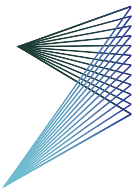
University of Zurich
Department of Informatics



ROBOTICS &
PERCEPTION
GROUP

Quim Sànchez Nicuesa

Fast vision-based relocalization for MAVs

Semester Thesis

Robotics and Perception Group
University of Zurich

Supervision

Christian Firster
Dr. Nuno Gracias
Dr. Davide Scaramuzza

June 2014

Contents

Abstract

Visual odometry techniques are becoming increasingly popular in robotic vehicles as a means to provide navigation information. This is further relevant in the context of Micro Aerial Veihicles (MAVs) where the payload constrains impose strong limitations on the choice of navigation sensors. Modern Visual Odometry algorithms, although quite sophisticated, are prone to failures when the tracking is lost due to image blur or sudden large changes in the image content. The loss of the tracking requites finding the location of the vehicle with respect to a previously built map (kidnapped robot problem). In this work we study and propose some relocalization solutions for Visual Odometry algorithms. This work is intended to work with SVO (Semi-direct Visual Odometry) but the proposed framework can be used with other methods. The relocalization method from PTAM is used as a base line and new alternatives based on space geometry and machine learning are proposed.

Chapter 1

Introduction

Micro Aerial Vehicles (MAVs) are about to play a major role in tasks like search and rescue, environment monitoring, security surveillance, inspection, goods delivery (Amazon), etc. However, for such operations, navigating based on GPS information only is not sufficient. Fully autonomous operation in cities or other dense environments requires MAVs to fly at low altitudes where GPS signals are often shadowed or indoors, and to actively explore unknown environments while avoiding collisions and creating maps. Precisely autonomous operations requires MAVs to rely on alternative localization systems. For minimal weight, power consumption and budget a single camera can be used.

Real-time monocular Visual Odometry (VO) algorithms can be used to estimate the 6 DoF pose of a camera relative to its surroundings. This is attractive for many applications such as mobile robotics (and not only aerial) and Augmented Reality (AR) because cameras are small and self-contained and therefore easy to attach to autonomous robots or AR displays. Further, they are cheap, and are now often pre-integrated into mobile computing devices such as PDAs, phones and laptops.

SVO (Semi-direct Visual Odometry) [6] is a very fast VO algorithm able to run at more than 300 frames per second on a consumer laptop. It builds a map based on keyframes and salient points. Most monocular VO are feature-based where scale and rotation invariant descriptors (SIFT, SURF...) are extracted and matched in order to recover the motion from frame to frame while finally refining the pose with reprojection error minimization with the map. SVO uses a different approach by using direct methods. Instead of matching descriptors, it uses intensity gradients to minimize the error between patches around detected salient points to estimate the frame to frame transformation. Finally, it uses Bundle Adjustment to align with the map and avoid or minimize drift.

The main problem with most existing monocular VO implementations (including SVO) is a lack of robustness. Rapid camera motions, occlusion, and motion blur (phenomena which are common in all but the most constrained experi-

mental settings) can often cause the tracking to fail. While this is inconvenient with any tracking system, tracking failure is particularly problematic for VO systems: not only is the camera pose lost, but the estimated map could become corrupted as well.

This problem is accentuated during a fast agile maneuver (e.g., a flip) and so a good relocalization is important when these are intended to be performed.

1.1 Objectives

The objective of this work is to investigate and implement different monocular vision-based relocalization algorithms which should work in the following scheme:

- In a training stage, the vehicle explores the environment where the relocalization is supposed to occur. During this stage, an appropriate representation of the scene is created.
- An event happens that causes the vehicle to loose tracking and get lost. For example the vehicle executes an agile maneuver during which vision-based tracking is no feasible.
- During the actual relocalization phase, the 6 DoF pose in the previously built map must be recovered as fast as possible.

1.2 Structure of the thesis

First of all, in chapter 2 some important background to follow the thesis is introduced. Next, in chapter 3 the different proposed methods are explained. Then, in chapter 4 these same methods are evaluated. Finally, in chapter 5, confusions are introduced, followed by a proposed future work. In appendix A, some details of the software implementation are discussed.

Chapter 2

Background

This section overviews the most relevant related work and presents a background information useful to understand the proposed approach.

2.1 Related Work

2.1.1 Place Recognition

Klein and Murray present in [11] the relocalization method used in PTAM [10]. PTAM is a VO algorithm based on keyframes that are used during the relocalization. The relocalization method consists of two steps. First, given the current frame, the most similar keyframe is retrieved, and its know pose is used as a baseline. As a measure of similarity the cross correlation, being the difference between subsampled, blurred and zero-mean images is used. The small blurry images are stored every time there is a new keyframe and the small blurry image of a new frame is computed during the relocalization to be compared with the keyframes.

Other methods can be used for image retrieval, for example using bag of words [18]. Nistér and Stewenius [15] propose to use a tree structure to store words in order to handle much larger vocabulary or have a much faster retrieval. Every node of the tree would have k child nodes which are the clustering results of k-means. The tree is build by recursive k-means. This structure is expensive to build because k-means is very resource consuming. During the online process, new words can be appended to the final leaves.

Özuysal et al. [16] proposes a simplified random forest classifier which relates image patches to objects. It is simplified because instead of using a tree structure, they use a linear structure applying all the binary tests to the patch. The result of the tests is a binary descriptor, the list of binary tests is called Fern. Every object is trained with multiple random warps of the known view to introduce

information from possible different views of the object. In the end every object can be represented with many binary descriptors and every descriptor should output a probability distribution of possible objects represented. Evaluating multiple Ferns and joining the produced distributions, the final classification is achieved.

2.1.2 Pose Estimation

Geometric methods are typically used to find the transformation from the found keyframe using the classic pipeline of salient points detection, feature extraction and matching. The five-point algorithm can then be used to find the scaled 6 DoF transformation or the full 6 DoF with the three-point algorithm if depth is known [9].

During the second step of the relocalization, the transformation from the retrieved frame to the query frame must be calculated. This transformation will be finally appended to the know keyframe pose. In PTAM, an image alignment algorithm, Efficient Second-Order Minimization method (ESM) [3], is employed. ESM is a Gauss-Newton gradient descent algorithm, which can be used with different image warp functions. It is similar and based on the Lucas-Kanade [1] algorithm but using Second-order functions; therefore, results in a faster convergence.

2.1.3 Joint Place and Pose estimation

One approach to solve the relocalization problem was proposed by Williams [21]. In their implementation, they use Random Forest classifiers to characterize a salient object in space. To do so, the classifier needs to be trained with as many as possible representations of the object (multiple views). Therefore, the first time an object is found, multiple warps of the patch are used to initialize its presence in the classifier. On later encounters with the object, the classifier is incrementally trained with additional data. During the relocalization phase salient points are classified using the trained classifier and the three-point algorithm is used to recover the 6 DoF position. This method is memory expensive and requires a GPU to generate the patch warps.

Shotton et al. [17] also propose a method to solve the place recognition and pose estimation problem simultaneously using random forests. RGB-D data is used to train the classifier. In this case, all the information is encoded in the classifier so no previous data storing or computing (salient point detection, descriptor extraction, etc...) is needed. The classifier is trained to an individual RGB-D pixel, and an RGB-D pixel query will output a probability distribution over the position in $\mathbb{R}^3$. This can be applied to all pixels of a frame or to a sparse subset selection of them. Ideally, the camera pose can be inferred from only three pixels, but as the output of the classifier can be very noisy, a second

step is applied. From the output from many pixels an energy function is minimized using preemptive RANSAC in order to find a pose that agrees with most of the distributions.

To train this method, a very complete dataset of RGB-D images with 6 DoF poses from the environment associated to them is needed. That makes it difficult to be used with SLAM problems where the map get populated incrementally. An online training method should be developed.

2.2 Methods

2.2.1 Semi-Direct Visual Odometry

SVO [6] is an algorithm used to track the pose of a camera over time based on keyframes. It is divided in two main parts. On one side there is the creation and maintenance of a map, and on the other side, the motion estimation.

The map consists of images, its pose and world frame points. World points are computed from featured points in the image. To find the depth of these points the following depth filter is used. When a point is first observed a possible depth distribution is initialized with a high uncertainty. This distribution is then updated in posterior observations of the same point in a Bayesian fashion. This process is illustrated in 2.1.

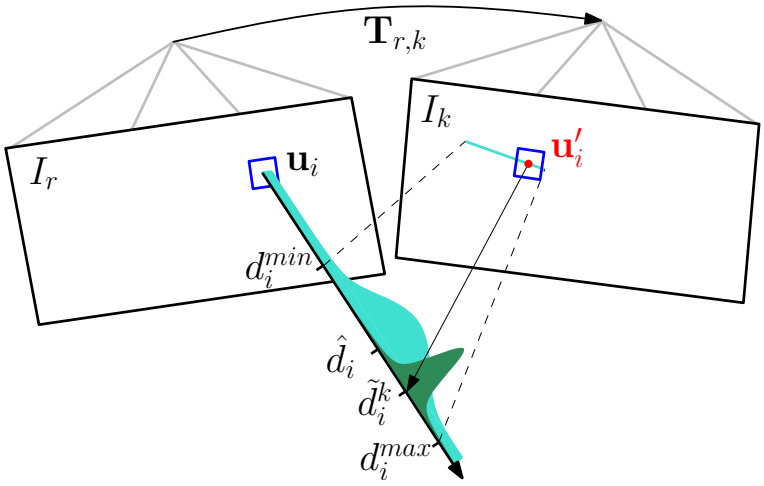


Figure 2.1: SVO depth filter [6]

Given a point in the reference frame, its corresponding location in a new image is found by tacking a patch around the point in the reference frame, and correlating along the epipolar line in the new image. The point whose patch

has maximum correlation is taken as match. Once the uncertainty of the depth distribution is low enough, its accepted in the map and used in the tracking step.

