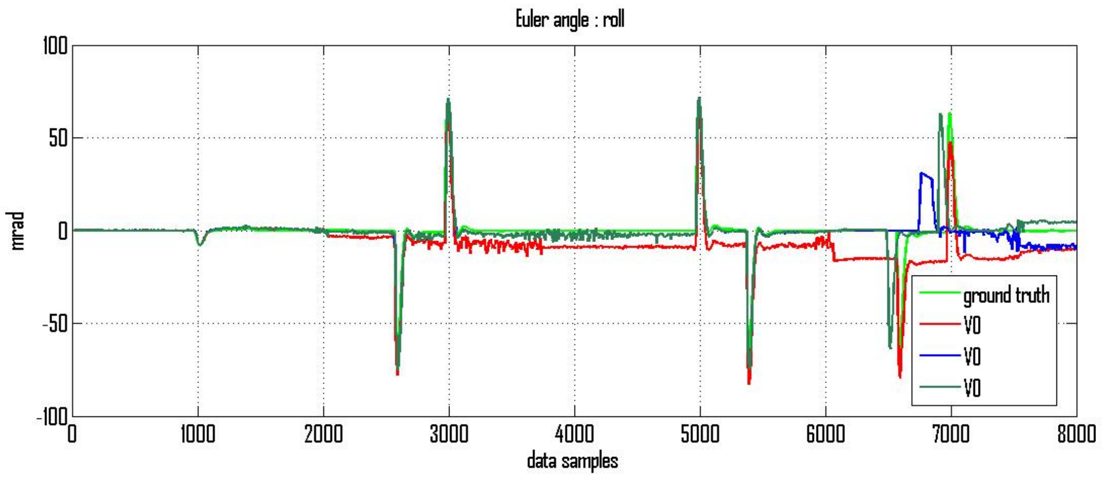


Figure 17: The Figures shows the difference between three cases about the attitude estimation on x-axis: the new keyframe chosen as the last frame available (case 1), the new keyframe chosen in a list of old frames and the list is deleted when the keyframe is updated (case 2), similar to the case 2 but the list is not deleted (case 3).

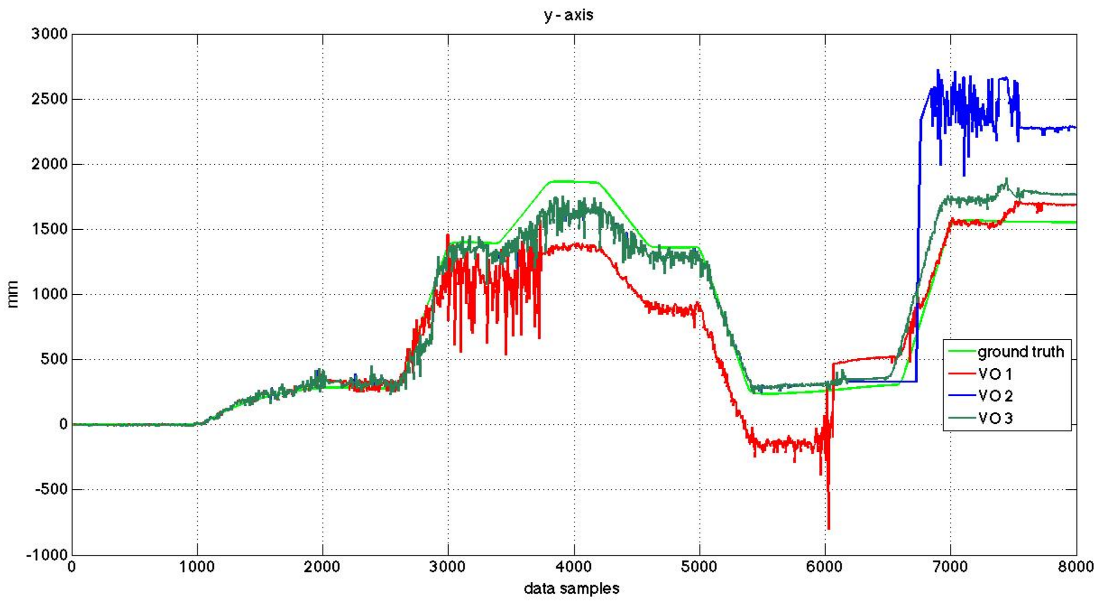


Figure 18: The Figures shows the difference between three cases about the pose estimation on y-axis: the new keyframe chosen as the last frame available (case 1), the new keyframe chosen in a list of old frames and the list is deleted when the keyframe is updated (case 2), similar to the case 2 but the list is not deleted (case 3).

