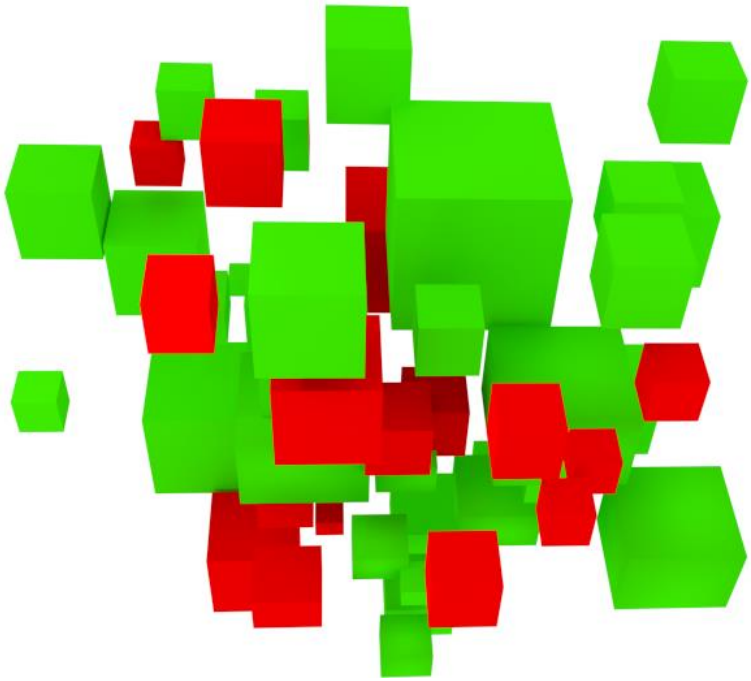
Master Thesis

Improving robustness of pose estimation in AR using motion segmentation

Rolf Janszen
TU Delft

Supervisors:
prof.dr.ir. Pieter Jonker
ir. Xin Wang

Delft Biorobotics Laboratory
Department of BioMechanical Engineering
TU Delft
May 6, 2015



Content

Abstract

Recent improvements in mobile technology have allowed computationally intensive augmented reality systems to become increasingly ubiquitous. As will be shown in this thesis an increasing number of companies have started developing markerless AR systems that can be used professionally. Robust pose estimation is essential for such systems to convey users into a believable AR experience by excluding outlying and erroneous tracked features. Robust statistics can detect and reject large percentages of outliers but in highly dynamic scenes more dedicated motion segmentation is required.

In this thesis a motion segmentation algorithm is presented that is designed to increase robustness in a vision based marker-less stereo AR framework, to improve pose estimation and mapping. This is based on a motion segmentation method that has successfully been tested on PTAM [1]. We believe that by expanding this to stereo vision, improvements can be made by utilizing the extra visual information obtained by the second camera.

By modeling the stereo triangulation error we could use the Mahalanobis distance for clustering features and correct for their error distribution, which increases with distance from the camera. Initial visual inspection showed that this improved the dynamic feature rejection with respect to the Euclidean distance and with that also the accuracy of pose estimation. This is shown in the test results.
By using the 3D distribution in the scoring of each set, the correct labeling of a set of clustered features as being static was improved. This is because static features are more likely to be arbitrarily spread over the 3D scene while dynamic features belonging to the same moving object are more likely to be grouped in a relatively planar configuration. Stereo vision provides 2.5 D information (in contrast with X-ray images) and moving objects are generally smaller than the 3D scene in which the user resides itself.

Acknowledgment

I hereby would like to thank Pieter Jonker for giving me the opportunity to do my master thesis on the subject of computer vision.

I also would like to thank Xin Wang for learning me to program in C++, using the OpenCV library and teaching me to write on an academic level, you have been a great help.

Also I would like to thank the people in the BioRobotics lab; Boris Lenseigne, Floris Gaisser, Wouter Caarls, Toby Huyssen and Aswin Chandarr, for helping me with fixing bugs and discussing problems, it was nice to be able to walk in if I had a question.

1. Introduction

Recent advances in mobile technology have made computationally intensive augmented reality (AR) applications become more widely available. Although these applications are still very limited (mainly marketing gimmicks), AR could reach its full potential in 5 to 10 years with further development and improvements in technology. Figure 1.1 shows for example an AR application that shows venues and spots in the user's vicinity that might be interesting.