Featured points are only computed in keyframes and then tracked in the following frames. A new keyframe will be selected after a certain translation threshold is passed.

To find the pose of a new incoming frame, the following steps are performed: first, a sparse image alignment 2.2 over $SE(3)$ [20] is performed between the new frame and the last one. That is, patches from featured points whose depth is know from the last frame are projected into the new frame. The difference between patches is an error function on the transformation that is then minimized using an iterative Gauss-Newton procedure.

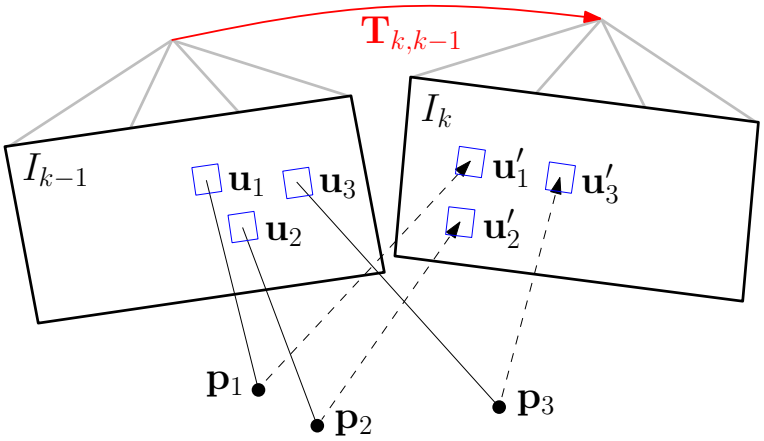


Figure 2.2: SVO sparse image alignment [6]

After this step a global pose of the frame is found. To improve the results not only the last frame should be used. All the points from the map, that are visible to the frame, are then projected on it straight from the keyframe where they were initially found and a similar alignment is performed as seen in 2.3. In this case, as the used frames might be farther away and the used patch is larger, an affine warp is applied to the patch.

Finally, Bundle Adjustment is applied to the map to optimize the pose of keyframes and the 3D location of the world points 2.4.

2.2.2 Interest point detection, descriptor extraction and matching

The goal here is, given a pair of images, find points in both images that represent the same object in the world. Two things are important to achieve this. First,

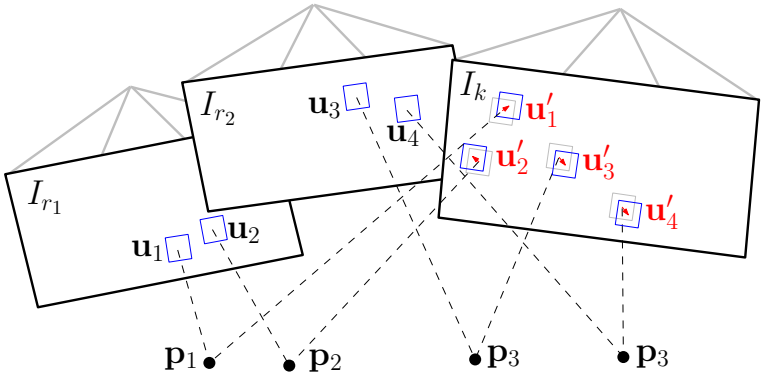


Figure 2.3: SVO feature alignment [6]

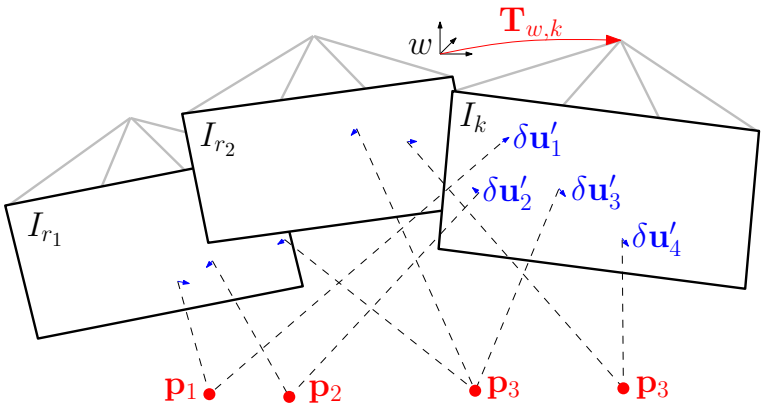


Figure 2.4: SVO global optimization [6]

the point detection should be repeatable. If one point is detected in one image then it should also be detected in the second. And second, given a pair of points from two different images, there must be a way to tell if they are the same or not.

To achieve the repeatability, only salient points are used. These are points that, under some criterion can be distinctive from their surrounding. An early criterion for saliency was the measure of cornerness. Corners in an image are easily distinguishable. One salient point detector that uses point's cornerness is the Harris detector [8]. Later on, many other detectors where developed, such as the Hessian or Laplacian detectors [2], the Difference of Gaussian detector from SIFT [14], which tries to approximate the Laplacian.

Finally, to be able to tell points from different images apart, a way to compare them is needed. Intensity around to point could be used to compare, but a rotation or a change on illumination could easily, among others, make this method very unstable. To solve this problem different invariant descriptors have been developed. Descriptors are a set of N values that encode the appearance of the point. This N values can be regarded as a point in a N-dimensional space, and the distance between those can be used as a measurement of similarity.

The most well known descriptor and still one of the best performing one is SIFT [14]. It is a descriptor based on a histogram of gradient directions. The fact that it is based on gradient makes it invariant to illumination changes, while the fact that the histogram is sorted makes it invariant to rotation.

During the matching process, from two sets of descriptors, the search of the nearest neighbour of every of the values in one set is performed in the other leading to a list of pairs. This might lead to false positives like false associations and one to many associations which should be taken care of in the subsequent posterior processes. To avoid this, most image registration approaches use robust estimators [5] that are ale to identify and reject outliers.

Chapter 3

Approach

We propose two different approaches to address the relocalization proble. First, a local approache is based on the PTAM implementation, where two steps are performed. The first step has been named *Place Finder* and the second *Real Pose Recognition*. Multiple methods will be proposed to solve the second step. Then, on the other side, a global approach will be proposed. In this case machine learning methods (*ferns*) will be used to recognize points in space.

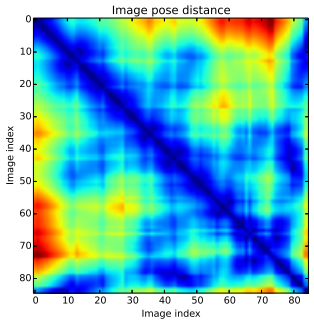
3.1 PTAM method

PTAM [10] is a VO algorithm based on keyframes and so the relocalization method proposed is based on keyframes as well. Every keyframe is associated with a camera pose that will be used to relocalize. During the relocalization there are two steps involved. We called the first step *Place Finder* and the second *Real Pose Finder*.

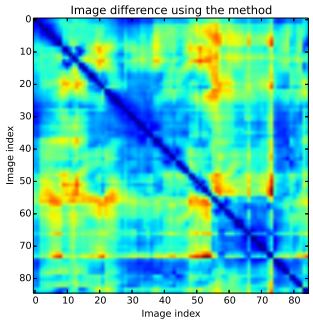
3.1.1 Place Finder

During this step, the algorithm tries to find the keyframe image most similar to the last acquired image. The pose associated with the most similar keyframe is used as an initial rough estimation of the current pose. The similarity score should be invariant to small view point changes because the new acquired image will, most probably, never be taken from the same pose as any of the keyframes. Also it should be fast to compute.

The used similarity score is the Cross Correlation between images meaning the sum of the squared error between two zero-mean images as in 3.1. To make to computation faster both images are resized become 40×30. Then, to make the images more resistant to view point changes it is blurred with a 3×3 Gaussian kernel with $\sigma = 2.5$. The resulting image is a resized, blurred and zero-mean



(a) Image to image real distance between the keyframe pose



(b) Image to image distance approximated using the Cross Correlation value

Figure 3.1

image called *small-blurry-image*.

$$d_{CC} = \sum_{x,y}[(I(x,y) - \bar{I}) - (G(x,y) - \bar{G})]^2 \tag{3.1}$$

During the normal map building pipeline this image is computed and stored every time a new keyframe is added to the map. And then, during the relocalization, the sum of squared difference between every stored *small-blurry-image* and the last acquired frame is computed, to find the most similar keyframe, which is then use its pose as an initial estimation of the current camera pose.

Method validation

To evaluate the method the real distance between two frames, distance between camera centres in 3D space, is going to be compared with the Cross Correlation value described above. Far away frames should be dissimilar and close by frames should be more similar and have a lower CC value. In Figure 3.1a can be seen the pair distances between image using the real pose while in Figure 3.1b there is the approximated pair distances using the CC value as distance. It can be seen that they have a similar distribution.

Finally, to show that this method can be used, for every image the most similar, in CC seance, was taken being it actually the Kth closest image. Ideally the most similar image should always be the closest image. In Figure 3.2 there is the count of occurrences of each K. It can be seen that most images resolve to the first or second closest image using this method.

