Map-based localization for micro air vehicles autonomous navigation

by

Magnus Offermanns a master student of the University of Klagenfurt
maoffermanns@edu.aau.at

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by JVSRP and the National Aeronautics and Space Administration (80NM0018D004).

Abstract

Global localization of Robots in GPS denied environments is an active area of research. In this work we investigated if orbital images can be used to be matched to camera images of a Micro Aerial Vehicle (MAV) in order to find the position of a agent on a global map. Therefore we implemented a versatile feature matching frame work which can be used to conduct experiments with current state of the art feature matching algorithms. Furthermore we were able to estimate the position of a camera both in 2D and 3D space using our code. Moreover we determined experimentally how scale invariant state of the art feature matchers are. Lastly we investigated the theoretical scale difference between a camera mounted on a flying MAV and a satellite Image from a High Resolution Imaging Science Experiment (HiRise) image from the surface of Mars.

Acknowledgements

I want to thank the Austrian Marshall Plan Foundation for funding my internship which resulted in this report. I want to also thank Professor Stephan Weiss for enabling the exchange and for teaching the necessary knowledge to me to conduct research in our area. Finally I want to thank my advisor Dr. Roland Brockers and Dr. Jeff Delaune for supervising me at JPL ad giving me the guidance in which my research should head as well as handing on practical knowledge which I would not have learned from anyone else.

Contents

List of Acronyms

UAV Unmanned Aerial Vehicle

MAV Micro Aerial Vehicle

HOG Histogram of Oriented Gradients

RANSAC Random Sample Consensus

DEM Digital Elevation Model

HiRise High Resolution Imaging Science Experiment

ROS Robot Operating System

SIFT Scale-invariant Feature Transform

SURF Speeded Up Robust Features

ORB Oriented Fast and Rotated BRIEF

LoG Laplacian of Gaussian

DoG Difference of Gaussian

DoH Determinant of Hessian

FAST Features From Accelerated Segment Test

GLOH Gradient Location and Orientation Histogram

POI Point of Interest

BRIEF Binary Robust Independent Elementary Features

MPP Meters per Pixel

GPS Global Positioning System

ROI Regions of Interest

1

Introduction

Robotic localization and mapping are some of the biggest problems in Robotics. May it be on the earths surface or in space. If a robot does not know its location in reference to its next goals or a global map it does not know how to activate its motors and actors so that the location of the next mission goal can be reached and completed. Additionally it might be necessary to revisit locations that have been explored previously and might have changed over time. If the exact location of both the robot and its research target is not known this is not possible.
A 3 dimensional mapping of the surrounding is also from utter importance. If the agent does not know where the obstacles in its vicinity are it might damage itself or objects close to it. This might even lead to injuries on the human operator or other interacting humans. When a Global Positioning System (GPS) is available, like in most places on earth, global localization can be easily solved. The satellite based location system returns a clear degree of longitude and latitude which can be used for further processing and localizing. But in a lot of use cases Satellites can not reach the robot or there is just no global localization system present. Examples for this cases on earth are under water exploration, cave systems [1] or the skyscraper canyons of mayor cities. In space there is no GPS, hence it is necessary for every robot on the Moon, Mars or a different place in the solar system to use another system to orientate itself. In the last cases mentioned the robot is also isolated from its operators. That means no physical robot operator interaction can take place. This increases the consequence of a operation failure even more, since the robot can not be recovered. In most cases this means that the mission is over. This kind of undertakings are called remote sensing missions. Targets need to be visited, observed and later on reliably found to notice changes over time. All of this has to play out while the robots operation is faultless to prevent a mission failure. An example for such a mission would be a temporal observation of a significant rock on Mars.

The research we conducted focuses on this use case. A global localization of a agent in a GPS denied environment. Previous research mainly focused on localization of Rovers on Mars using its on-board cameras and information obtained by orbiting satellites. Possible solutions were horizon matching [2] or the matching of distinct stones or rock formations [3]. Problems of this kind of approaches are the strong difference of point of view between the satellite image and the agent on Mars. Therefore images need to be orthorectified [4] i.e. warped from the ground perspective in to the bird's eye view which introduces errors inside the processed

ground images. Additionally there is a big difference between the distance of the camera of the rover and the camera of the satellite, this so called scale difference lets common matching technologies fail.

In the last years NASAs research did not just focus on rovers on Mars but also on MAVs since the Mars 2020 mission includes a helicopter which has a downward facing camera which is also used for navigational tasks. While the big scale change needs to be compensated to find similarities between a orbital image and a MAV/Helicopter image, the perspective of both the satellite and the flying robot are similar.
In this work we intended to recreate the problem of matching a Drone image to a orbital satellite image. Instead of a Marsian map image we use a common Google maps image which should be matched with a life camera feed here on earth. But this introduces another challenge. Since the image of the satellite and the image of the drone are taken at different times the appearance of the observed places changes. While the changes on Mars are not as rapid, the visual nature of objects on earth varies strongly between the seasons. It should have been tested how appearance and lighting changes worsen the estimation.
The implemented result is a drone frame work which is able to match previously taken map images, with a low resolution, with a camera feed of images from a lower flying camera with a higher resolution. When a matching occurs the pose is estimated and displayed in the map image. Furthermore it is very easy to change the code to switch between common matching algorithms for localization. Additionally we tested the scale and rotation invariance of the common matching algorithms SIFT [5],SURF [6], upright SURF and ORB [7]. All experiments were conducted in a laboratory. Finally we investigated how high a camera must hover over the earth to have a comparable resolution to a orbital satellite imagery.
Goals not reached are the matching of a map image and a camera feed in an outdoor environment as well as the matching of a camera feed with Google maps images. Furthermore the influence of weather and seasons on camera/map matching was not investigated.

2

Related Work

Feature matching has been an important research topic in the computer vision community for years [8] . There are various use cases like the recovery of similar images from a database [9] as used in the Google image search, the detection of objects in a life feed e.g. the counting of humans on a surveillance camera [10] or to detect visual input from a user based on a camera feed [11]. In order to detect image snippets between images there are two main approaches which were previously used to solve comparable problems. Correlation based matching where a smaller image snippet is correlated with subsections of a bigger image in which we assume the smaller image is a part of. Another way to identify the similarity of images is called *feature based matching*. In which distinct pixel groups in the two or more images that shall be matched are identified, described and then compared in order to pair pixel groups with akin appearance. If we find enough similarities between both images we can be certain that we look at the same or a comparable image.

Matching techniques vary in their properties. First of all their run time, the ability to compensate for different scales, rotations and point of view changes. Correlation based methods are disadvantageous concerning run time because of its many multiplications. Additionally is the bare correlation very little invariant to scale or rotation. Feature based matching algorithms vary in the same characteristics. For example is the SIFT [5] algorithm the state of the art concerning scale and rotation invariance but can not be run in real time. The SURF [6] algorithm is based on SIFT but decreases the run time with a slight loss of matching ability. Other common feature matching technologies are ORB [7] or the not rotation invariant upright SURF.

The problem of matching an image from a low resolution map image to a camera of a drone has already been attempted with different approaches. Shan [12] has tried to estimate the position of a flying agent using a particle filter and by describing the image and the map using Histogram of Oriented Gradients (HOG) features followed by a image correlation. The preprocessing comes up for the scale and rotation invariance. Problems of this solution are that an extensive preprocessing of the map is necessary. Additionally it is not very robust to lighting changes. Another approach by Conte et al. [13] also uses correlation but prepossesses the raw images of the Unmanned Aerial Vehicle (UAV) with a Sobel edge detector. But preprocessing is still necessary and the run time of 1hz is to slow to run with common camera

image frequencies.
There has been very few approaches using feature based matching. One of them is the paper by Tao [4] which uses SIFT features followed by a RANSAC outlier detection. Advantages are that no preprocessing is necessary. One disadvantage though is that the quality of solution depends highly on the extractor/descriptor/matcher. An unorthodox approach towards global localization takes the paper of Lindsten [14]. To identify a snippet the map is segmented into superpixels. For each superpixel, a histogram of the central circular region is calculated. The same is done for the current UAV. Advantageous is that the histogram of round regions is rotation and scale invariant. Using a custom statistical method, the position of the drone is calculated. Disadvantages of this method are the necessary pre-processing for the segmentation of the map in super pixels.