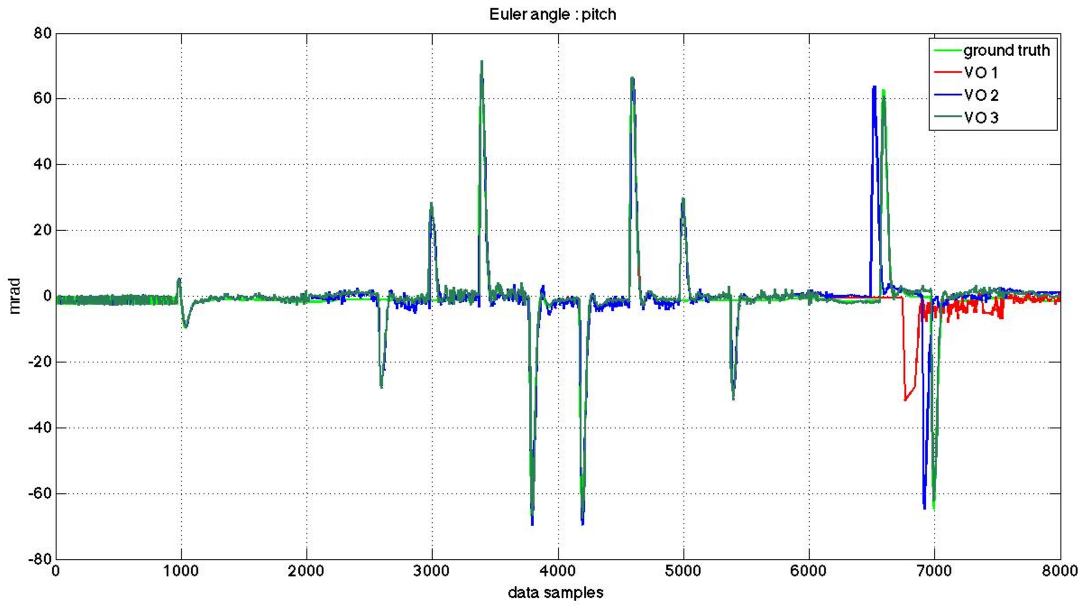


Figure 19: The Figures shows the difference between three cases about the attitude estimation on y-axis: the new keyframe chosen as the last frame available (case 1), the new keyframe chosen in a list of old frames and the list is deleted when the keyframe is updated (case 2), similar to the case 2 but the list is not deleted (case 3).

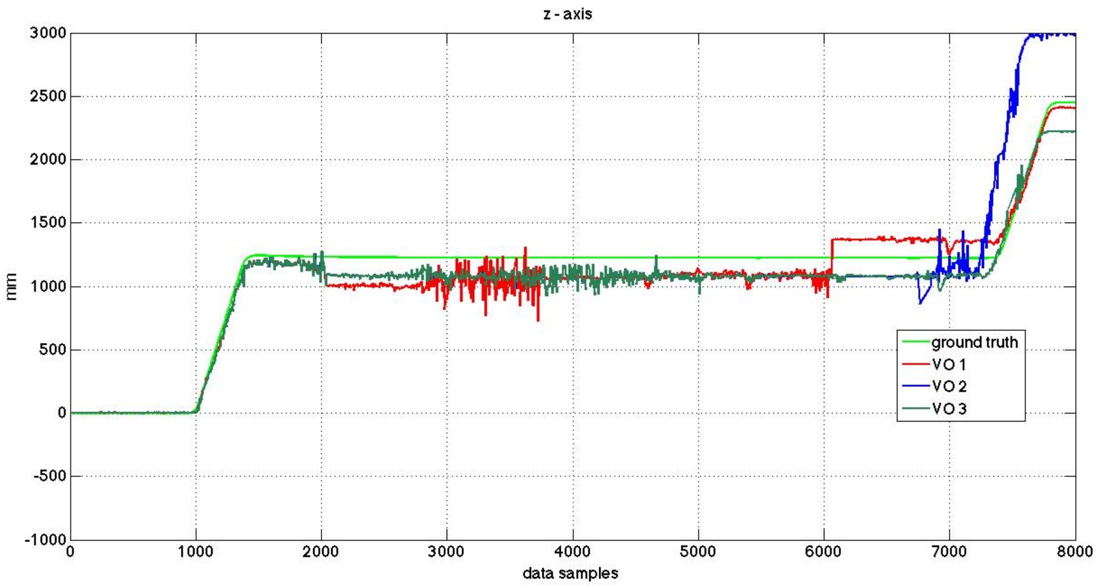


Figure 20: The Figures shows the difference between three cases about the pose estimation on z-axis: the new keyframe chosen as the last frame available (case 1), the new keyframe chosen in a list of old frames and the list is deleted when the keyframe is updated (case 2), similar to the case 2 but the list is not deleted (case 3).