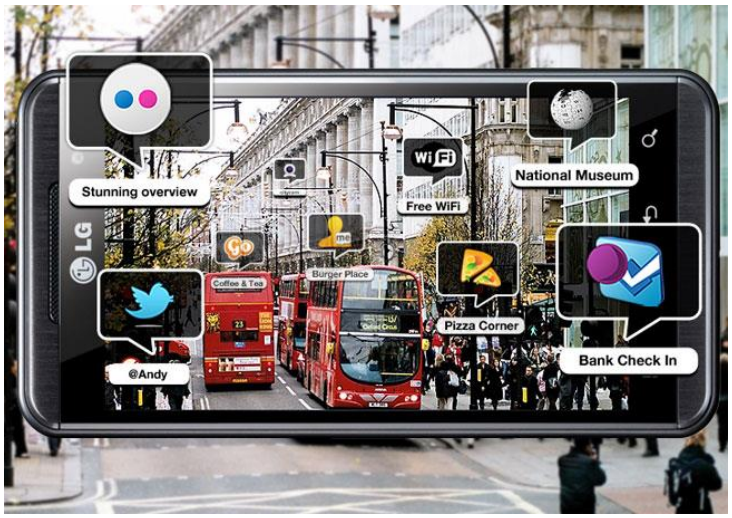


Figure 1.1 AR application that can show the user points of interest in the vicinity.

1.1 Augmented reality (AR)

In AR, virtual images are superimposed over those of the real world so the user can view and interact with them. Azuma [2] defined AR as a technology that;

- Combines real and virtual imagery
- Is interactive in real time
- Registers the virtual imagery with the real world

For the AR experience to become believable to the user the system needs to perform these three tasks convincingly. The AR system therefore needs to have an accurate notion of its own position and orientation (pose) in relation to the environment, even when the user wearing the AR device moves around.

1.2 Previous work and developments

The earliest AR system was developed by Sutherland in the 60's [3], who made a device which projects wire-frame images onto a see-through head-mounted display. The wire-frame images were projected in accordance to the movements of the user's head, which was tracked using ultra-sonic and mechanical sensors.

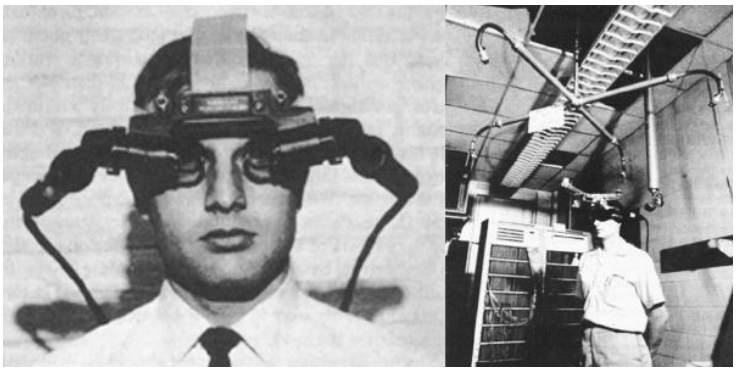


Figure 1.2 The first AR installation developed by Sutherland in the 60's called the sword of Damocles because of the large structure that hangs over the user.

In 1992 a team from Columbia University created what can be considered to be one of the first real AR systems; Knowledge based Augmented Reality for Maintenance Assistance (KARMA) [4]. It uses markers attached to an object to determine the pose of that object. It overlays instructions on the video image which the user can then carry out.
From that moment on many AR systems have been designed to also use visual information for pose estimation.

Depending on the AR system, the entire process of registering the environment and estimating the pose can be computationally expensive. Because of the increased computational power of mobile devices they can now support AR applications that make use of the integrated sensors and cameras. One example of such an application is a ghost capture game SpecTrek. In it, ghosts are placed in an area with a radius of 124 meters which the user can capture using the phone's camera, as shown in Figure 1.3.