After looking at the publications about global localization, we will look at strategies to locate ground bound robots on the surface of Mars since these matching techniques are designed for the same environment than NASA MAVs. The first group of approaches is observing distinct rocks and is trying to match them to HiRise images. In the work of Hwangbo [15] he extracts a 3d pointcloud out of stereo images from a rover. Then rocks inside the HiRise image are extracted using pixel grouping. After a RANSAC refinement the extracted rocks are matched and a position is estimated. A similar approach is also taken by Li [3].
A second group of algorithms for global localization intents to generate orthorectified images from the two cameras of the rover. Now that both the HiRise image as well as the rover image have the same perspective the images are processed for the easier identification of Regions of Interest (ROI). Afterwards a feature extractor/descriptor/matcher identifies the position of the agent. Notable examples for this approaches are the papers with the citations [16] and [17].
One of the most distinctive features on Mars is the horizon. HiRise is also equipped to calculate a height profile of Mars. This information can be used to calculate the horizon line for any position the rover could stand at. Since the horizon line of the actual rover is known, the correlation between all simulated horizons and the actual horizon can be used to estimate the position of the rover. Notable examples of this approach are the papers of Nefian [18] and Chiodini [2].
For a full overview over this topic I can recommend the paper of Boukas [19].

An application that is also very similar to locating a drone on Mars using camera images is the estimation of the position of a lander while approaching a moon or a planet. Both the drone and the lander are equipped with cameras and fly over the surface. There are approaches to use satellite images to reduce the uncertainty of the landing position of the lander. I.e. the problem statement between a flying drone and a landing spacecraft are comparable.
The paper of Delaune [20] is using the known Digital Elevation Model (DEM) of the moon as map to match it to a in flight generated DEM from a landing spacecraft. An advantage of this approach is that the feature descriptor not only contains image characteristics but also carries information of its elevation. Using a Harris-Laplace feature extractor and descriptor followed by a custom matcher and a RANSAC refinement it is possible to estimate the current position of the lander. Another approach is taken by the Mourikis [21] and Chris [22]. The position estimation is

split into two parts. The first step is the estimation of the pose of the lander in comparison to the orientation of the map. In this part the absolute position of the lander is not important, as it is just necessary to know the orientation displacement between the lander image and the map. This prediction is only using the features of a high resolution camera mounted on the craft. This features are referred to as opportunistic features. The second class of features is called mapped landmarks and can be seen on both images. Since only very little features of the lander images qualify as mapped features and can therefore be identified in the satellite image, it is of very high importance that the features are matched correctly. Since the pose of the lander and the satellite image are not equal, the feature descriptors would need to compensate for a strong scale and rotation difference. To reduce this discrepancy the estimated pose based on the opportunistic features is used to warp the image into the right resolution and orientation. Afterwards a feature extraction/description/matching takes place and a homography is estimated. This matrix can now be used to position the lander in the global map.

3

Design

In this section we will describe the design considerations that went into the project. We will first write about technology used followed by a general description of feature matchers. The next part of the text is describing the steps taken to validate and compare the different matchers and its invariance towards scale and orientation. Followed by a introduction into homographies as well as the theory to recover the pose of a MAV out of it.

3.1 ROS - The Robot Operating System

The ROS [23] is a open source programming framework which makes it easier to develop software for robotic systems and incorporate hardware from different distributors. It is possible to program ROS in various programming languages like Python and LISP. In this project C++ is prevalent.

3.1.1 Structure of ROS

ROS is set up as a Graph structure in which individual processes called nodes represent the vertices. These offer topics or services other nodes can subscribe to. Over this advertiser/subscriber relationship between nodes, information can be exchanged. Topics and services are like edges known from graph theory.

The ROScore

Before all other nodes are started a service called ROScore needs to be initialized. It works as central coordinator to organize the different nodes which register their topics and services after startup. If new nodes want to subscribe to certain topics or services the ROScore connects the provider and the recipient. A great advantage of the position as a linker is that during runtime no data is tunneled through the ROScore and there is no middleman between two connected nodes.

ROS Nodes

The individual entities of the distributed system that ROS is, are called nodes. Every node is an individual process and is independent of other nodes. As long as the ROScore is running individual nodes can fail and all the other parts of the system can go on working undisturbed. Nodes are also not unique, multiple instances of

the same node can be started at the same time. While nodes have a association to a package and a clear executable they are distinguished by given names at start up. This makes it possible that nodes of the same kind can work at the same time, increasing re-usability of code. This is is used in cases where an agent has multiple sensors of the same type (i.e. pressure sensors at multiple locations on the chassis) where the same node with a different name can be used to access the sensor readings of different sensors of the same kind.

3.1.2 ROS Topics & Services

The nodes described in section 3.1.1 are able to communicate with each other based on a publisher and subscriber system. If information or data should be provided for other nodes we can publish a topic. Then a message, which is described in section 3.1.4, is sent out in a certain frequency for all other nodes to receive. A node can collect this public messages by subscribing to a topic. A topic is a unique identifier of the data stream. If a node is subscribed to a topic, it can call a function every time a message is released and is thereby a so called subscriber. Topics can be organized by certain naming rules. Topics can be a global- ,private- relative- or base-topics. This feature can be used to access just global topics, topics of a certain name space which are used to organize certain nodes or private topics that are topics with a clear association to the name of a node. A description of the naming conventions can be found on the official ROS website [24].

Another form of information exchange are the so called services. Services publish in contrary to topics not in a regular frequency but on demand. That means that a subscriber can request the needed information when information is necessary for further computation. This is done by a message followed up by a response message by the service provider. An advantage is that services reduce data overhead i.e the messages are published even if no other node requires the information. A down side of services is though that they make synchronization more complex.

3.1.3 The Parameterserver & Launchfiles

The parameterserver is a service provided by the roscore, described in section 3.1.1, and contains hyperparameters which should be accessible by all nodes. Since the server is not made for complex data it is best used for configuration data and simple data to be shared between all running nodes. The parameters are organized by the same naming conventions as ROS topics. It is described in section 3.1.2.
In this project we shared the camera matrices of the used cameras using the parameterserver to make it available to all nodes processing camera images. We made the decision of sharing this data because it full-filled all prerequisites, having a simple structure of 9 integers and being used by multiple nodes.

The launchfile is a *.XML* file which can be interpreted by the ROS tool Roslaunch and can be used to setup a complex graph of ROS nodes. The first use of the *.launch* file is the setting of global parameters which are stored in the parameter server and are accessible for all nodes. The second use is the start-up of nodes. Multiple nodes can be initialized using one command and can be temporarily ordered. That means

if nodes need to be started in a certain order because of dependencies this is possible. Additionally nodes can be described as mandatory i.e. if a mandatory node fails all other nodes shut down as well. Another feature that can be done is the remapping of topics. Topic names can be renamed to connect nodes without changing the source code. Lastly private parameters of individual nodes can be set. This is also use-full to set hyper-parameters without changing the source code.

3.1.4 Messages

In order to let nodes communicate with each other messages are used. Messages are standardized data packages defined in the *.msg* files in the msg subdirectory of a package. Additionally ROS provides already a diverse set of useful commonly used messages. Messages can carry the information of basic datatypes as ints, floats and arrays of these but can also comprise messages of other types. A good example for the first kind is the *geometry_msgs/Vector3* Message which contains three doubles and can be used for positions or rotations. A group of messages that is regularly used and are made out of other common messages are the stamped messages. If a name of message contains the word stamped it means that additionally to the normal data, a header i.e. information of the timing and the sequence number, is incorporated. This happens by adding the *std_msgs/Header* Message to the *geometry_msgs/Vector3* message resulting in the creation of the *geometry_msgs/Vector3Stamped* message. This enables the versatile creation of easy fast to deploy messages. In this project we used the following basic messages:

- *geometry_msgs/Vector3*
- *geometry_msgs/Quaternion*
- *geometry_msgs/PoseStamped*
- *geometry_msgs/TransformStamped*
- *sensor_msgs/CameraInfo*
- *sensor_msgs/Image*

For a description of the messages created for this project alone refer to section 4.3.1.