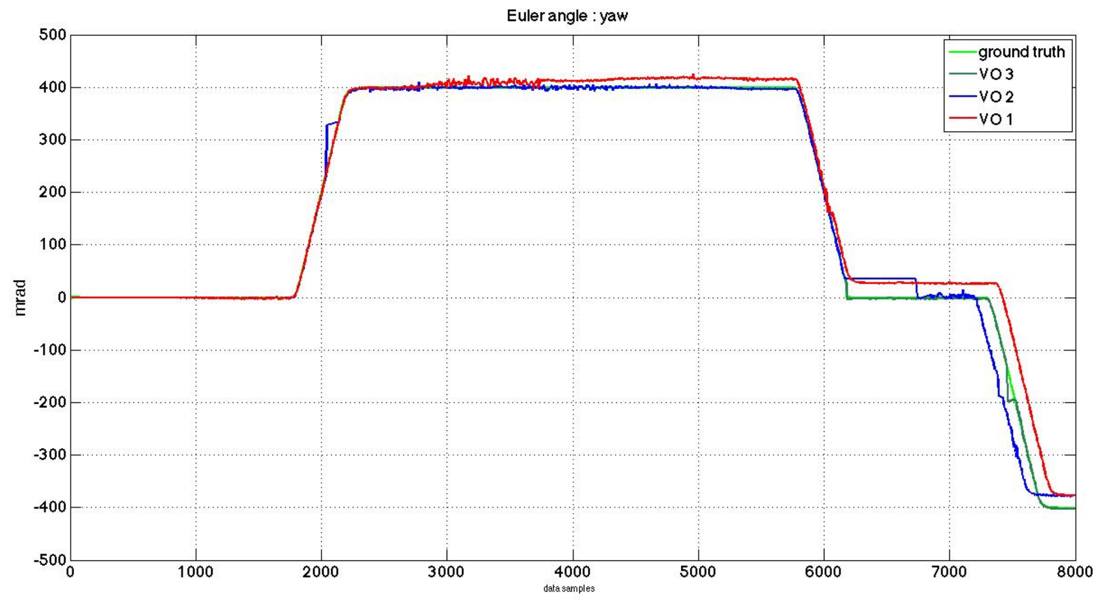


Figure 21: The Figures shows the difference between three cases about the attitude estimation on z-axis: the new keyframe chosen as the last frame available (case 1), the new keyframe chosen in a list of old frames and the list is deleted when the keyframe is updated (case 2), similar to the case 2 but the list is not deleted (case 3).

7 Integration of the Inertial System with Visual Odometry

The UAV is equipped with a stereo camera, and an onboard IMU sensor, in order to do pose estimation. However both the inertial navigation system, obtained from the IMU, and the visual navigation system have important advantages but also serious limitations. The first type of systems are affected by errors that increase over time, in particular on the estimation of the position. In the second ones errors may occur on the estimation of attitude or position.

Therefore it is evident the importance of an integration between the two navigation systems. The goal is to obtain a navigation system, whose primary component is given by the visual, while the intervention of measurements coming from the inertial system, able to follow with great accuracy the short-term dynamics of the vehicle, allows to correct possibily errors of camera pose estimation and ensuring accuracies higher than those that would be achieved by using separate components.

In this thesis we proposed a algorithm where we realize the interation between the systems when it is necessary to update the keyframe .In the following section we explain in details how the integration is realised.

7.1 Update Keyframe with inertial measurement

When it is necessary to update the keyframe, the new keyframe is chosen in a list of old frames, that are associated to a pose estimation w.r.t. the first keyframe, taken in the inizialization $\underline{C}_{vcam0}^{key_i}$.

After the keyframe is updated, all the next pose estimations will be computed respect to this one. In order to correct possible errors on the attitude, and to avoid that these errors affects the next estimations, we compute the camera attitude estimation at the i-th frame, chosen as new keyframe, from the inertial system ,too $\underline{C}_{icam0}^{key_i}$.

Then we compute the difference between the two attitude estimations. If the difference between the two estimates is greater than a certain threshold, we set the attitude estimation of the vehicles at i-th frame equal to the one coming from the inertial system, and then we compute the new camera pose estimation.

However, the inertial system computes the attitude of the vehicles integrating the gyroscopes in the navigation frame. Then from the inertial system we have the rotation matrix $\underline{C}_{ib}^{n}$. The visual system computes the attitude of the vehicle in the camera frame F_{cam0}, so we have the rotation matrix $\underline{C}_{vcam0}^{key_i}$, where F_{key_i} is the camera frame, at the i-th frame, rigidily attached with the stereo camera. So it's necessary the folowing matrix transformations, where v stands for visual system, and i for inertial system:

$$\underline{C}_{ikey_i}^{n} = \underline{C}_{ib}^{n} \cdot \underline{C}_{imu}^{b} \cdot \underline{C}_{key_i}^{imu} \tag{42}$$

$$\underline{C}_{icam0}^{key_i} = \underline{C}_{i\ key_i}^{Tn} \cdot \underline{C}_{\ \ n}^{T\,cam0} \tag{43}$$

where $\underline{C}_{imu}^{b} = \underline{I}$, $\underline{C}_{n}^{cam0} = \underline{C}_{imu}^{cam0} \cdot \underline{C}_{b}^{imu} \cdot \underline{C}_{0\ b}^{Tn}$, $\underline{C}_{key_i}^{imu}$ come from the calibration beacuse is equal to $\underline{C}_{cam}^{imu}$.