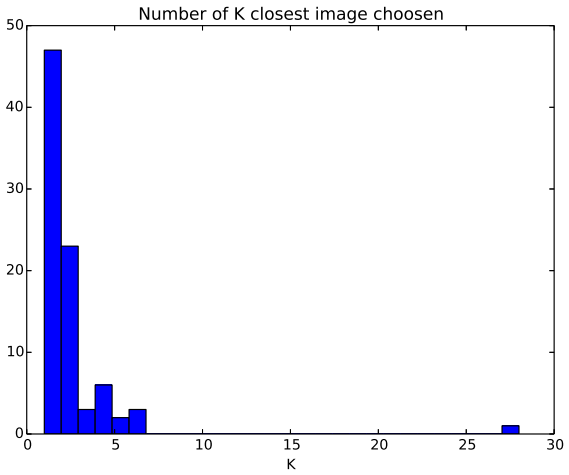


Figure 3.2: Distribution of closest chosen images using CC

In Figure 3.3 demonstrates what the algorithm would return for a given query image for which there is no pose information available. Figure 3.3a is the image seen by the camera at the moment of the relocalization and 3.3b is what the system found to be the most similar, closest, image using the cross correlation procedure explained previously. In this case the two camera poses are not exactly the same, but they should be similar enough in order to find the lose difference in posterior steps.

3.1.2 Real Pose Finder

The second step of the PTAM relocalization algorithm found, in order to refine the pose of the most similar keyframe to explain the current pose of the camera. During this step, in the implementation from PTAM, only rotations are corrected. An image alignment through optimization algorithm (ESM [13]) is used to find the $SE(2)$ transformation between the two, followed by a minimization to find a transformation in the world frame.

Image alignment

The Extended Second order Minimization algorithm (ESM) is based on the algorithm proposed by Lucas and Kanade [1] in 1981. The goal of both algorithms is to align one template image T to a different input image I through a parametrised warping function.

With this goal an error function is defined on which the Gauss-Newton or

(a) Query image

(b) Found closest image

Figure 3.3

Levenberg-Marquardt schemes can be applied. The error is the squared difference between the template image and the warped input image

$$e = \sum_x [I(W(x;p)) - T(x)]^2 \tag{3.2}$$

The warping function is going to be in $SE(2)$, that is, translation and rotation of the image plane.

$$W(x;p) = \begin{bmatrix} cos(\alpha) & sin(\alpha) & t_x \\ -sin(\alpha) & cos(\alpha) & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} xcos(\alpha) + ysin(\alpha) + t_x \\ -xsin(\alpha) + ycos(\alpha) + t_y \end{bmatrix} \tag{3.3}$$

where $p = \big(\alpha, t_x, t_y\big)$ are the $SE(2)$ parameters and $x = \big(x, y\big)$ is a pixel coordinate.

The algorithm assumes that a current estimate of p exists and iteratively tries improves it by increments Δp. On every iteration equation 3.4 is solved on Δp and then it is used to update p as in 3.5

$$\min_{\Delta p} \sum_x [I(W(x;p+\Delta p)) - T(x)]^2 \tag{3.4}$$

$$p \leftarrow p + \Delta p \tag{3.5}$$

To solve question 3.4 it first needs to be linearised. With this goal, a first Taylor expansion on $I(W(x:p+\Delta p))$ is carried on leading to

$$\sum_x [I(W(x;p)) + \nabla I \frac{W}{\partial p} \Delta p - T(x)]^2 \tag{3.6}$$

where ∇I is the gradient of the input image and $\frac{\partial W}{\partial p}$ is the Jacobian of the warping function. In this case the Jacobian of 3.3 is

$$\frac{\partial W}{\partial p} = \begin{bmatrix} -xsin(\alpha) - ycos(\alpha) & 1 & 0 \\ xcos(\alpha) - ysin(\alpha) & 0 & 1 \end{bmatrix} \tag{3.7}$$

The solution that minimizes 3.4 on Δp can be found in a least squares sense. The equation needs to be derived and then set it equal to zero. The partial derivative of 3.4 with respect to Δp is:

$$2 \sum_x \left[\nabla I \frac{\partial W}{\partial p} \right]^T \left[I(W(x;p)) + \nabla I \frac{\partial W}{\partial p} \Delta p - T(x) \right] \tag{3.8}$$

Then, setting 3.8 equal to zero to find where it is minimized (or maximized) by Δp and isolating Δp the next closed form solution can be found:

$$\Delta p = H^{-1} \sum_{x} \left[\nabla I \frac{\partial W}{\partial p} \right]^{T} [T(x) - I(W(x;p))] \tag{3.9}$$

Where H is the Gauss-Newton approximation of the Hessian matrix:

$$H = \sum_{x} \left[\nabla I \frac{\partial W}{\partial p} \right]^{T} \left[\nabla I \frac{\partial W}{\partial p} \right] \tag{3.10}$$

The ESM algorithm used in PTAM is very similar to the derived above with the difference that while the Lucas-Kanade takes the gradient from the input image, ESM used both the gradient of the input image and the gradient of the template image and averages them.

$$\nabla I = \frac{1}{2} \left[\nabla I_t + \nabla I_q \right] \tag{3.11}$$

$$\textit{steepest descent} \text{ image} = \nabla I \frac{\partial W}{\partial p} \tag{3.12}$$

What in [1] is referred as *steepest descent* image 3.12 in PTAM and other references is the Jacobian. That is, the gradient of the image multiplied by the Jacobian of the warp function. Taking the used Jacobian 3.7 the *steepest descent* image would be

$$\begin{aligned} \textit{steepest descent} \text{ pixel} &= \begin{bmatrix} g_x & g_y \end{bmatrix} \begin{bmatrix} -xsin(\alpha) - ycos(\alpha) & 1 & 0 \\ xcos(\alpha) - ysin(\alpha) & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} g_x(-xsin(\alpha) - ycos(\alpha)) + g_y(xcos(\alpha) - ysin(\alpha)) & g_x & g_y \end{bmatrix} \\ &\approx \begin{bmatrix} -yg_x + xg_y & g_x & g_y \end{bmatrix} \quad \text{when} \quad \alpha \approx 0 \end{aligned} \tag{3.13}$$

During the implementation, α is considered to be close to zero. This way the computation of the trigonometric functions is avoided to save computational time. This method is first applied to the last level of the scale pyramid roughly aligning the images and then refined with upper levels (used top level is a configuration parameter).

World frame interpretation

This transformation is not easily interpreted in $SE(3)$ (6 DoF including three degrees of translation and three degrees of rotation $SO(3)$), since a translation in pixels can mean different things in the world frame depending on the distance to the object. Also, in $SE(3)$ there are three angular degrees of freedom. The found transformation can be interpreted in multiple ways in the world frame. In the PTAM implementation it is assumed that the translation will only involve rotations, and so, the next step is the mapping from $SE(2)$ to $SO(3)$ (world frame rotations). The found rotations will be applied form the closes frame pose which in most cases is not in the origin of the frame. A pure rotational transformation applied on a pose not set in the origin will effect on its position as well, introducing translation.

A Gauss-Newton minimization algorithm is used to find the $SO(3)$ model that modifies the image in the same way as the found $SE(2)$ model. The minimization is over the $SO(3)$ parameters ξ and the error is expressed as in 3.14

$$\delta_i(\xi) = T_{SE(2)}u_i - \pi(T_{SO(3)}(\xi)p_i) \qquad \text{where} \quad u_i = \pi(p_i) \tag{3.14}$$

where u_i is a pixel position (during the implementation $u_0 = (5, 0)$ and $u_1 = (-5, 0)$ are used). Also the Jacobian of δ is needed during the minimization process.

$$\frac{\partial\delta(\xi)}{\partial\xi} = -\frac{\partial\pi(b)}{\partial b}|_{b=p}\frac{\partial T_{SO(3)}(\xi)}{\partial\xi}|_{\xi=0} \quad p \tag{3.15}$$

with

$$\frac{\partial\pi(b)}{\partial b}|_{b=p} = \frac{f}{z}\begin{bmatrix}1 & 0 & -\frac{x}{z}\\ 0 & 1 & -\frac{y}{z}\end{bmatrix} \qquad \text{where} \quad p = \begin{bmatrix}x\\ y\\ z\end{bmatrix} \quad \text{and} \quad f = \text{focal length} \tag{3.16}$$

$$\frac{\partial T_{SO(3)}(\xi)}{\partial\xi_k} = G_k \qquad \text{where} \quad G = SO(3)\text{ Generator} \tag{3.17}$$

Method validation

To visualize the results of the different optimization procedures described above, the found transformation has been applied to the image. In the first step, the $SE(2)$ transformation between two image is computed and the transformed image at every step is one of the used variables. In Figure 3.4 can be seen the

final transformation found from 3.3b to 3.3a. In Figure 3.6 the error of the overlapped images is visualized. It can be seen that translation and rotation are well corrected but there still is a misalignment caused mostly by a change on scale which is not taken into account during the alignment.

Figure 3.4: $SE(2)$ transformed image

On the other side, during the second minimization where a $SO(3)$ translation is found, no image is actually involved, only two pixel coordinates. To generate a visualization of the translation the calibration of image unit plane, every pixel is unprojected form the image to the image plane, then the transformation is applied to its position and finally it is projected again to the image. The result of the described procedure can be seen in Figure 3.5.

3.1.3 Real Pose Finder Alternative

During the mapping of an area, the VO algorithm finds landmarks in the world frame which are associated with detected featured points in keyframes (i.e. every featured point in an image is related to a $3D$ position in the world frame). Given a new image, some extracted featured points can be related to a keyframe using descriptor-matching which at the same time are related to world positions. From this information the full 6 DoF translation $SE(3)$ can be computed using the prospective three-point algorithm (P3P).

The three-point algorithm and its implementation is described in [12]. In our case the described *Central absolute pose* is used.