3.2 Extractors, Descriptors, Matchers

In this section we will describe the parts all feature matching algorithms consist of. First the feature extraction (3.2.1), followed by the feature description (3.2.1) and closing the matching of image segments with comparable appearances (3.2.1). Subsequently we will address the challenges all matching algorithms face (3.2.1).Closing we will describe the matching algorithms we have been using in this work i.e. SIFT (3.2.2), SURF (3.2.3), upright SURF (3.2.3) and ORB (3.2.4).

3.2.1 Feature Matchers in General

Every feature matching algorithm consists out of three parts. First the extraction of interesting segments of the image. Followed by a description of the feature. That typically means collecting the previously calculated information and saving it into a vector of numbers followed by a matching. This is a distance calculation between the descriptive vectors of all features. Then the possible feature matches are first thresholded using their respective error and then the two or more features that have

the least error are considered as match.

Feature extraction

Since we are not able to match a unprocessed image with another image we want to reduce the complexity of our data. Therefore segments of the image with the most information should be chosen. Additionally the segments should not be redundant i.e. they should not be very similar to other parts of the image. The most commonly used features are edges, corners and other sudden brightness changes called blobs. Therefore we distinguish extractors by the kind of feature they detect. Edge detectors are detectors which only identify edges e.g. the Canny edge detector [25] or the Sobel edge detector [26]. Common Corner detectors are the Harris corner detector [27] and Features From Accelerated Segment Test (FAST) [28] which can serve besides having a low runtime also as corner and blob detector. Blob detectors are very often filters. For example the Laplacian of Gaussian (LoG) in which the image is first filtered using a Gaussian filter followed by a application of the Laplacian operator (LoG). Similar algorithms are the Difference of Gaussian (DoG) or the Determinant of Hessian (DoH).

Feature descriptors

Feature descriptors try to characterize the beforehand found feature and a small area around it as distinctive as possible. Additionally the descriptor should at best not change if the feature is in different scale, illumination or orientation on the image.

Therefore the pixel values of a neighborhood around the feature are taken into consideration. Histograms, which are rotation invariant, weighted brightnesses, where pixels in the center of the feature are weighted more, as well as the orientation of the feature (normally the direction in which the darkest parts of the neighborhood points at) are inserted into the vector. In general after the collection of information various normalization's take place to make the descriptor invariant to scale or illumination changes. All of this data is then filled into the vector for later comparison with other features. Common feature matching algorithms are HOG and Gradient Location and Orientation Histogram (GLOH). While whole feature matchers often also have their unique descriptors they do not have a specific name i.e. SIFT-descriptor or SURF-descriptor.

Feature matching

Feature matching is the last and least computational intensive task. To match two features the error or distance from all other features is calculated. Commonly norms like the Manhattan distance or the Euclidean distance are used to determine similarity. Then the distances are thresholded to eliminate matching of highly unequal features. The two or more features that have the most akin descriptor are than considered a match.

Challenges of Feature matchers

Feature matching is a complex topic even humans struggle with. For example are humans very often not able to recognize if they have already been in a street of a mayor city they have been visiting for holidays. One of the causes can be that streets of a city all look very similar. This can also occur when trying to identify objects using feature matching. Reoccurring patterns in images make the clear identification harder, also self-similar object might make it hard to identify scale. Another problem might be that one visits the street at a another time of the day. Illumination changes also need to be solved when feature matching. But when using cameras, illumination changes can not only occur form external influences but also if the settings of the cameras (e.g exposure) change. A matcher that can cope with these changes is called *illumination-invariant*. Another problem might be that the scene we observe changes over time. Temporal-invariance is still a major topic of research. Rotation of objects is also problematic. If the camera or the object we want to detect rotates it might not be possible to be recognized. Therefore a matching algorithm need to be *rotation-invariant* i.e. the object should be detected independent of its rotation. The last mayor problem that makes it hard to detect features is that objects look different depending on the distance of the camera because the resolution changes. In this work we try to match images that have highly different scales. That makes it clear that *scale-invariant* feature detectors are still an open field of research and need to be improved in future.

3.2.2 SIFT

SIFT is a full feature matching pipeline from extraction till outlier detection and is well described in Lowes paper from 2004 [5]. As previously mentioned SIFT is very scale- and rotation-invariant but has drawbacks concerning runtime. In this section we will only describe the first three steps of the algorithm i.e. from extraction till matching. We did also use the SIFT outlier detection for SURF and ORB therefore SIFT or LOWE outlier detection will be described in section 3.3 together with other outlier detection mechanisms as RANSAC.

Feature Extraction SIFT

The SIFT feature extraction uses multiple basic principles to identify the best features, their location and rotation. First a blob detection is used to identify irregularities in the image. LoG is a good choice but its long calculation time would increase the runtime of SIFT unnecessarily. Therefore Lowe decided to use DoG which is an approximation of the aforementioned filter and is faster. The DoG is calculated by applying Gaussian kernels with different σ to the image. Every two images with the closest σ are then subtracted resulting in $n-1$ images while n is the number of Gaussian kernels applied. Ensuing local extremas in this $n-1$ images are Point of Interest (POI). If the extrema shows up in multiple consecutive DoG images we can assume that we have found a local extrema at this location. But if this calculation is only applied in one scale we are not able to determine if there would be features at more places in the image if the camera is closer or further away from the scene. Therefore we warp our starting image in multiple image resolutions in order to not

only find features in one scale but in multiple scales. This enables us to recognize features even if the camera that took the photo is closer or further away from the scene and makes SIFT highly scale-invariant. Every individual scale we warp our starting image into is called "octave".

After the last step we have a feature location in space and in scale which accuracy is bounded by the sample frequency of the pixel and the distance between the octaves. We can only assume that the detected feature is exactly at the sample point of the pixel and exactly at the scale we are currently observing i.e. we do not have sub pixel accuracy and have an error depending on how we choose the scale steps. Therefore interpolation between adjacent pixels and scale levels is performed. Additionally this mechanism can be used to filter the POI based on stability. The distance of the sample point from the interpolated value is used as a measure of feature quality and in [5] a value of 0.03 is used to further reduce the number of detected features.

Another challenge we encounter because the of the DoG is that it has a strong response at edges even if the edge does not have a high contrast or there is a lot of noise in this part of the image. Therefore Lowe needed to find a way to eliminate bad edge responses. He used a Hessian matrix to calculate the principal curvature at the place of the disputed keypoint. The sample points close to the edges are used to calculate the necessary derivatives for the Hessian. Badly defined edges will have a high principal curvature along one edge while having a small one in perpendicular direction. Therefore Lowe used the indirect method proposed by Harris and Stephens [27] to calculate the ratio between the two eigenvalues of the Hessian. Lowe proposes that this ratio needs to be below 10 do deliver good results.

Rotation assignment SIFT

After the last calculation the keypoints we receive are highly accurate and scale invariant. One problem we still encounter is that the keypoints do not get detected if the camera is rotated. Therefore the next step will make our POI rotation invariant.

In order to find a orientation we look at pixels in the proximity of our feature. Depending on the scale of the feature we organize them in 36 bins where each bin is assigned to a certain direction to cover the full 0 to 360 degrees. Now the gradient direction as well as amplitude is calculated for each pixel in the area around the feature. This amount is added into one of the 36 bins waited by the amplitude of the gradient. After filling the bins, the most full bin is regarded as 100 percent. The ratio of the other bins in comparison to the fullest bin is also calculated. To now create a orientation to the POI, the strongest orientation is assigned to the feature. If another bin also has a amplitude of 80 percent in comparison to the main orientation, one more feature for each orientation bin which is above that threshold is created. These features have the same information about the pixel location and scale as the main feature. The only difference is that the orientation changes. By creating multiple keypoints with similar properties the matching is more robust, because the information of all strong orientations is kept.