Now the difference between the two rotation matrices is computed.

$$\Delta C_{cam0}^{key_i} = \underline{C}_{v\ cam0}^{T key_i} \cdot \underline{C}_{icam0}^{key_i} \tag{44}$$

From the matrix $\Delta C_{cam0}^{key_i}$ we compute the euler parametrization and we check if the euler angles are greater than a certain threshold. In this case we set $\underline{C}_{vcam0}^{key_i} = \underline{C}_{icam0}^{key_i}$.

7.2 Evaluation of Pose Estimation Algorithm

The goal of this thesis is to realized a Visual Navigation System for MAV (Micro Air vehicle). In order to verify the performance of pose estimation EuRoC provided a dataset with recorded real data. The quadrotor used for flight testingin in an industrial plant have a stereo camera and an a onboard IMU unit. The cameras have a resolution of 752x480 pixel and a fequency of 20Hz. The IMU unit has a frequency of 200Hz.

The Figures 22,23,24 shown the difference between the attitude estimation from Inertial Navigation System , and from Visual Odometry without attitude correction. The Figure 25 shows the difference between the groung truth, provided by EuRoC, and the estimation of the translations comes from Visual Odometry without attitude correction. The accuracy of the pose estimation in this case is very low.

The Figure 26,27,28 shown the difference between the attitude estimation from Inertial Navigation System , and from Visual Odometry with attitude correction.The Figure 29 shows the difference between the groung truth, provided by EuRoC, and the estimation of the translations comes from Visual Odometry without attitude correction. The accuracy of the pose estimation in this case is better than the previous case.

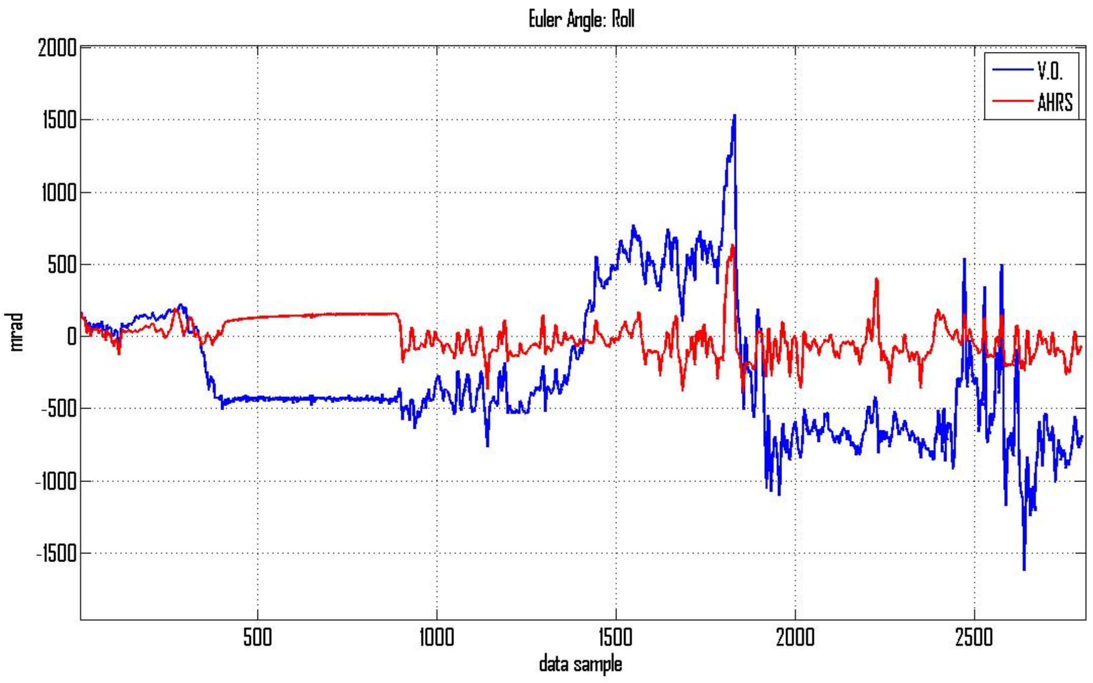


Figure 22: Estiamtion of roll angle from Inertial System (red) , and from Visual Odometry (blue)