First, descriptors from every featured point are extracted (both SIFT and SURF can be used updating a configuration file, but in the experiments only SIFT is

Figure 3.5: $SO(3)$ transformed image

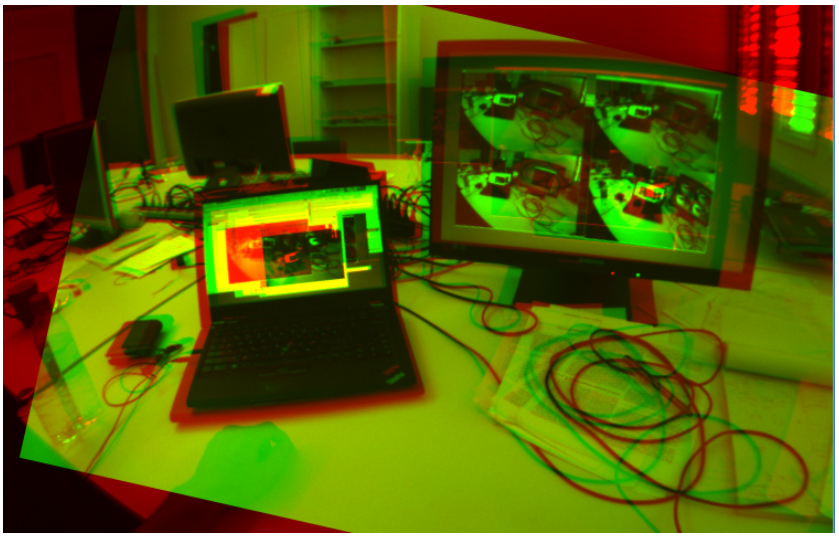


Figure 3.6: $SE(2)$ transformation error visualization

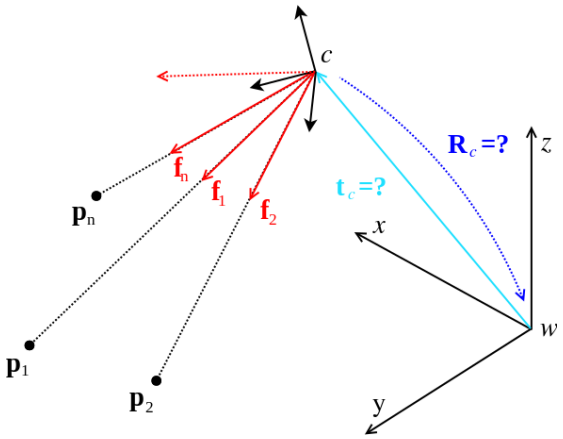


Figure 3.7: From camera frame vectors f (or image pixels if the camera calibration is available) and fix points p the transformation between the two coordinate frames can be computed using the P3P algorithm. Image taken from [12]

used). Second, a brute force KNN matching is performed from all the extracted descriptors from on image and all the descriptors of the second image. The first, and second most similar descriptor are retrieved. Then, only good matches are kept, that is, using the matching technique described by Lowe [14], only matches with a descriptor ratio between the first and the second closest match of 0.8 or less are kept, only discriminant matches are used.

Finally, the three-point algorithm is fed with the pixel positions from the query image and the landmarks from the other. Because there are still outliers after the described simple filtering, this process is run in RANSAC [5] framework.

Method validation

Figures 3.8 and 3.9 are an example of the described above. The pose output was used to successfully relocalize in SVO.

The first part, where descriptors are extracted, matches and filtered, can be seen in Figure 3.8. It can be seen that most matches are correct after this first filtering.

The inliers found during the RANSAC process can be seen in 3.9. One has to keep in mind that the model which is fitted is not from image to image but from world frame, using world points, to camera frame points (projections). These world points can not be easily visualized on the image but only its projection. For this reason matches, that might look correct, are not used as inliers because its 3D point might be wrong or inaccurate not fitting the model.

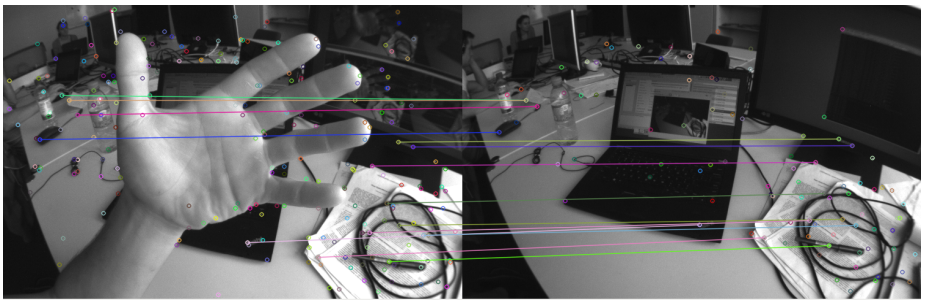


Figure 3.8: Accepted matches using SIFT

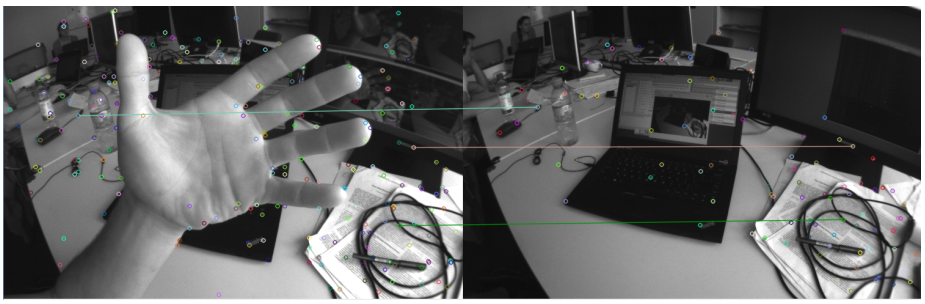


Figure 3.9: Inliers from RANSAC used to calculate the pose with the three-point algorithm

3.2 Using ferns

As said previously with the three-point algorithm it is possible to recover the 6 DoF of the camera pose from the relation from pixel coordinates and points in space. In this case machine learning techniques are used to model this relationship. In the classifier scheme, an object in space is a class and multiple views seen from the camera should all be classified as this class.

A *fern* [16] is a descriptor made from a set of binary tests such as 3.18, build like random forests but flattened (no tree structure). When used as a classifier, every possible evaluation of a *fern* will contain a posterior probability distribution for every class, like random forests' final leafs do.

$$f_j = \begin{cases} 1, & \text{if} \quad I(d_j, 1) < I(d_j, 2) \\ 0, & \text{otherwise} \end{cases} \tag{3.18}$$

The result of the tests is encoded into the bits of an unsigned integer as is it shown in Listing 3.1. If a *fern* has S tests then its representation can have $K = 2^S$ possible values which are then used as index of its posterior distribution.

Listing 3.1: Fern evaluation

```
uint8_t fern = 0;
for (size_t j = 0; j < S; ++j)
{
  //Shift bits
  fern <<= 1;
  if (I(d_j,1) < I(d_j,2))
  {
    //Change last bit
    fern++;
  }
}
```

One fern is usually not descriptive enough to correctly classify. In [16] it is claimed that with 50 ferns and $S = 11$ a problem with 200 different classes is tractable. Regarding storage requirements, it would involve $50 \times 2^{11} \times 200 = 20480000$ elements to be stored in memory. If these are stored as *float* then 78 MB are needed, which is tractable. It should be noticed that the problem does not scale well with the number of tests S, but at the same time, it is the most critic parameter as illustrated in the next subsection.

3.2.1 Seminaive Bayesian Approach

Let $c_i, i = 1, \ldots, H$ be a set of classes and $f_j, j = 1, \ldots, N$ be a set of binary tests 3.18 calculated over an image patch that is being classified. Formally, we are looking for

$$\hat{c}_i = \underset{c_i}{\operatorname{argmax}} \quad P(C = c_i | f_1, f_2, \ldots, f_N) \tag{3.19}$$

then, Bayes' formula tells

$$P(C = c_i | f_1, f_2, \ldots, f_N) = \frac{P(f_1, f_2, \ldots, f_N | C = c_i) P(C = c_i)}{P(f_1, f_2, \ldots, f_N)} \tag{3.20}$$

Assuming a uniform prior and since the denominator is a scaling factor the problem is reduced to

$$\hat{c}_i = \underset{c_i}{\operatorname{argmax}} \quad P(f_1, f_2, \ldots, f_N | C = c_i) \tag{3.21}$$

Since these tests are very simple, many of them are needed ($N \approx 300$), the storage of joint probabilities for every outcome is not feasible because 2^N entries for each class would be required. One way to avoid this storage issue is to assume

total independence between tests. This way the class probability distributions can be calculated as

$$P(f_1, f_2, \ldots, f_N | C = c_i) = \prod_{j=1}^{N} P(f_j | C = c_i) \tag{3.22}$$

However this ignores the possible correlation between tests. With ferns a trade off is achieved, since the tests are grouped in sets F and then the conditional probability becomes

$$P(f_1, f_2, \ldots, f_N | C = c_i) = \prod_{k=1}^{M} P(F_k | C = c_i) \tag{3.23}$$

It can be seen here why S is important. An increment in the number of tests in a *fern* means an increment in the modelled correlation.

3.2.2 Training

During the training step many different views of a world point are needed to correctly model it. In the original work by Ozuysal and Calonder [16] only one image is used to train. To generate more possible views of the objects, multiple (about 10,000) randomly generated warps are applied to it. All these warped images are used for training. Those warps include affine transformations, noise addition and smoothing with a Gaussian kernel (with all the parameters randomly picked from a uniform distribution).