Feature description SIFT

After now identifying the location, scale and rotation of the keypoint a descriptor needs to be calculated which can be compared to other keypoint descriptors. The length of the SIFT feature detector is 128, if an extended feature vector is used the length of the vector is 256.
In order to find values that are descriptive to the feature we look at a 16x16 window around the keypoint in the optimal scale that we previously determined. This window is weighted by a Gaussian function in order to weight the pixel value by the distance from the keypoint. For each 4x4 block a histogram of orientated gradients with 8 bins is calculated. That means that each pixel has a direction and amplitude. Afterwards they are ordered in 8 bins where each bin represents a certain range of degrees i.e. the fist bin is 0-45 degrees the second bin 46-90 degrees and so on. This procedure is very similar to the rotation assignment of section 3.2.2. The resulting vector now has 16 times 8 bin values representing the feature. But we have not yet used the rotation that we calculated previously. In order to make the feature vector rotation invariant we rotate all bins by the rotation that we previously determined. By doing so we eliminate the rotation of the feature because if we detect the same orientation of the same feature displayed in different orientation we are able to rotate both in the same rotation. In the last step the feature vector should be made illumination invariant. Therefore we normalize the vector values to make it resistant to brightness changes where a constant is added to all pixel values. Since the values are calculated by a gradient they are not influenced by constant changes after normalization. The second brightness changes that can happen are non-linear by camera saturation or reflections on 3d surfaces. To still keep the descriptor ilumination invariant we filter high values and set a maximum value that a pixel can take. Lowe [5] determined experimentally that a maximum value of 0.2 after normalization is delivering the best results.

The result after all calculation steps is the state of the art of feature matchers without time constraints in 2019. Unfortunately the long calculation time prevents SIFT from being used in real time applications. Inspired by this constraints Bay et. al. [6] came up with SURF which is a slightly less robust but much faster version of SIFT which is described in the following section.

3.2.3 SURF and upright SURF

The SURF detector is a faster version of the SIFT matcher. It uses multiple ways to speed up the extraction/description/matching. First of all it uses approximations of the hessian matrix both to find interest points and their optimal scale (described in section 3.2.3). Further more, since the filters used to find features are box filters and require sums over squares of the image an integral image is used to speed up calculation. Lastly the Haar-wavelet response is used to calculate the orientation and the feature vector resulting in a vector of the length of 64 also resulting in quicker calculation time. This is described in section 3.2.3 and 3.2.3.

Feature Extraction SURF

SIFT uses a approximation of the LoG in scale space in order to identify points of interest followed by a Hessian matrix to filter non use full edges. The SURF descriptor uses a approximated hessian matrix for both. The algorithm uses box filters in order to calculate the hessian of each pixel. The determinant of the hessian is then used to filter for good or bad features.

In order to also check in which scale the feature is, the the whole image is not scaled but we re-size the box filters suitable for the scale we want to observe. This has the same effect as a resolution change because the box filter serves both as Gaussian filter which blurs the image like a resolution reduction and can be used to calculate the second derivative. In order to detect the optimum scale a neighborhood of 9 pixels in the scale above and below the feature is observed. The best feature in this three layers is than taken as best scale.

Integral Images and Speeding up the Feature Extraction Process

Since the box filters are calculated by summing square regions of the image it is convenient if the summations do not need to be repeated every time the filter is applied. Therefore a integral image is precalculated as first step of the algorithm. A pixel of the integral image is calculated by summing up all previous pixel in x and y axis and the current pixel as can be seen in formula 3.1. The calculated image can be used to calculate sums of squares in the original image with 4 additions. This makes the application of boxfilters high-performance.

$$I_{\sum}(I(x,y)) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i,j) \tag{3.1}$$

Calculation of Feature Orientations in SURF

In contrary to the SIFT orientation assignment by using the gradient of pixels SURF uses Haar-wavelet responses. In a round environment which size is dependent on the previously determined scale we calculate the Haar-wavelet response in x and y axis. Since this is also a filter operation based on window additions we can reuse the calculated Integral image described in the section above (3.2.3). After locating the wavelet responses we count the responses in a rotating window covering $\frac{\pi}{3}$ and give the vector representing the window a amplitude depending on this quantity. The region with the highest amplitude is regarded as the feature orientation. In the special case that we know that the image is not rotated this step can be skipped. The resulting algorithm is than called upright surf and is only working if the camera was not spun. It is called Upright SURF.

Generating the SURF descriptor

After we identified the scale and orientation of the feature we now create the descriptor which is needed for comparison to other features. Therefore we generate a square region which is orthogonal to the direction calculated in section 3.2.3. This regions size depends of the scale of the feature and is split up into 16 square regions.

We calculate the Haar-response in x and y at 25 evenly spaced locations in each of the 16 regions. Than we sum the wavelet response in x as well as in y and use them as features. If we do this for 16 regions we get 32 features. Additionally we calculate the absolute value of the responses in x and y and use it as features as well. The Added up we get a full feature vector of 64 values which is half the size of conventional SIFT.

Summarizing it can be said that SURF improves the runtime by smart use of block filters and integral images. Additionally Haar-wavelets can also be calculated with integral images improving the speed even further.

3.2.4 The ORB Feature Matcher

SURF and SIFT are highly performing matching algorithms. Unfortunately both of them are patented and not free to use. For this gap ORB was developed [7]. By using FAST features followed by a rotation invariant binary feature descriptor, the matcher performs almost as well as the state of the art but has a much faster runtime.

The ORB Feature Extractor

FAST is a very fast feature extractor. It looks at a circle consisting of 16 pixels around each pixel. Now each pixel is compared to the center sample point of the circle. If 8 consecutive pixels are brighter or darker than the center it is flagged as a feature.

Unfortunately FAST without an additional mechanism is not scale-invariant. ORB therefore uses an image pyramid which scales the image to a higher and lower resolution. Then FAST is applied again to the all images including the scaled one. Then the scale in which the feature is found gets assigned to the POI.

Orientation Assignment with FAST

The original FAST invented by Rosten and Drummond [28] did not have a orientation assignment to the features. Therefore Rublee et al. [7] came up with the idea to use the center of mass to assign a orientation to the features. Therefore the first moments in x and y axis are calculated using the formula 3.2. Now a vector between the Centroid (center of mass) and the actual center is calculated. Using the arcus tangens in formula 3.4 and the components of the before calculated vector a angle can be calculated. This angle (θ) is then assigned to the feature. When now comparing, all features are rotated to the same direction.

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{3.2}$$

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}) \tag{3.3}$$

$$\theta = atan2(m_{01}, m_{10}) \tag{3.4}$$

Feature Description with Binary Robust Independent Elementary Features (BRIEF)

In contrary to the previous descriptors used in SIFT 3.2.2 and SURF 3.2.3 where the descriptor consists of integers and doubles the BRIEF descriptor is a binary descriptor. Meaning that it only consist out of Booleans. The information is extracted from a round area around the feature. Because binary features are prone to noise the image region is down scaled with a Gaussian filter. Before information is collected the feature is rotated by the orientation calculated in section 3.2.4 in order to align all extracted features.

In order to extract the booleans, a beforehand defined pattern of pixels is chosen. This pixels are then listed successively in a vector starting from the first sample point and ending at the last sample point. Subsequently depending on the intensity of the previous pixel, the Booleans in the feature vector are set to *true* or *false*. If the intensity value of the previous pixel is smaller than the current one the value is set to one. In all other cases the value is set to zero. The function to build the vector is taken from the paper of Rublee et al. [7] and can also be found below in formula 3.5. Where $p(x)$ denotes the current pixel and $p(y)$ denotes the previouse pixel. In formula 3.6, which is also taken from Rublee et al. [7] the building of the vector is described where $f(n)$ is the n_{th} entry in the feature vector.

$$\tau(p; x, y) = \begin{cases} 1 & : p(x) < p(y) \\ 0 & : p(x) \geq p(y) \end{cases} \tag{3.5}$$

$$f(n) = \sum_{1<i<n} 2^{i-1} \tau(p; x_i, y_i) \tag{3.6}$$

3.3 Outlierdetection, RANSAC

After the matching of features, the quantity of features is very high and the quality of matches can still be improved on. Therefore we used algorithms to filter bad matches and keep good matches for further use in the homography estimation. Therefore we first use the LOWE ratio test to keep 75 percent of the best features followed by RANSAC applied to a flat earth assumption which filters features that do not lie on a flat surface.