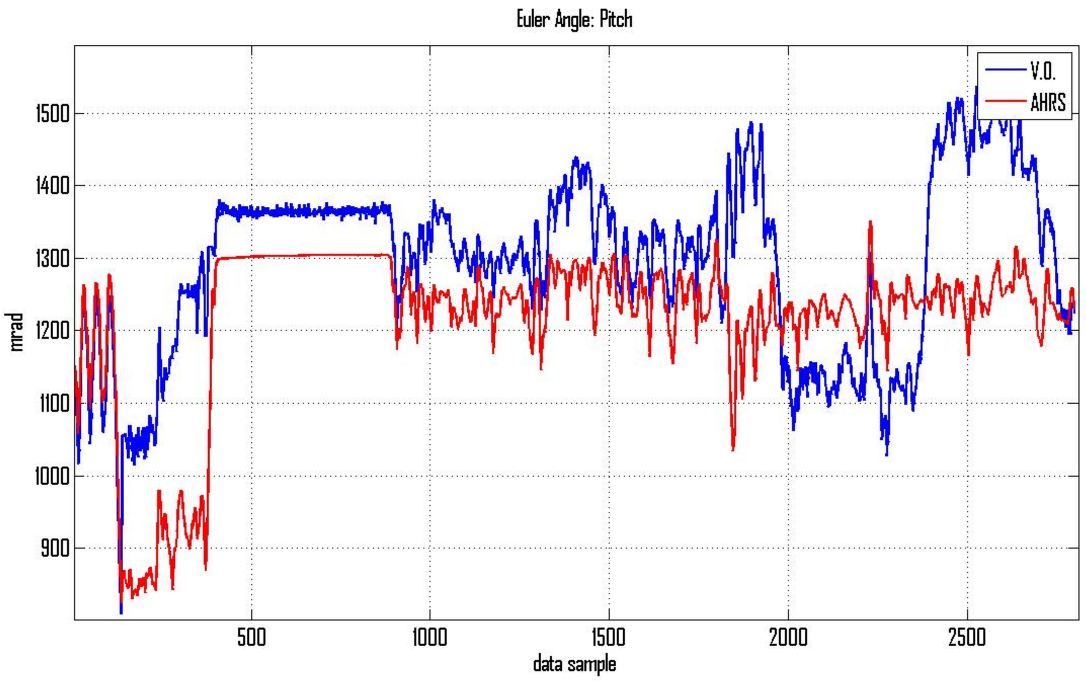


Figure 23: Estiamtion of pitch angle from Inertial System (red) , and from Visual Odometry (blue)

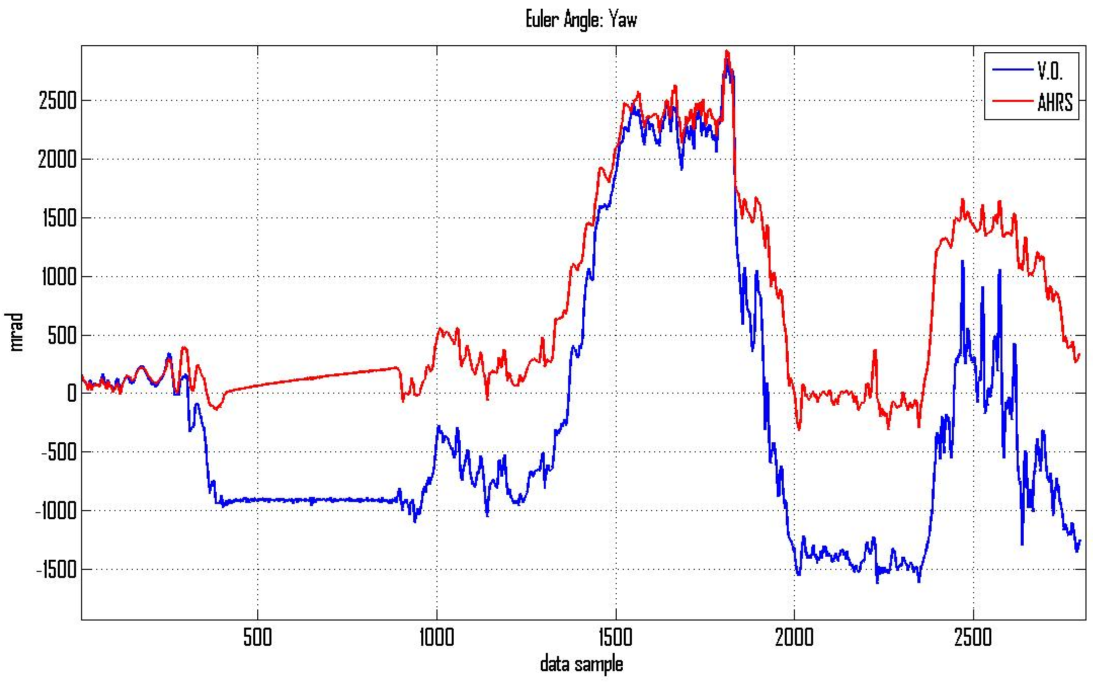


Figure 24: Estiamtion of yaw angle from Inertial System (red) , and from Visual Odometry (blue).