In the studied case here, the algorithm can already provide multiple real views of the object (around 5-10), but still on each patch many randomly generated warps are applied and used to train (usually 100). In this case, the warps are simplified to include only image rotation and scale.

During the training, every warped patch is evaluated with every *fern*. The count of a class evaluation on each *fern* is performed along with the count of per class patches used. Those values are used during the classification step to calculate the probability distributions

$$p_{k,c_i} = \frac{N_{k,c_i} + N_r}{N_{c_i} + K \times N_r} \tag{3.24}$$

where N_{k,c_i} is the number views of class c_i that evaluates to k and N_{c_i} is the number of views of class c_i to train. A regularization term N_r is used to overcome untrained situations that would would evaluate to 0 probability. In the

original work different regularization terms N_r are tested all leading to similar results, they conclude that its value is not important but it presence is, it is usually set to 1. The implemented classifier sores c_i and N_{k,c_i}, allowing online training by evaluating new incoming images and featured points.

In the current implementation, no online training is performed because training one frame is already computationally expensive, and it cannot be done at every new keyframe from SVO without drastically lowering the amount of frames treated per second. Instead, one global training is triggered manually when it cannot disturb the MAV's flight, for example when landed. Online training could be implemented on a different thread if the processor used had unused cores.

Method Validation

To validate the classifier a small test has been set up. One image is taken and from it 100 random patches are used to train the classifier. Then, 100 random warps are applied to each one these patches and are classified. This process is reproduced 100 times because it is a random process. The ratio of well classified warped patches with contrast to the number of *ferns* used and the number of tests per *fern* can be seen in the next plot.

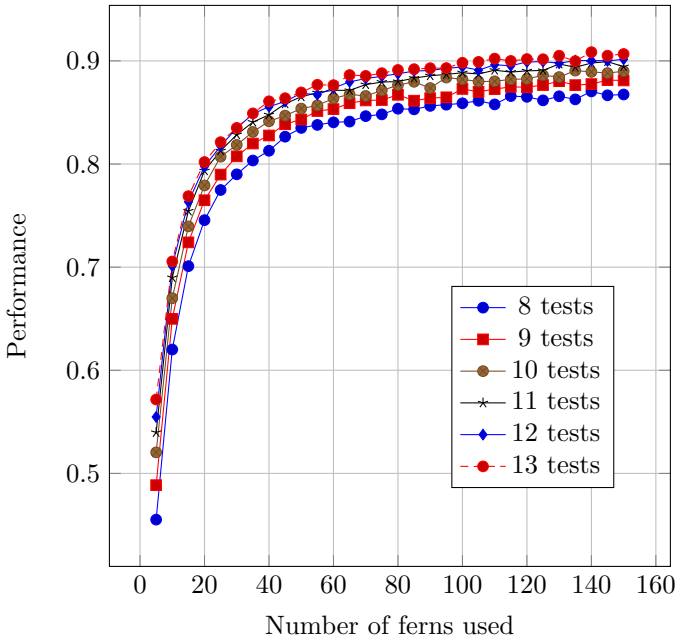


Figure 3.10: Ferns classifier evaluation

It can be seen that a minimum number of *ferns* is needed to achieve the best performance for a given number of tests, between 50 and 80. After that, more

tests cannot improve the results. On the other side, it can be seen that incrementing the number of tests per *fern* can give a qualitative jump, even with only one test. As said before, the number of tests S is an important parameter, but it is very limited by the available memory.

Chapter 4

Experiments

Discussion and details about the software implementation can be found in Appendix A.

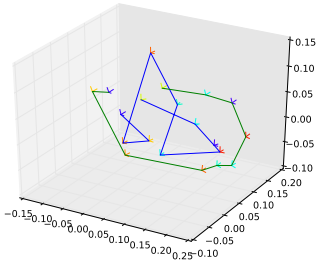
Here all the described algorithms are going to be applied to different datasets. As described, the Multi Relocalizer consists of two parts *Place Finder* and *Real Pose Finder*. Both parts will be analysed independently and finally together. Then, the global method using *ferns* will be analysed.

The SVO frame is initialized with with the algorithm with the first few frames. A initial known depth is taken from the configuration and used to find the transformation between the two first frames. This known depth might not relate completely with reality leading to a non standard space metric (poses are not in meters). The following results and its errors are in this metric and should be considered as relative to other solutions and not as absolute.

4.1 Desktop dataset

The first used dataset has been captured on a desktop. The first part of the dataset will be used for training the relocalizer while the second part will be used for testing. From both parts the pose is known. It will be used, not only to visualize the error as ground truth, but also in the *naive Place Finder* described later.

In figure 4.1a both the training and testing data can be seen together, where every set contains 10 images that cover a space of about 2 square meters. It should be noted that the training and the testing frames do not share positions. Using only closest frame will not yield good results, and so the *Real Pose Finder* step is very important. In 4.1b there is an example of the images in the dataset.



(a) In green, the path used for training. In blue, the path used for testing

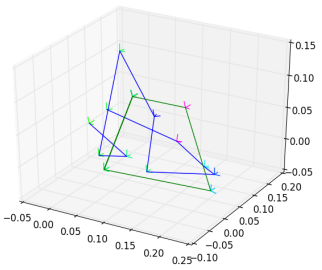
(b) Example of an image from the dataset

Figure 4.1

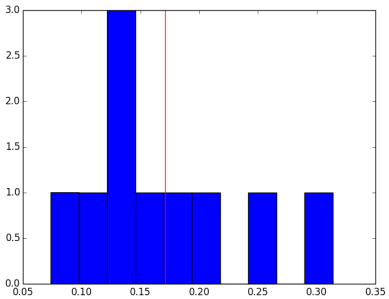
4.1.1 Multi Relocalizer

Naive *Place Finder*

To analyse the performance of the different *Real Pose Finder* methods a *naive* implementation of a *Place Finder* is used, it uses the ground truth information to find the real closest frame, being it an ideal *Place Recognition* method. In 4.2a there is the result using this method, it can be seen that even being ideal it is not so good. Next, different *Real Pose Finder* will be used to correct this situation.



(a) In blue, testing ground truth. In green, found path



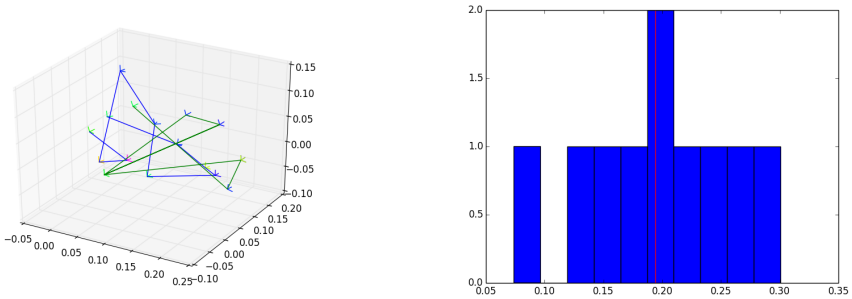
(b) Translation error histogram, with the mean marked as a red line

Figure 4.2: Results using the Naive *Place Finder* an no *Real Pose Finder*

To evaluate the results in a numeric way, a histogram of the translation error have been plotted in 4.2b and is going to be used as a baseline to see how posterior methods can improve it.

Cross Covariance *Place Finder*

As said, the naive approach is the optimal *Place Finder*. This implementation should be as close as possible to naive. Looking at the error histogram 4.3b it can be seen that the mean error is higher but in general the results are similar. If the naive approach can be corrected with the second step, this method should also be corrected. This error is used as a baseline to see how the results can be improved.



(a) In blue, testing ground truth. In green, found path

(b) Translation error histogram, with the mean marked as a red line

Figure 4.3: Results using Cross Covariance *Place Finder* and no *Real Pose Finder*

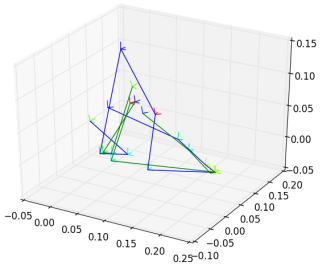
Extended Second-other Minimization *Real Pose Finder*

This is the method used in PTAM. First, it has been applied together with the naive *Place Finder* and then with cross correlation. Although the found paths in 4.4a and 4.5a are not easily evaluated visually, looking at 4.4b and 4.5b and comparing them to their corresponding baseline previously mentioned, it can be seen that in both cases the results improve.

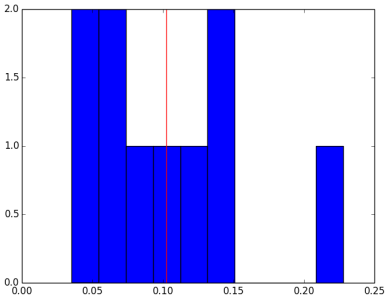
The algorithm performs better with the mentioned ideal *Place Finder*, so alternative and more accurate implementations should be considered to improve the results of this *Real Pose Finder*.

Three-point *Real Pose Finder*

As an alternative to the method proposed by PTAM, we applied the feature extraction and matching framework to find the transformation between two frames knowing the world frame location of the features. Tracking the world location of features is something that is no available on all VO implementations and it can be very useful to solve the described problem.