3.3.1 Lowe Ratio Test

The Lowe ratio test is used to discard ambiguous features. Therefore it compares the distance of the two nearest matches to the feature and if the two matches are closer than a certain threshold they are discarded. It was first proposed in Lowes paper in which he also described SIFT [5]. A good threshold stated and determined by experiments is 0.75.

3.3.2 Filtering Using RANSAC and the Flat Earth Assumption

After identifying good matches using the Lowe ratio test described in section 3.3.1 we use the RANSAC algorithm in order to identify the best fitting homography between the two images we want to match. A homography is a 3×3 matrix mapping pixels between two different images with the same optical center. That means if we know the pixel position of pixel p in the first camera frame and the homography we can calculate this pixel position in the second camera (p') frame using formula 3.7.

$$p' = H \cdot p \tag{3.7}$$

It is possible to use the homography to project between the pixels because we assume that our features lie on a plane in 3D space. In the laboratory this is the case because the features we track are markers on a flat ground. In the real world we assume that the drone with the mounted camera flies in a height at which the elevation changes in comparison to the distance of the camera to the ground are negligible.

We now use RANSAC and our previously found matches to estimate the best homography with the most inliers possible. RANSAC is a iterative algorithm that has six steps. In the first step it chooses four random feature pairs in order to calculate the exact homography for example with the 8 point algorithm [29]. In the second step we project all features and calculate the error between the corresponding projected feature and the associated matched feature. In the third step we threshold the matches by an experimentally determined hyperparameter using the resulting error of each feature in order to divide them in inliers and outliers. This three steps are repeated n times. In the fourth step the 4 matches with the most inliers and/or the lowest reprojection error are determined. Subsequently we use all inliers of this model to re-calculate a homography not just with 8 features but with a error minimizing homography estimating optimization algorithm. With the now estimated homography it is possible to project pixels from one image frame into the other image frame. Additionally it is possible to decompose the estimated homography in order to determine the position of the camera in 3d space. This is done in section 3.4.

3.4 Position Estimation using Homography Decomposition

Once a homography is estimated pixels can be reprojected from one camera frame into the other. This establishes 2D to 2D assosciations but does not give us any information on the actual 3d positions of the features and the cameras that took the images. Therefore it is necessary to extract the position of the cameras. Unfortunately the homography matrix has 8 degrees of freedom and is accurate to scale. That means that we get 8 solutions before the filtering of the solutions and 4 solutions that can not be mathematically distinguished. Therefore algorithms to distinguish the last four solutions needed to be implemented.

3.4.1 Results of the Homography Decomposition

The homography decomposition returns four sets of solutions. Each solution consist of 2 vectors and one matrix. The translation vector t that is the translation from the first camera to the second. In the same manner it also returns the rotation matrix R from the camera one to camera two. The last returned vector the decomposition returns is the normal vector of the plane n on which the features are positioned in 3d space. With this three results it is possible to estimate the position with a degree of freedom in distance from the plane of the two cameras in 3D space. This is illustrated by figure 3.1 and which was created by Malis and Vargas. [30]. As can be seen in the figure π denotes the plane on which the 3D features P_i lie. F and F' denote the centers of the two cameras. m and m' are the respective image planes. Closing d and d' are the distances of the camera centers to the plane which are just known in relation to each other. If the metric value of the depth shall be known an additional sensor defining scale should be used.

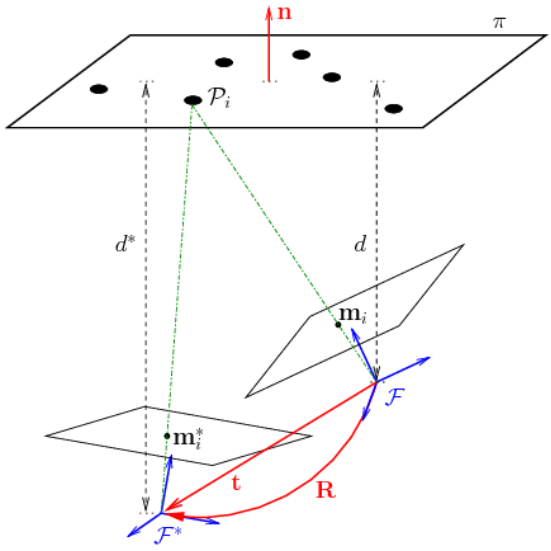


Figure 3.1: Visual representation of the result of the homography decomposition

Filtering the Results

In order to determine the one correct solution of the homography decompositon we have to use two tricks. Two of the returned results do have a negative depth i.e. the plane on which the features are positioned is behind the camera. This is impossible because we know that the plane is in front of our setup. Therefore we eliminate 2 solutions and two are left.

The second algorithm we propose is possible because of our use case on a drone. The setup is described in section 4.1. Since we can be sure that the camera on the drone moves within certain borders we can discard solutions outside this restrictions. We can be sure that the drone rotates just very little around the x and y axis since the drone would get out of control and crash. Furthermore far rotations around the z-axis are also not possible because of the 30 Hz frequency in which we take the pictures. It is just not possible to rotate far in $\frac{1}{30}$ of a second.
Having this constraints in mind we observe the results and if a result is outside the borders it gets eliminated. In the rare case that no other solution gets eliminated we choose the solution with less change from the last position.

4

Implementation

In this section we discuss the setup and the environment we have been running our test in. Additionally we will describe the programs and the code we wrote to implement the algorithms described in the Design chapter 3.

4.1 Setup of the Drone and the Test Environment

The drone we are using is the AscTec Hummingbird drone with a Ubuntu 14.4 running on it. We run our complete code using the ROS framework. The programms have been written in C++. The camera we are using is the Matrix Vision BlueFox with a resolution of 480×752. The camera is fully customizable concerning image frequency, exposure and ISO. We have been doing all tests in a laboratory environment with artificial features sticked on the floor. For the result verification we used a motion capturing system from the company vicon running on a Windows machine. The measurements of the Vicon system are published with 120 Hz into the ROS system using a ROS node connecting the windows system to the Linux based rest of the setup.

4.2 Programms written

In this section we will describe the ROS nodes we wrote and used from different sources and their functions. Additionally we describe the network of subscribers and publishers that connects the nodes.

4.2.1 Picture_align_node.launch

The first group of nodes can be started by executing the launch file *picture_align_node.launch*. It starts 4 nodes and is visualized in figure 4.1. The setup can match a static image to a running image stream. The position of the image stream is displayed with four points in the static map image.

Rosbag play

The first node we start is the robag play node which plays a bag file. It is a standard node of the ROS package. This node is connected to the image_undistort_node.

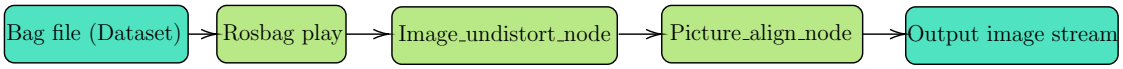


Figure 4.1: Blockdiagramm of the nodes that are started when the picture_align_node.launch file is executed

Image_undistort image_undistort_node

The *image_undistort_node* is a node created by *ZacharyTaylor* a member of the *ethz-asl* group and is available from a github page [31]. It undistorts a image based on undistortion parameters provided in a *.yaml* file. We have been using the radtan undistortion model. It receives images from the rosbag play node and hands the undistorted pictures on to the *picture_align_node*.

Test_node Picture_align_node

The *picture_align_node* is one of the two main nodes we wrote in this thesis. It receives a image stream from the *Image_undistort_node* and displays a image on the screen. The *picture_align_node* is matching a camera stream to a previously defined map image. The position of the camera taking the images for the camera stream is then displayed on the map image. This is achieved by projecting 5 points (a square and the center point of the camera) from the camera stream into the map image. As can be seen in the result chapter 4.1 the points in the camera stream are static, the projected points in the map image move depending on how the camera in the camera stream moves.