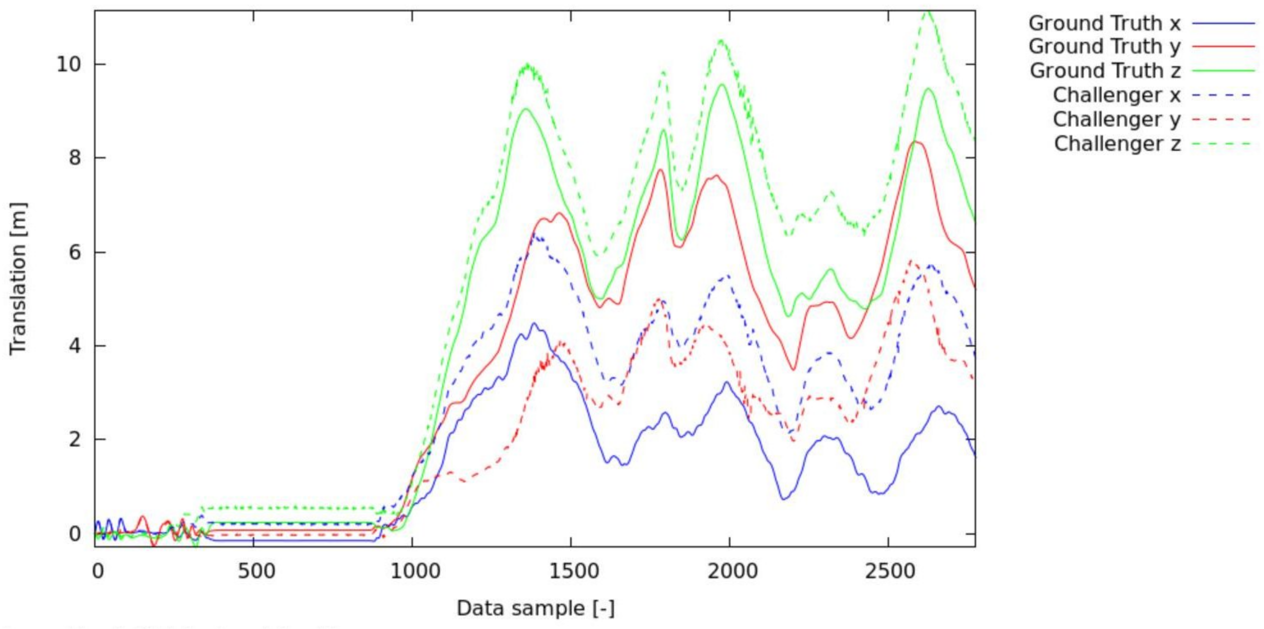


Figure 25: Difference between the groung truth, provided by EuRoC, and the estimation of the translations comes from Visual Odometry without attitude correction

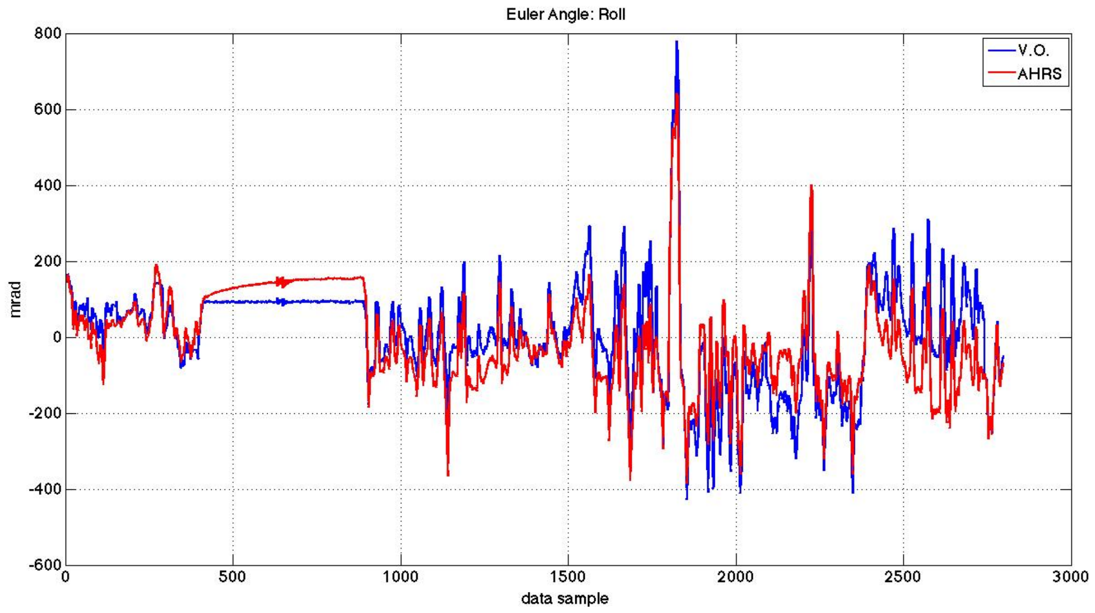


Figure 26: Estiamtion of roll angle from Inertial System (red) , and from Visual Odometry with attitude correction (blue)