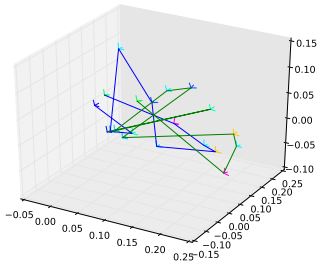


(a) In blue, testing ground truth. In green, found path

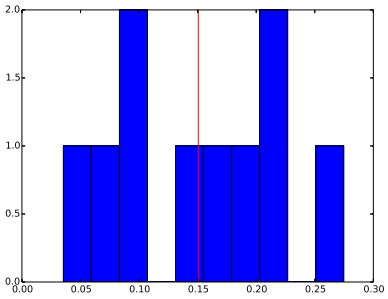


(b) Translation error histogram, with the mean marked as a red line

Figure 4.4: Results using Naive *Place Finder* and ESM *Real Pose Finder*

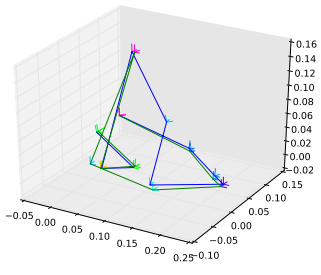


(a) In blue, testing ground truth. In green, found path

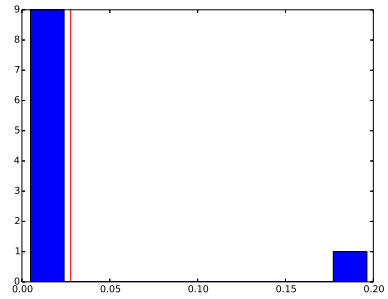


(b) Translation error histogram, with the mean marked as a red line

Figure 4.5: Results using CC *Place Finder* and ESM *Real Pose Finder*



(a) In blue, testing ground truth. In green, found path



(b) Translation error histogram, with the mean marked as a red line

Figure 4.6: Results using Naive *Place Finder* and P3P *Real Pose Finder*

In the case of the naive approach, in 4.6b, the results are very good. Using this method the pose of the frames is recovered, in most cases, perfectly yielding a very low mean error. There is one frame which was not correctly relocalized, due to the lack of enough matches found between the pair of images. As seen in 4.7 only two matches where found and, although correct, these are not enough to run the three-point algorithm. A large change in viewpoint can affect severely the descriptors performance been this the cause of the missing matches.

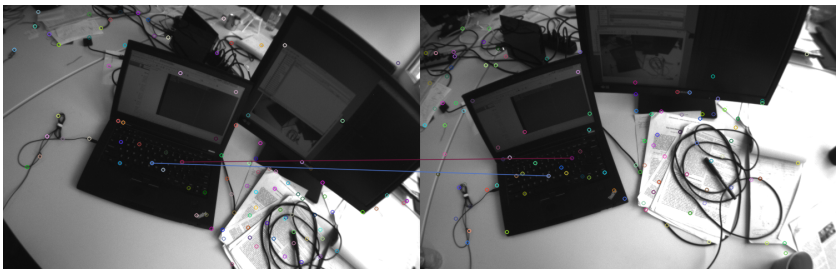
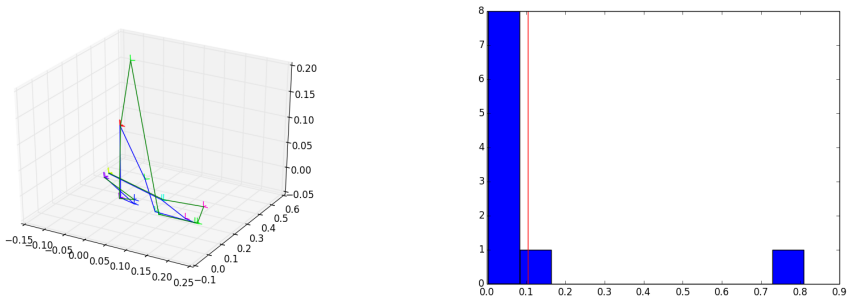


Figure 4.7: Found matches. Not enough to run the three-point algorithm

On the other case 4.14b, using CC, the results are also good. Here, there is one of the frames whose pose is not correctly recovered. In this case, as seen in 4.9, there are enough matches but during RANSAC one match from three is taken in as valid leading to a wrong solution.



(a) In blue, testing ground truth. In green, found path

(b) Translation error histogram, with the mean marked as a red line

Figure 4.8: Results using CC *Place Finder* and 3pt *Real Pose Finder*

4.1.2 Ferns

Finally, as an alternative to the PTAM relocalization method, it was proposed to use machine learning techniques, more concretely *ferns* [16]. As seen in 4.16b in this simple example it performs well having a very low mean error.

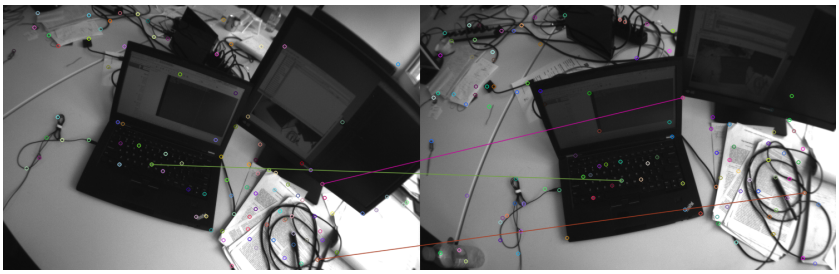
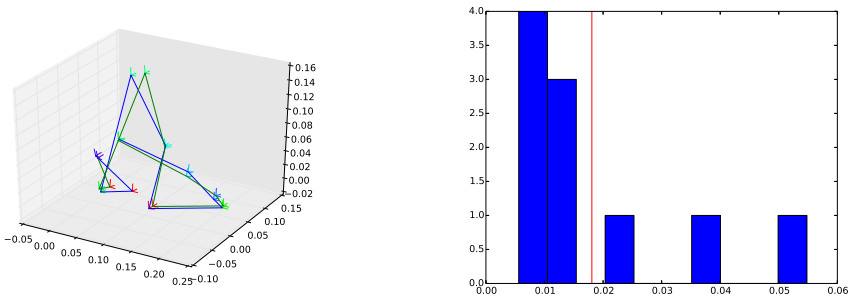


Figure 4.9: Example of accepted wrong match after RANSAC

When comparing the found path in 4.16a with the found with the Multi relocalizer with three-point it seems that it is not as precise, but in this case all frames where well relocalized. The three-point *Real Pose Finder* works very well locally yielding precise results when good matches are found, but it is very dependent on the results of the *Place Finder*. On the other hand the *ferns* relocalizer works globally. As such, it performs better in some cases but cannot easily be used when scaling to larger environments, due to the amount of classes a larger environment would contain.

In this example 237 classes are used (one per world point). It should be noticed that not all points need to be well classified, as long as more than 3 points are correctly identified then the posterior RANSAC will find and use the once that agree.



(a) In blue, testing ground truth. In green, found path

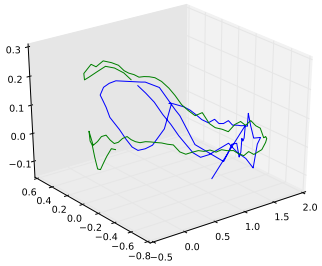
(b) Translation error histogram, with the mean marked as a red line

Figure 4.10: Results using *ferns* with 12 tests

4.2 Large Dataset

As a second experiment a larger scenario has been closed. It is also a desktop scene, but in this case, covering an area of 7x3 meters. The path taken to acquire training and testing data covering the mentioned area can be seen in 4.11a

while in Figure 4.11b there is an example of the dataset images. The dataset contains 84 frames for training and 69 for testing.



(a) In green, the path used for training. In blue, the path used for testing

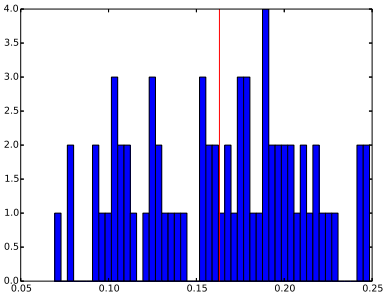
(b) Example of an image from the dataset

Figure 4.11

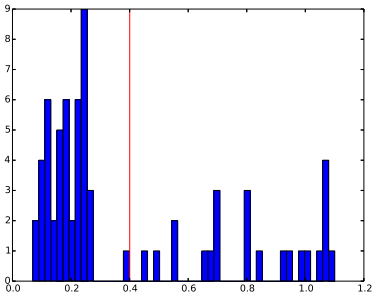
4.2.1 Multi Relocalizer

Naive and CC *Place finder*

Again, the performance of the *Place Finder* is first analysed to see then how posterior processes can improve it. In this case the Naive method gives a good baseline as was expected. On the other side, the corss correlation method produces a number of good associations that seem to agree with the naive method with errors between 0.05 and 0.25. However, it also has some wrong associations which probably cannot be corrected, such as the example illustrated in figure 4.13.