To start the node the following files need to be provided:

1. A map image
   This image is mapped to the camera stream

2. Two from Kalibr [32] generated Calibration file
   The *.yaml* file stores the resolution of both, the MAV camera and the map camera.

4.2.2 Extractor_descriptor_matcher.launch

The *extractor_descriptor_matcher.launch* launches the second major group of nodes that we implemented in the extent of this work. The *extractor_descriptor_evaluator_node* which estimates a homography with one of the in section 3.2 outlined matching algorithms. Then the homography is decomposed as described in section 3.4. The resulting position in 3D space is compared to a ground truth provided by a motion-capturing system and publised for RVIZ to display. A visual representation of the nodes that are started can be found in in figure 4.2.

Rosbag play

Is described in section 4.2.1

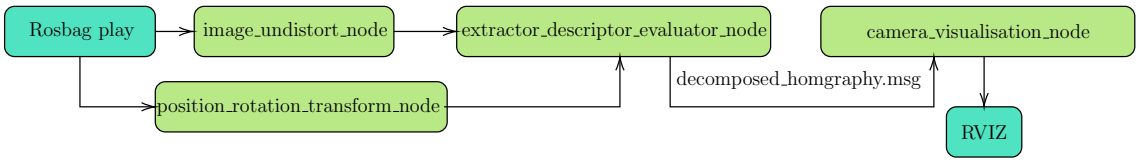


Figure 4.2: Blockdiagramm of the nodes that are started when executing the extractor_descriptor_matcher_evaluator launch file

Image_undistort image_undistort_node

Is described in section 4.2.1

Test_package position_rotation_transform_node

With this node we compensate if a published position is not in the same coordinate system or has an offset. In a *.yaml* file a rotation matrix and a translation can be defined by which the incoming position is rotated and translated. Then the resulting transform is published again.

Test_package camera_visualisation_node

The *test_packagecamera_visualisation_node* is a node that visualizes the estimated camera positions in RVIZ. It receives a *decomposed_homograhy* message which is described in section 4.3.1. If RVIZ is started two camera frustums hovering in space should be visible. One being the frustum at the position where the map image was taken, the second frustum is displayed where the MAV camera is located.

Test_package extractor_descriptor_evaluator_node

The *extractor_descriptor_evaluator_node* takes two images, one being a map image and one from the camera stream. The camera stream image is received from the *image_undistort_node*. Additionally the node collects the pose from the *position_rotation_transform_node* which is used for ground truth calculation. It estimates a homography as the *Picture_align_node* does. Additionally the homography is decomposed as described in section 3.4 and published with a *decomposed_homograhy* message. It requires the following yaml files to start.

1. A map image
   A image to match the image stream to

2. A pose where the map image was taken
   The pose of the map camera at the time the image was taken. Used for ground truth calculations.

3. Two Kalibr calibration *.yaml* files for the map camera and the MAV camera. Contains the camera matrices.

4.3 Custom Generated Messages

In this section we will describe the message that we generated to publish the decomposed homography.

4.3.1 The Decomposed Homography Message

The decomposed homography message was used to hand on the results from a homography decomposition to a next node. The homography decomposition returns three results. The normal vector of the plane on which the features lie. The translation from camera one to camera two and the rotation from camera one to camera two. This is also represented in the fields of the *decomposed homography* message We used a *geometry_msgs/Quaternion* message with the name *rotation_cam1_cam2* to send out the before described rotation from camera one to camera two. Then the message has two more field with *geometry_msgs/Vector3* fields with the name *translation_cam1_cam2* and *normalvector* which contain the other two results of the homography decomposition. Additionally the message has a *Header* which contains the sequence number and the time when the message was send out.

5

Results

In this section we will discuss the results of this work. First we will talk about the frame work that we provided for future experiments with feature matchers. Secondly we will describe our observations of scale invariance of common feature matches and lastly we will discuss the calculations we did to investigate which resolutions are achievable by hovering MAVs on different height.

5.1 The Feature Matching Framework

In the extent of this work we made various experiments with feature matchers to observe their scale invariance. In order to do so we implemented a frame work for feature extractors/descriptors/matchers using opencv that can easily be altered for different use cases. By changing one parameter in the launch file the feature matching algorithms SIFT, SURF, upright SURF and ORB can be used. Additionally it is very easy to exchange cameras because also only the calibration files generated by Kalibr [32] need to be changed. Lastly we provided a Matlab script that extracted the position of an arbitrary image of a bag file based on its time and sequence number. This can be used to work with individual images or if someone wants to control the correctness of matches.

In figure 5.1 you see a screenshot of the working matcher on a arbitrary dataset. On the right side you see the image of the camera stream. On the left side you see the map image against which the camera stream is matched. The red lines from the left picture to the right are the respective matches on which the homography is based. The white points in the right image are the points which are projected in the map image. They are four points in a square and the center point. The pixel position of does not change in the left camera stream. On the right camera stream the white points move depending on where the camera currently is. In our screenshot (figure 5.1) the square in the map image is smaller than in the camera stream because the camera is already closer to the ground than when the map image was taken.

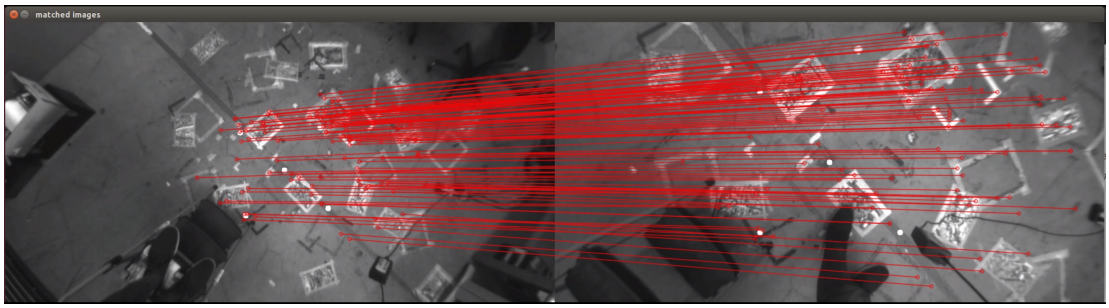


Figure 5.1: Picture of the working matcher

5.2 Observations on Scale-Invariance of Feature-matchers

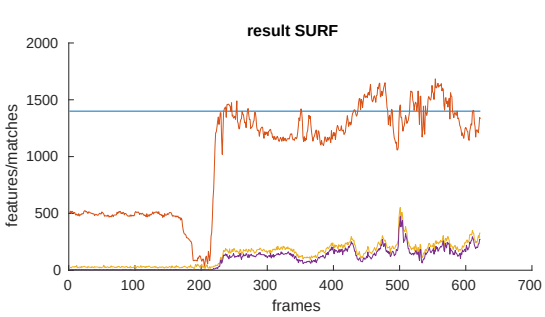
With the feature matcher running we conducted experiments to assess how well scale invariant matchers perform and how many matches are found over all after the different outlier detection mechanisms described in section 3.3 eliminated all bad matches.

5.2.1 Tests on Matching

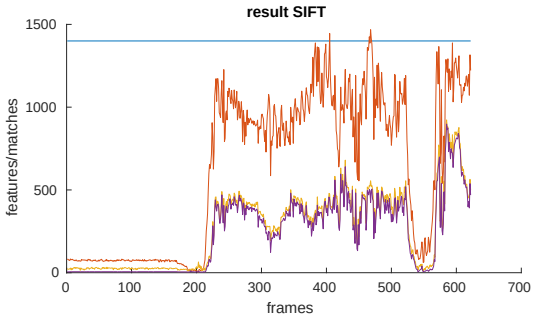
In the first test we ran a arbitrary datasets on SIFT (figure 5.2(b)), SURF (figure 5.2(a)) and upright SURF (figure 5.2(c)). The images are the result of one dataset and describes how we evaluated the matchers.
In the figures 5.2(b),5.2(a) and 5.2(c) we plotted the maximum number of features to be matched with a blue line. This is the number of features that the matching algorithm is at maximum able to match. This number is constant because the map image does not change. Secondly we plotted the features found in each camera image from the camera stream. This is the dark orange line. The bright orange line is then the number of matches obtained after the Lowe ratio test. This is in all cases a much smaller number than the previous number. And the last line which is purple are the remaining features after RANSAC filtering and with that the number of features we use for the homography estimation. In the dataset which plots are displayed here, hover over the ground at a constant speed and rotate the drone. One can see in the upright SURF plot that we rotate the drone trice since the map image and the camera stream align at frame 300,400 and 570. We put the drone to the floor for a short moment at frame 550 which describes the dip in matches at that point. Overall we could observe that SIFT found the most matches and the quality of the homography was the best. Unfortunately we were not able to run the algorithm in real time.