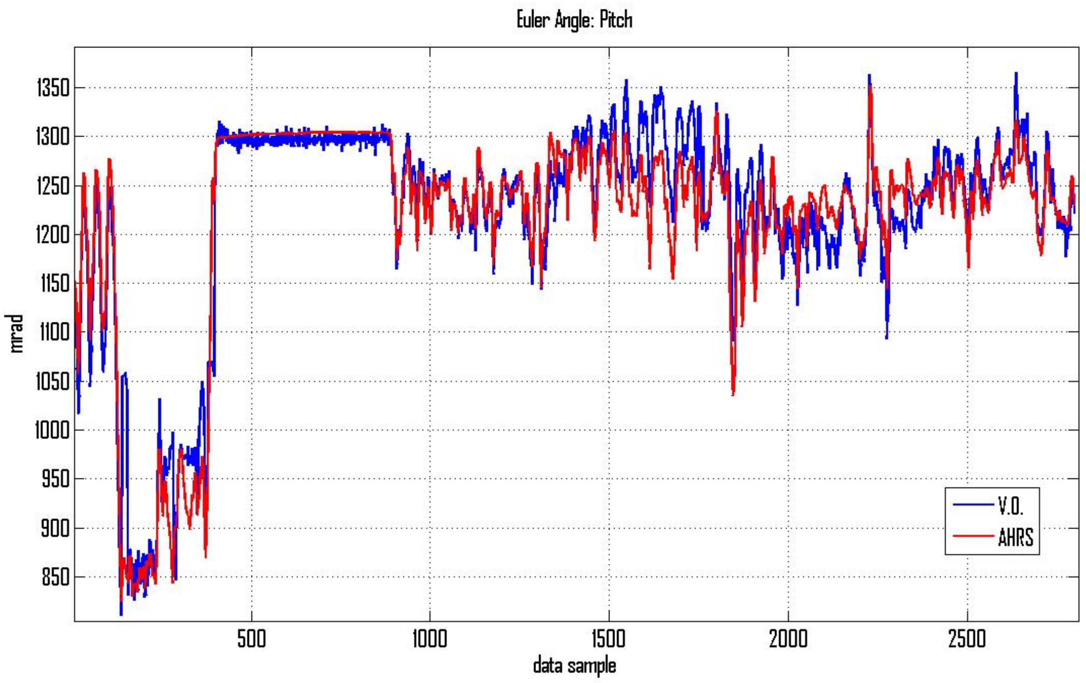


Figure 27: Estiamtion of pitch angle from Inertial System (red) , and from Visual Odometry with attitude correction (blue)

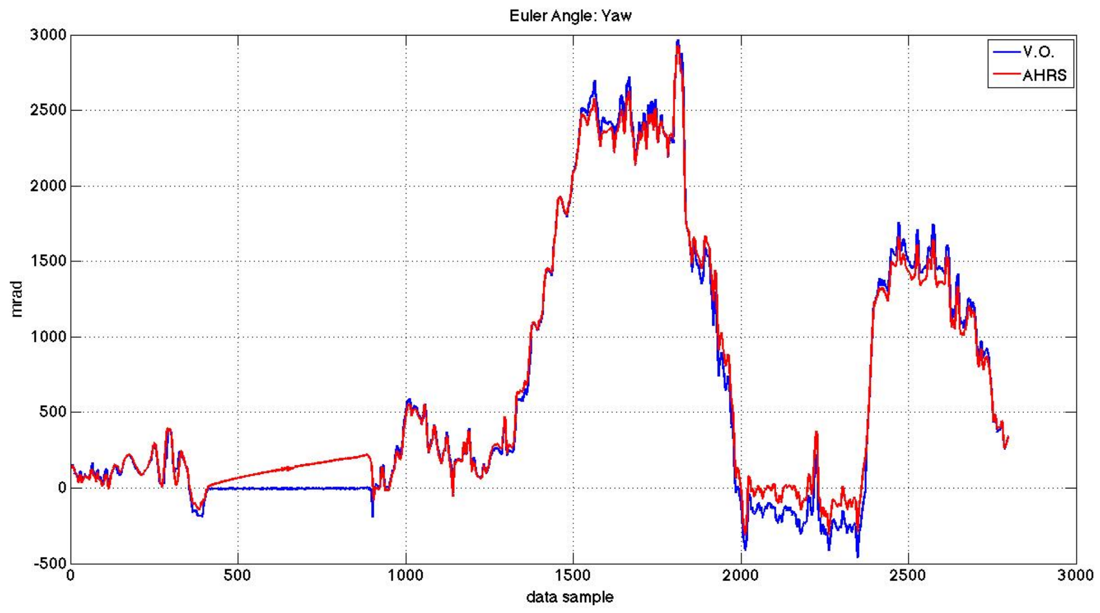


Figure 28: Estiamtion of yaw angle from Inertial System (red) , and from Visual Odometry with attitude correction (blue).