(a) Naive *Place Finder* translation error histogram



(b) CC *Place Finder* translation error histogram

Figure 4.12

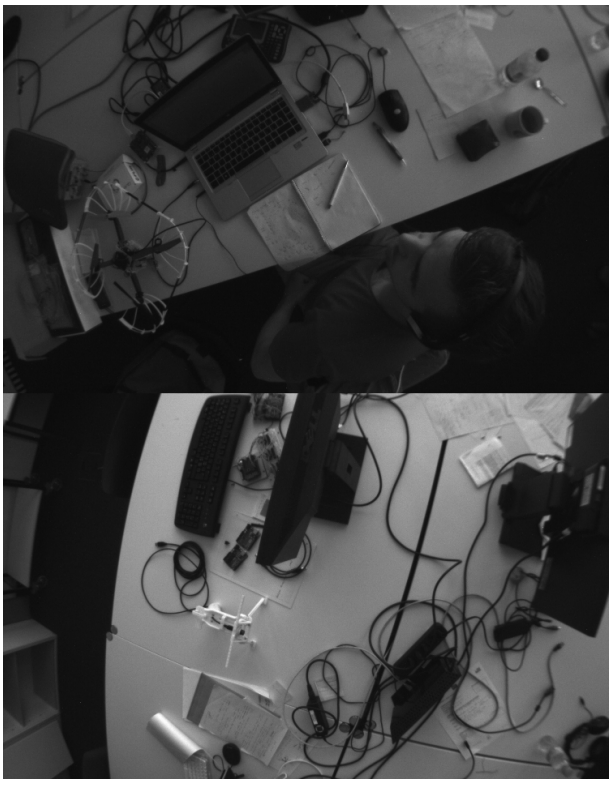
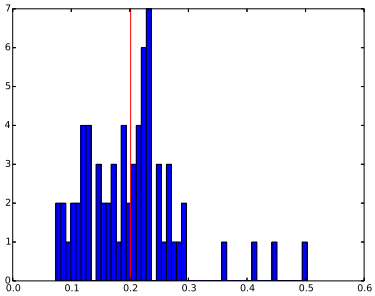


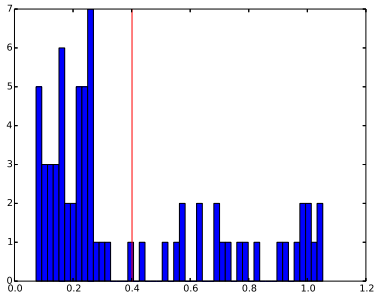
Figure 4.13: CC wrong association

Extended Second-order Minimization *Real Pose Finder*

In this case the produced path plots are not shown due to its minimal visual information contribution. Looking at the error distributions and comparing them to the performance from 4.12 it seems that together with the Naive *Place Finder* it worsen the results while with CC the results are similar, maybe slightly better. In general this method does not seem to apply well on this dataset.



(a) Translation error histogram using Naive *Place Finder* and ESM *Real Pose Finder*



(b) Translation error histogram using CC *Place Finder* and ESM *Real Pose Finder*

Figure 4.14

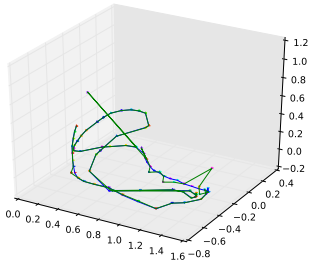
Three-point *Real Pose Finder*

On the other side, the Three-point *Real Pose Finder* seems to work very well with this dataset. Using the Naive *Place Finder* it is able to correctly find the pose of every frame. Using CC the results are also acceptably good, correctly retrieving the pose of 42 of the 69 frames. As said before, and it can be seen again in this case, this method is very dependent on the results of the *Place Finder*, and on this dataset, the cross correlation method was not very effective.

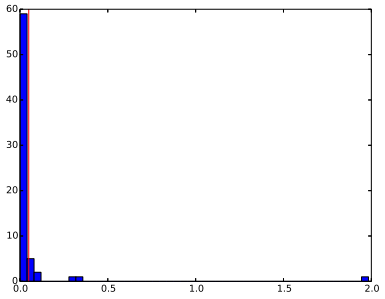
4.2.2 Ferns

In [16] it is shown that *ferns* can be used to distinguish up to around 200 different classes. We applied the method to this dataset where there are 1730 classes. In figure 4.17b it can be seen that almost 50 of the 69 frames where correctly relocalized. Which is more than using Multi Relocalizer with the three-point *Real Pose Finder*. The classifier was trained with 100 *ferns* of 12 tests each.

It is interesting to see that both methods, three-point with CC and this one, can correctly relocalize the same number of frames. Probably there is one part

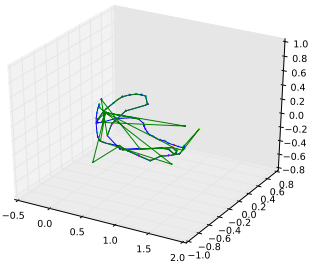


(a) In blue, testing ground truth. In green, found path

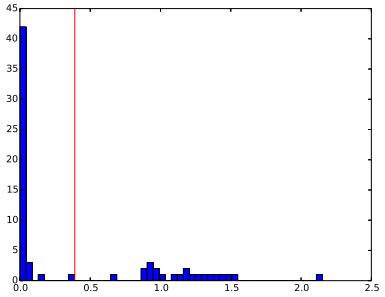


(b) Translation error histogram, with the mean marked as a red line

Figure 4.15: Results using Naive and P3P on a larger dataset



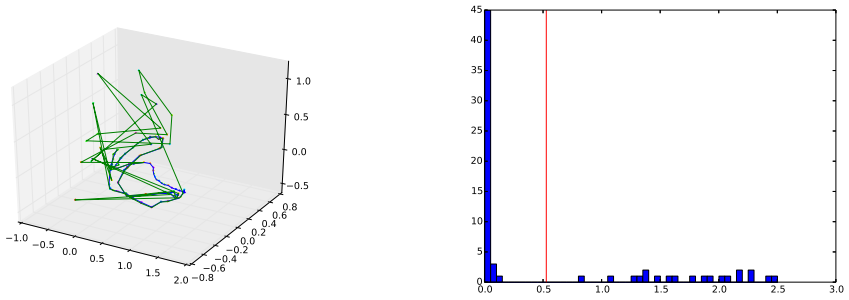
(a) In blue, testing ground truth. In green, found path



(b) Translation error histogram, with the mean marked as a red line

Figure 4.16: Results using CC and P3P on a larger dataset

of the dataset that is more ambiguous and difficult to recognize and both algorithms struggle with it.



(a) In blue, testing ground truth. In green, found path

(b) Translation error histogram, with the mean marked as a red line

Figure 4.17: Results using *ferns* with 12 tests

4.3 Execution time

In figure 4.18 the mean execution time of relocalization is shown. There, it can be seen that the ESM and the P3P methods are the faster while the method using *ferns* classifier is slower and is sensitive to the number of classes. Also, while ESM and P3P use optimized third party libraries, the *ferns* based classifier has been integrally implemented by us, maybe not achieving the best performance. The used training time for the classifier can be seen in figure 4.19.

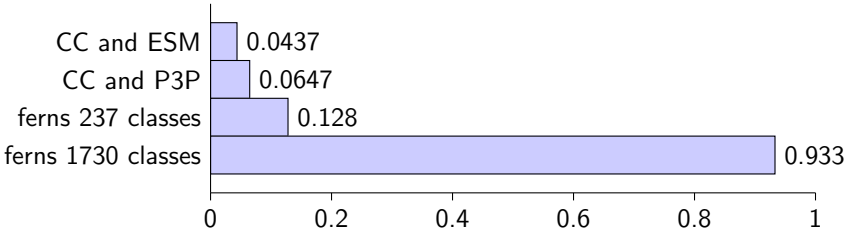


Figure 4.18: Single relocalization execution time

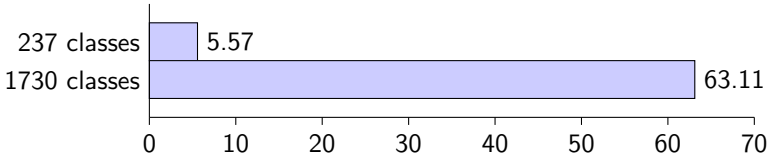


Figure 4.19: *ferns* classifier training time

Chapter 5

Discussion

This work has addressed an important part was missing in SVO, a good relocalization method which should recover the 6 DoF pose from only a map and a new frame. In this thesis different methods have been studied and implemented. First, as a starting point, the method from PTAM has been implemented which is based on image alignment. During its development the Lucas-Kanard image alignment algorithm and finally ESM has been implemented in C++ using OpenCV library.

Then, an alternative method based on the previous has been proposed. This method is based on space geometry and on the descriptor extraction and matching framework and uses the three-point algorithm to find the camera pose by relating some world points and pixel coordinates. This method produces very accurate results when matches are found between images and in general is a great improvement over the base line, being more robust and accurate.

Finally a new approach is proposed. The central idea is to use machine learning techniques to characterise the appearance of some known points in space, to later be able to retrieve their position from the pixels of an image. *Ferns* are very similar to Random Forests but simpler and easier to implement, while being able to encode the same of information. A classifier based on *ferns* have been implemented and the same time as integrated in a relocalizer.

This method has been found to give good results even in larger areas where more than 1700 classes need to be classified even though a simplified version of the training was used.

Along with this, the development of a relocalization framework has been carried on with the goal of being independent and usable by any Visual Odometry or vision based SLAM algorithm, although it has been developed to be used SVO. Furthermore, it has been developed as a ROS package in order to facilitat the integration in other robotic applications.

5.1 Future Work

There are several aspects that could be further improved to solve the proposed problem, as only a few have been studied here. Next, some are presented and discussed on what could or should be done next.

As seen in the experiments section 5 the weak point of the Multi Relocalizer at this point is the used *Place Recognition*, which is not robust to rotations and does not scale well with the number of used frames. New methods should be developed next based, for instance, on image retrieval or image classification techniques, for example using bag of words [22].

The used classifier is said to not handle many classes. One step to reduce this number could be to filter the not very characteristic classes as these might only introduce confusion to the model. An other option would be to work locally instead of globally with the whole map only keeping information from the last k frames.

In the original work on which our work have been based [16] a more intensive training is carried on. In that case more complex random affine warps are used, instead of only rotation and scale, also noise and bluriness are introduced. Trying to apply those could be some further work to be done.

Also other classifiers could be used or some point encoding which could maximize the used information. For example apply the classifier on the point's SIFT descriptor instead than on the surrounding patch.