5.2.2 Tests on Scale and Rotation

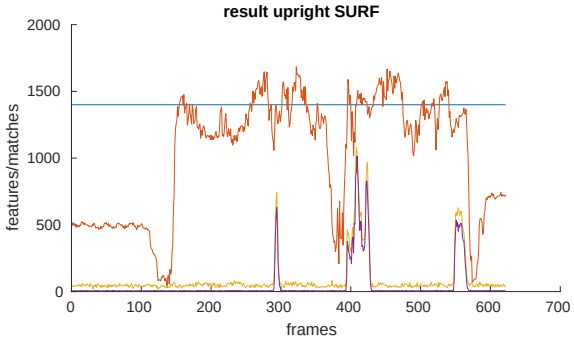
In the second tests that we conducted we observed the scale invariance of feature matchers. Therefore we recorded a dataset in which the camera only translated in z axis. The features in relation to flight height in meters can be found in figure 5.3(a) The red line at about 0.6 meters is the time at which the map image was taken i.e.



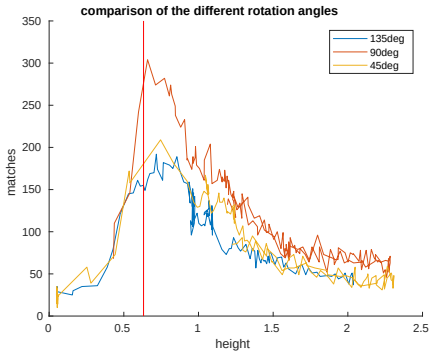
(a) Result of matches using SURF



(b) Result of matches using SIFT



(c) Result of matches using upright SURF

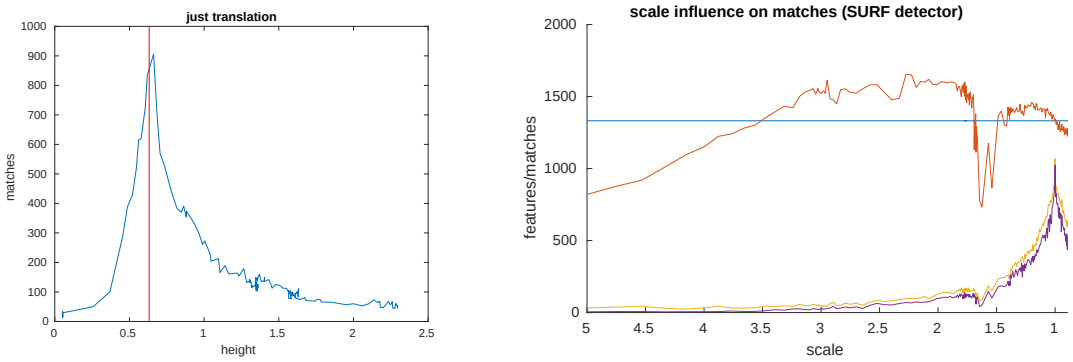


(d) Observation of rotation and scale on matching

Figure 5.2: Example tests on matching rotation and scale

a match against an image with scale 1. We then ran the tests with the before mentioned matchers. The result for the SURF matcher can be found in figure 5.3(b). The plot has the same lines as the plots in which we tested the general performance of matchers. The blue line being the features of the map. The dark orange line the current features on the camera stream. The bright orange line the matches after the Lowe ratio test and the purple line beeing the matches after RANSAC. The result that we were able to observe was that the homography estimation failed at SIFT, SURF and ORB starting at a scale of 2.5 and failed totally at 3. After this point the homography estimation stopped working at all. This is visible in figure 5.3(b) at which one can see that the matches halfed from scale 2 to 2.5 and even stronger after the scale of 2.5.

In the tests of figure 5.2(d) we observed how different rotations have an influence on the scale. We tried to assess with this tests if the warp into the right rotation (if a rotation of the camera is known) is viable. As we can see in the comparison between picture 5.2(d) and figure 5.3(a) we can see that we loose at least 50 percent of the matches because of rotations between 45 and 135 degrees. Therefore an idea to improve matching would be to warp images in the same orientation. Additionally computation time could be saved because we could use a rotation-variant feature matcher as upright SURF.



(a) Matches depending on height

(b) Matches depending on scale

Figure 5.3: Tests on scale

5.3 Theoretical Observations on Scale

In order to quantify the circumstances a orbital image to drone image matcher has to work in we calculated the meters per pixel of two different cameras. The first camera being a matrix vision camera with a resolution of 752×480 and the ueye LE with a resolution of 1280×1024.

Because satellites always fly at constant height the resolution per pixel is constant. With HiRise having a resolution of 0.3 meters per pixel under perfect circumstances. This is horizontal line in figure 5.4. When the blue resolution line of the matrix vision or the red line of the ueye camera cross the yellow border the resolutions of both cameras are the same. This is the case in y axis at a flight height of about 112 meters with the matrix vision and 192 meters with the Ueye LE camera. This means that it is not an option to fly high in order to equalize scale since these heights are above normal flight height on earth and significantly over the flight heights of MAVs on other planets.

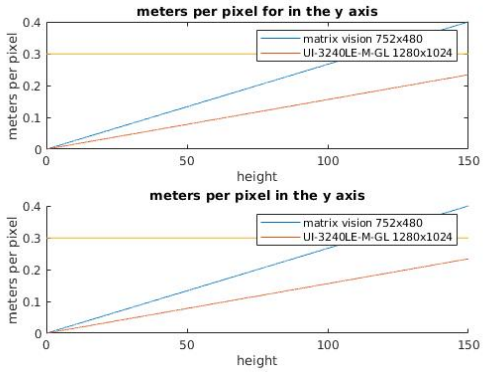


Figure 5.4: Pixel per meter of the two cameras which we used depending on flight height

The calculation is described in the formulas below while f_c is the focal length in pixel, S is the sensor size in meters, R is the camera resolution. α is the angle of view and h is the respective flight height i.e.the distance from the feature surface.

In formula 5.1 we calculate the effective focal length in meters from the focal length in pixels that we determined by a camera calibration and which is available in the camera matrix. The sensor size S is published by the producers and is readily available for most sensors in the internet. The effective focal length f_e is than metric.

In the next step in formula 3.4 is to calculate the angle of view the formula results from the camera geometry and the result is in degree. In the last step the Meters per Pixel (MPP) are calculated. The results we obtained are in the cases of our cameras similar in x and y resulting in square pixels. For the Matrix vision camera we obtained a MPP of $0.0027 \cdot h \;\; m/pixel$. For the Ueye LE camera the MPP we calcualted is $0.0016 \cdot h \;\; m/pixel$. Plots of this result can be seen in figure 5.4.

$$f_e\,[m] = f_c\,[pixel] * \frac{S\,[m]}{R\,[pixel]} \tag{5.1}$$

$$\alpha = 2 * atan(\frac{S\,[m]}{2 \cdot f_e\,[m]}) \tag{5.2}$$

$$MPP\,[m/pixel] = \frac{2 \cdot h\,[m] \cdot tan(\alpha \cdot 0.5)}{R\,[pixel]} \tag{5.3}$$

6

Conclusion and Futurework

In this work we worked for 6 month on Feature matchers and their scale-invariance. We build a frame work which makes it easy in future to make different matching algorithms work. Additionally we determined how robust SIFT, SURF and ORB are against Scale. Furthermore we ran experiments of how to estimate homographies and how to decompose them in order to estimate the position of a camera in 3D space. Closing we looked at the theoretical circumstances under which a feature matcher between a satellite image and a camera flying on a MAV had to work.