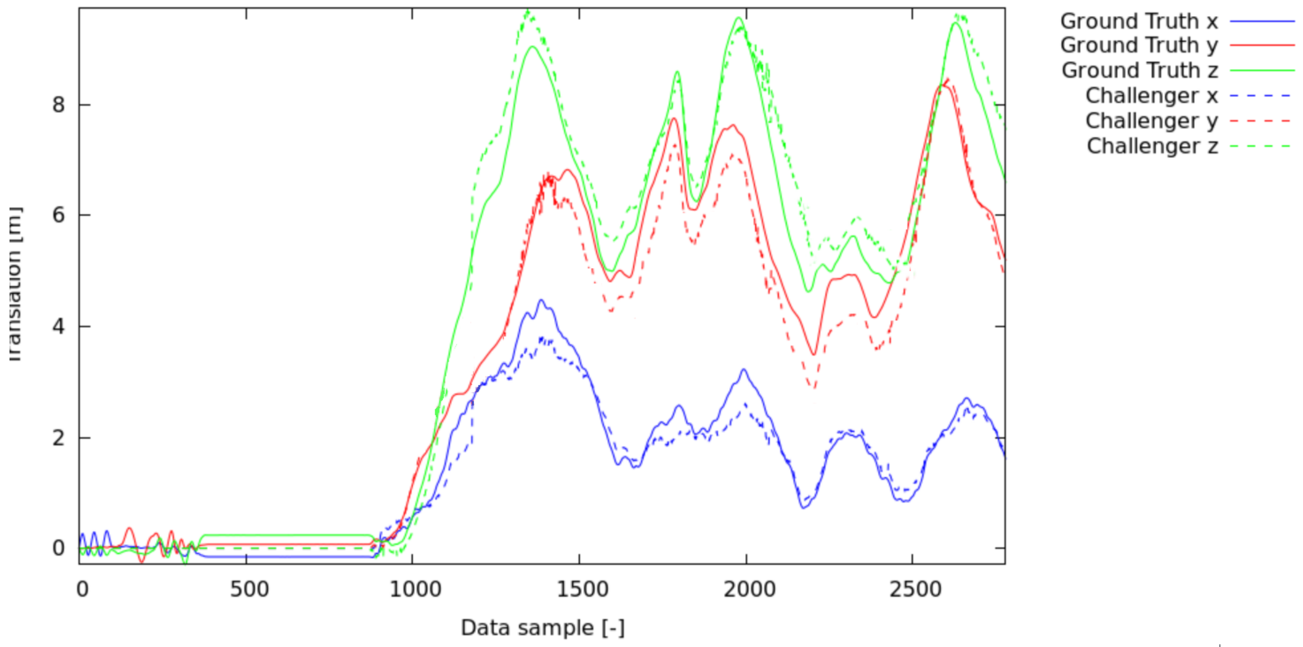


Figure 29: Difference between the groung truth, provided by EuRoC, and the estimation of the translations comes from Visual Odometry with attitude correction

8 Conclusion

The Visual Navigation System one provides good performance of pose estimation when lots of inliers are available. However the presence of outlier may affect the next pose estimations. So it 's necessary to detect the presence of outlier and use informations that can correct errors on pose estimation. In this thesis we have used the Inertial Navigation System to correct errors on attitude estimates.

The proposed Visual Navigation System integrated with the attitude estimation from the Inertial System provides better performances respect to the only use of the Visual Odometry as the tests with real data have, and in the virtual enviroment realized using ROS and Gazebo shown.

References

[1] G. Bradsky and A. Kaehler: *Learning OpenCV, Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[2] Robert M. Rogers : Applied Mathematics in Integrated Navigation System. Third Edition (AIAA Education), Hardcover , 2007

[3] R. Hartley and A. Zisserman: *Multiple view geometry in computer vision* . Cambridge University Press, New York, NY, USA, 2000

[4] http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstr
OpenCV , Camera Calibration and 3D Reconstruction

[5] http://www.ros.org/, ROS.

[6] David G. Lowe: *Distinctive Image Features from Scale-Invariant Keypoint*, Computer Science Department University of British Columbia Vancouver, B.C., Canada 2004

[7] Mahony, R : "Nonlinear Complementary Filters on the Special Orthogonal Group", Automatic Control, IEEE Transactions on (Volume:53 , Issue: 5), June 2008.

[8] http://www.euroc-project.eu/