Finally, it should be taken into account that, even though the camera is calibrated, none of the used images are rectified to save computation time (pixel positions used three-point algorithm are corrected with the camera calibration). But, the uncorrected distortion, which can be clearly seen in the example images through out the this document, can cause problems on the image alignment, on the extracted descriptors and on the patches used to train the classifier. The effect of this distortion should be tested and quantified and corrected if needed.

Appendix A

Design and Implementation

During the design and development of the library one of the goals was for it to be totally independent and self-contained. The method was planed to be used together with SVO but it had to be independent from it. For this reason none of the classes and data structures defined in SVO has been used during the implementation, only third-party libraries openly available.

This definition leaded to have some redundant definitions because in the end, this method is using a subset of the data from SVO. For example, the class SVO::Frame and the class reloc::Frame are somehow similar.

Different methods have been developed. An interface has been created to be able to use one or an other dynamically at run time. The interface is the abstract class reloc::AbstractRelocalizer, a pointer of this class is initialized to point to the real relocalizer algorithm. It can be then used then transparently, with no knowledge of which relocalizer implementation is actually being used during the rest of the execution.

The interface has been created with the following important methods:

- *addFrame*: This method is used to add a new frame used to relocalize, usually only keyframes are added to the relocalizer.
- *removeFrame*: Sometimes, a frame should not be used any more, such as, when the vehicle is sure it has moved to a different area.
- *train*: Some of the implemented methods need to train before they are able to relocalize. It should always be executed, but not all methods will do something.
- *relocalize*: This is the actual method used when the vehicle is lost. Given one new image, it will try to find its current 6 DoF pose.

There are two implementations of the interface as can be seen in A.1. *MultipleRelocalizer* implement the two problems, Place Finder and Real Pose Finder,

and connects them. Interfaces to solve those problems have been implemented too, so this class only contains pointers to its abstract interface. Different solutions can be used and are interchangeable.

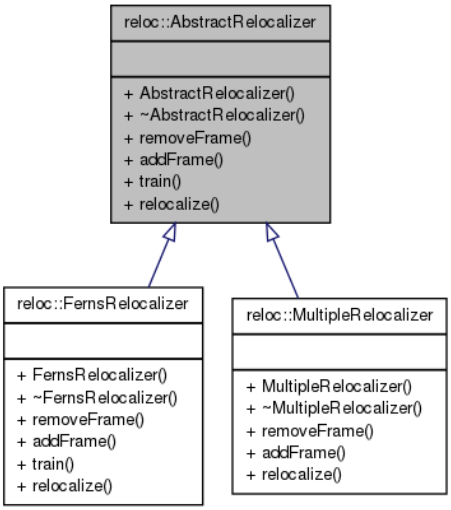


Figure A.1: Relocalizer class interface and implementations

The Place Finder interface has two important methods:

- *add/removeFrame*: The methods from the *relocalizer* with the same name pass its arguments to this methods. Here is where actual computation might be performed depending on the implementation.
- *findPlace*: This method tries to find the closes stored frame to the query image.

Only one implementation is available at the moment, using the cross correlation estimation similarly to the PTAM method. In A.2 a second implementation can be seen, it uses the actual known pose of frames. It can only be used when is information is available, which is only when testing with known ground truth. It will always find the best option available, and it is used to test different *Real Pose Finder* implementations when this first step is optimal.

Finally, there is also an interface for *Real Pose Finder*, it can be seen in Figure A.3 along with all its implementations. The methods follow the same idea as for *Place Finder* except for one method:

- *findRelps*: this method computes the transformation between two frames. The frame found by *Place Finder* and the new frame are used as input.

A class have been created to contain all the information from a frame that might be needed in the different processes, it eases the propagation of information. Also, the use of smart pointers on it eases the memory management and the data redundancy. The class reloc::Frame has the next information:

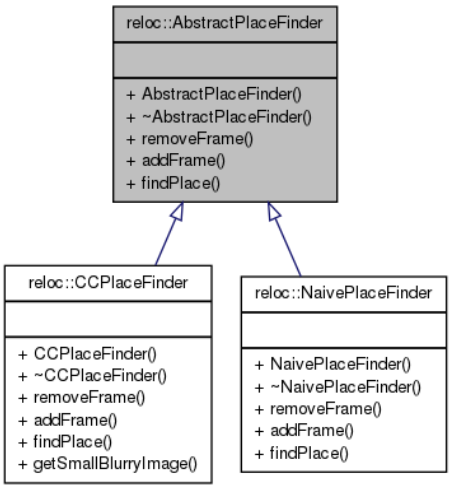


Figure A.2: Place Finder interface and implementations

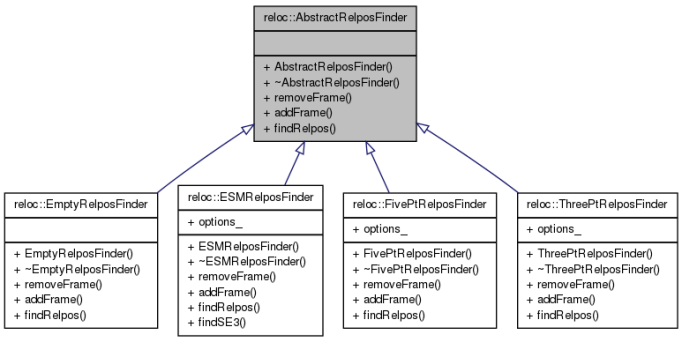


Figure A.3: Real Pose Finder interface and implementations

- **Image Pyramid**: A list of different scales of the same image.
- **Transformation World to Frame**: The $SE(3)$ transformation from the world frame to the camera frame when the image was taken.
- **Features**: A list of salient points in the image. Every feature contains:
  - The image point where it was found
  - The pyramid scale level where it was found
  - If known, the world frame point to which it corresponds

A.1 Third-party libraries

The following libraries have been used as third-party dependencies:

- **OpenCV**[4]: Mostly for its image encapsulation. Also used in image warping and descriptor extraction
- **Eigen**[7]: General matrix operations and linear algebra solutions
- **Sophus**[19]: Lie algebra groups implementation using Eigen
- **OpenGV**[12]: Three point algorithm implementation using Eigen

Bibliography

[1] Simon Baker and Iain Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. *International Journal of Computer Vision*, 56(3):221–255, February 2004.

[2] Paul R Beaudet. Rotationally invariant image operators. In *Proceedings of the International Joint Conference on Pattern Recognition*, volume 579, pages 579–583, 1978.

[3] S Benhimane and E Malis. Homography-based 2D visual servoing. *Robotics and Automation, 2006. ICRA*, 2006.

[4] G. Bradski. *Dr. Dobb's Journal of Software Tools.*

[5] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[6] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast Semi-Direct Monocular Visual Odometry. *Proc. IEEE Intl. Conf. on Robotics*, 2014.

[7] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[8] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[9] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[10] Georg Klein and David Murray. Parallel Tracking and Mapping for Small AR Workspaces. *IEEE and ACM International Symposium on Mixed and Augmented Reality*, November 2007.

[11] Georg Klein and David Murray. Improving the agility of keyframe-based SLAM. *Computer VisionECCV*, 2008.

[12] Laurent Kneip and Paul Furgale. Opengv: A unified and generalized approach to real-time calibrated geometric vision.

[13] Steven Lovegrove. *Parametric dense visual SLAM.* PhD thesis, Imperial College London, 2012.

[14] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[15] D Nister and H Stewenius. Scalable recognition with a vocabulary tree. *Vision and Pattern Recognition*, 2006.

[16] M Ozuysal and M Calonder. Fast keypoint recognition using random ferns. *Pattern Analysis*, 2010.

[17] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene Coordinate Regression Forests for Camera Relocalization in RGB-D Images. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 2930–2937, 2013.

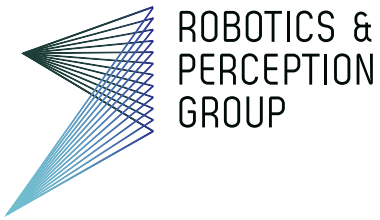
[18] Josef Sivic and Andrew Zisserman. Video Google: A text retrieval approach to object matching in videos. *Computer Vision, 2003. Proceedings.*, (Iccv):1470–1477 vol.2, 2003.

[19] Hauke Strasdat. Sophus v0.9a. c++ implementation of lie groups using eigen. https://github.com/strasdat/Sophus.

[20] Hauke Strasdat. Local Accuracy and Global Consistency for Efficient Visual SLAM. (October), 2012.

[21] Brian Williams, Georg Klein, and Ian Reid. Real-time SLAM relocalisation. *Vision, ICCV. IEEE 11th*, pages 1–8, 2007.

[22] Jun Yang, Yu-Gang Jiang, Alexander G Hauptmann, and Chong-Wah Ngo. Evaluating bag-of-visual-words representations in scene classification. In *Proceedings of the international workshop on Workshop on multimedia information retrieval*, pages 197–206. ACM, 2007.



Title of work:

Fast vision-based relocalization for MAVs

Thesis type and date:

Semester Thesis, June 2014

Supervision:

Christian Firster
Dr. Nuno Gracias
Dr. Davide Scaramuzza

Student:

Name: Quim S anchez Nicuesa
E-mail: u1068466@campus.udg.edu
Legi-Nr.:

Statement regarding plagiarism:

By signing this statement, I affirm that I have read the information notice on plagiarism, independently produced this paper, and adhered to the general practice of source citation in this subject-area.

Information notice on plagiarism:

`http://www.ethz.ch/students/semester/plagiarism_s_en.pdf`

Zurich, 2. 6. 2014: _____