The predefined goal of this work was to match satellite images to a stream of images from a flying drone. Unfortunately the time ran out to do outdoor tests with images with strong scale differences. Therefore the first step after this work handed in we should run tests outside in a real life test environment and see if it is possible to make SIFT, SURF or ORB run under real life circumstances. Additionally we did not look deeply into Satellite images. It is interesting how this images have to be processed to match them to real live images. Moreover what camera matrix is used to normalize orbital images. If feature matching algorithms would work on Satellite image it would be possible for example to observe environmental change automatically.
Finally it would be necessary to match images between a MAV mounted camera to orbital images. First using readily available Google maps images followed by matching images with similar circumstances as on Mars. This would be possible if datasets are collected in analogue missions for example in the desert of Nevada or in Oman where the environment looks the same as on the red planet.

Lastly the feature matchers should be altered for the use cases with high scale invariance. Therefore more measurements and sensors should be incorporated.

Bibliography

[1] "Subterranean Challenge," https://www.subtchallenge.com/, accessed: 2019-06-29.

[2] S. Chiodini, M. Pertile, S. Debei, L. Bramante, E. Ferrentino, A. G. Villa, I. Musso, and M. Barrera, "Mars rovers localization by matching local horizon to surface digital elevation models," *4th IEEE International Workshop on Metrology for AeroSpace, MetroAeroSpace 2017 - Proceedings*, pp. 374–379, 2017.

[3] R. Li, K. Di, J. Wang, S. He, A. Howard, and L. Matthies, "Rock Modeling and Matching for Autonomous Mars Rover Localization," *Citeseer*, 2009. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Rock+Modeling+and+Matching+for+Autonomous+Mars+Rover+Localization#0

[4] Y. Tao, J. P. Muller, and W. Poole, "Automated localisation of Mars rovers using co-registered HiRISE-CTX-HRSC orthorectified images and wide baseline Navcam orthorectified mosaics," *Icarus*, vol. 280, pp. 139–157, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.icarus.2016.06.017

[5] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, nov 2004. [Online]. Available: https://doi.org/10.1023/B:VISI.0000029664.99615.94

[6] H. Bay, T. Tuytelaars, and L. V. Gool, "LNCS 3951 - SURF: Speeded Up Robust Features," *Computer Vision–ECCV 2006*, pp. 404–417, 2006. [Online]. Available: http://link.springer.com/chapter/10.1007/11744023_32

[7] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2564–2571, 2011.

[8] T. Tuytelaars and K. Mikolajczyk, "Local Invariant Feature Detectors: A Survey," *Foundations and Trends® in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2007.

[9] C. Wu, F. Fraundorfer, and J. Frahm, "Image localization in satellite imagery with feature-based indexing," *Proceedings of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Beijing: ISPRS*, pp. 197–202, 2008. [Online]. Available: http://www.cs.unc.edu/$\sim$jmf/publications/ISPRS_2008_Wu_et_al.pdf

[10] A. B. Chan, Z. S. J. Liang, and N. Vasconcelos, "Privacy preserving crowd monitoring: Counting people without people models or tracking," *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008.

[11] Z. Ren, J. Yuan, and Z. Zhang, "Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera," p. 1093, 2011.

[12] M. Shan, F. Wang, F. Lin, Z. Gao, Y. Z. Tang, and B. M. Chen, "Google map aided visual navigation for UAVs in GPS-denied environment," *2015 IEEE International Conference on Robotics and Biomimetics, IEEE-ROBIO 2015*, pp. 114–119, 2015.

[13] G. Conte and P. Doherty, "An Integrated UAV Navigation System Based on Aerial Image Matching," 2008. [Online]. Available: https://www.ida.liu.se/divisions/aiics/publications/AEROCONF-2008-Integrated-UAV-Navigation.pdf

[14] F. Lindsten, J. Callmer, H. Ohlsson, D. Törnqvist, T. B. Schön, and F. Gustafsson, "Geo-referencing for UAV navigation using environmental classification," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1420–1425, 2010.

[15] J. W. J. Hwangbo, K. Di, and R. Li, "Integration of Orbital and Ground Image Networks for the Automation of Rover Localization," *ASPRS 2009 Annual Conference, Mar*, pp. CD–ROM, 2009. [Online]. Available: http://shoreline.eng.ohio-state.edu/publications/09asprs_hwangbo.pdf

[16] K. Di, Z. Liu, and Z. Yue, "Mars Rover Localization based on Feature Matching between Ground and Orbital Imagery," *Photogrammetric Engineering and Remote Sensing*, vol. 77, no. 8, pp. 781–791, 2013.

[17] Y. Tao, J. P. Muller, and W. Poole, "Automated localisation of Mars rovers using co-registered HiRISE-CTX-HRSC orthorectified images and wide baseline Navcam orthorectified mosaics," *Icarus*, vol. 280, pp. 139–157, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.icarus.2016.06.017

[18] A. V. Nefian, X. Bouyssounouse, L. Edwards, T. Kim, E. Hand, J. Rhizor, M. Deans, G. Bebis, and T. Fong, "Planetary rover localization within orbital maps," *2014 IEEE International Conference on Image Processing, ICIP 2014*, pp. 1628–1632, 2014.

[19] E. Boukas, a. Gasteratos, and G. Visentin, "Localization of Planetary Exploration Rovers with Orbital Imaging: a survey of approaches," *Wmepc14.Irccyn.Ec-Nantes.Fr*, 2014. [Online]. Available: http://wmepc14.irccyn.ec-nantes.fr/material/paper/paper-Boukas.pdf

[20] J. Delaune, G. Le Besnerais, T. Voirin, J. L. Farges, and C. Bourdarias, "Visual-inertial navigation for pinpoint planetary landing using scale-based landmark matching," *Robotics and Autonomous Systems*, vol. 78, pp. 63–82, 2016.

[21] A. I. Mourikis, N. Trawny, S. I. Roumeliotis, A. Johnson, and L. Matthies, "Vision-Aided Inertial Navigation for Precise Planetary Landing: Analysis and Experiments," *Robotics*, 2018.

[22] U. Chris, A. Joshua, B. Drew, B. Christopher, B. Robert, N. C. M, D. John, D. Dave, G. Tugrul, G. Chris, G. Michele, H. Sam, H. Martial, M. H. Thomas, K. Sascha, K. Alonzo, L. Maxim, and F. Dave, "Autonomous driving in urban environments: Boss and the Urban Challenge," *Springer Tracts in Advanced Robotics*, vol. 56, no. November 2009, pp. 3–59, 2009. [Online]. Available: http://www3.interscience.wiley.com/journal/120846964/abstract

[23] "ROS mainpage," https://www.ros.org/, accessed: 2019-06-29.

[24] "ROS topic naming conventions," https://wiki.ros.org/Names, accessed: 2019-06-29.

[25] J. Canny, "A computational approach to edge detection," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 1986.

[26] I. Sobel, "An Isotropic 3x3 Image Gradient Operator," no. August, 2015.

[27] C. Harris and M. Stephens, "a Combined Corner and Edge Detector," pp. 147–152, 1988. [Online]. Available: http://courses.daiict.ac.in/pluginfile.php/13002/mod_resource/content/0/References/harris1988.pdf

[28] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," pp. 430–443, 2006.

[29] H. C. Longuet-Higgins, "A Computer Algorithm for Reconstructing a Scene from Two Projections," *Nature*, vol. 293, no. September, pp. 133 – 135, 1981.

[30] E. Malis and M. Vargas, "Deeper understanding of the homography decomposition for vision-based control To cite this version : HAL Id : inria-00174036 Deeper understanding of the homography decomposition for vision-based control," 2007.

[31] "Github repository of the Image_undistort node," https://github.com/ethz-asl/image_undistort, accessed: 2019-06-29.

[32] "Github repository of the Kalibr Calibration tool," https://github.com/ethz-asl/kalibr, accessed: 2019-06-29.