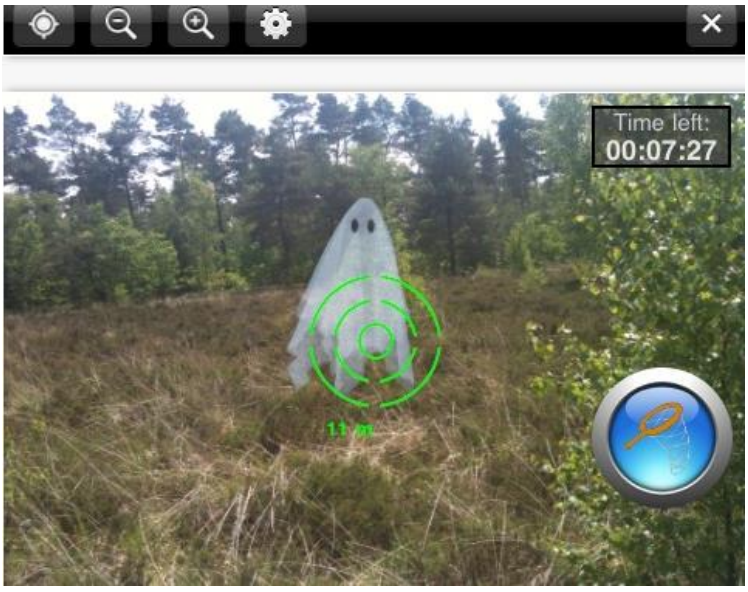


Figure 1.3 SpecTrek; A simple AR game about hunting ghosts in the area. It uses GPS to track the users position and once close to a ghost it can be displayed in the camera image and be caught.

Nintendo has always been known for pushing novel gaming interfaces and experiences. So when Nintendo launched the Nintendo 3DS it also featured some small AR games. The 3DS has a camera on the back side that tracks markers to update its pose.

Figure 1.4 AR games on the Nintendo 3DS uses markers for pose estimation.

A very promising advancement in AR comes from a Silicon-Valley start-up Meta which develops the 3000 dollar Meta-pro [5]. The Meta-pro creates a real 3D virtual interface where users can interact with virtual objects, giving a holographic computing experience. Meta sees surgery and 3D CAD as promising fields to apply this technology. In order for the Meta-Pro to estimate its own pose and the position of the users hand it needs reference information from the environment which it receives through a 3D time-of flight depth sensor, a stereo-camera, accelerometers, a gyroscope and a compass. To process all this information it has a 1.5 GHz i5 Intel processor and 4Gb RAM, which is located in a separate unit connected through a wire to the glasses. The Meta-pro requires all these sensors because believable AR needs a reliable and accurate pose estimation to be convincing, in contrast to virtual reality (VR) that also works well with a rougher estimate. The high price, however, makes that the Meta-pro is primarily aimed at professional user.

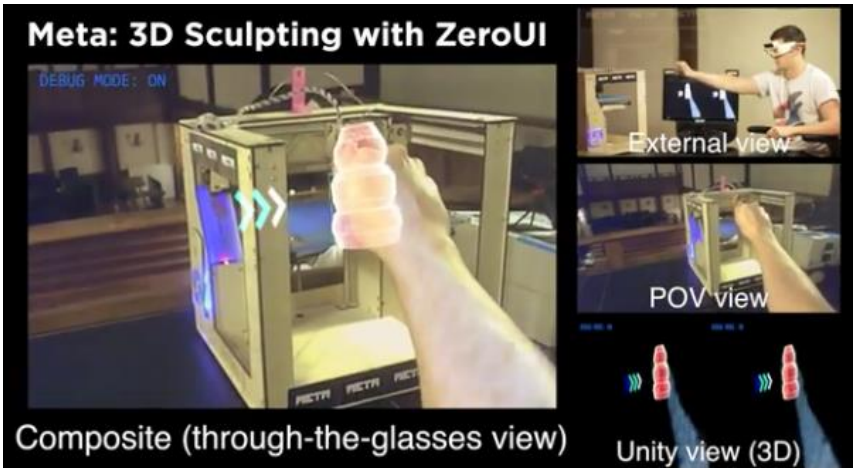


Figure 1.5 The development of the Meta-Pro is currently also focused on using it as a more intuitive 3D CAD tool. In this figure it is used to sculpt objects that can be dragged-dropped to a 3D printer to start prototyping.

In 2007 Klein and Murray developed a popular AR framework; Parallel Tracking And Mapping (PTAM) [1] (see section 2.2 for more details). It was the first AR framework to benefit from parallel processors, that became increasingly common in computers, by running the tracking and mapping processes in parallel threads. PTAM has been adapted by O. Akman to use a stereo- instead of a monocular camera; Stereo Tracking And Mapping (STAM) [6]. The advantage of a stereo camera is that the baseline between the right and left camera is known giving a real world scale reference to estimate 3D coordinates in true scale. Stereo images can be rectified so that finding image correspondences is easier through stereo-matching. More on the subject of stereopsis in chapter 2.2. STAM will also be the framework in which the method to make pose estimation more robust against dynamic features will be implemented.

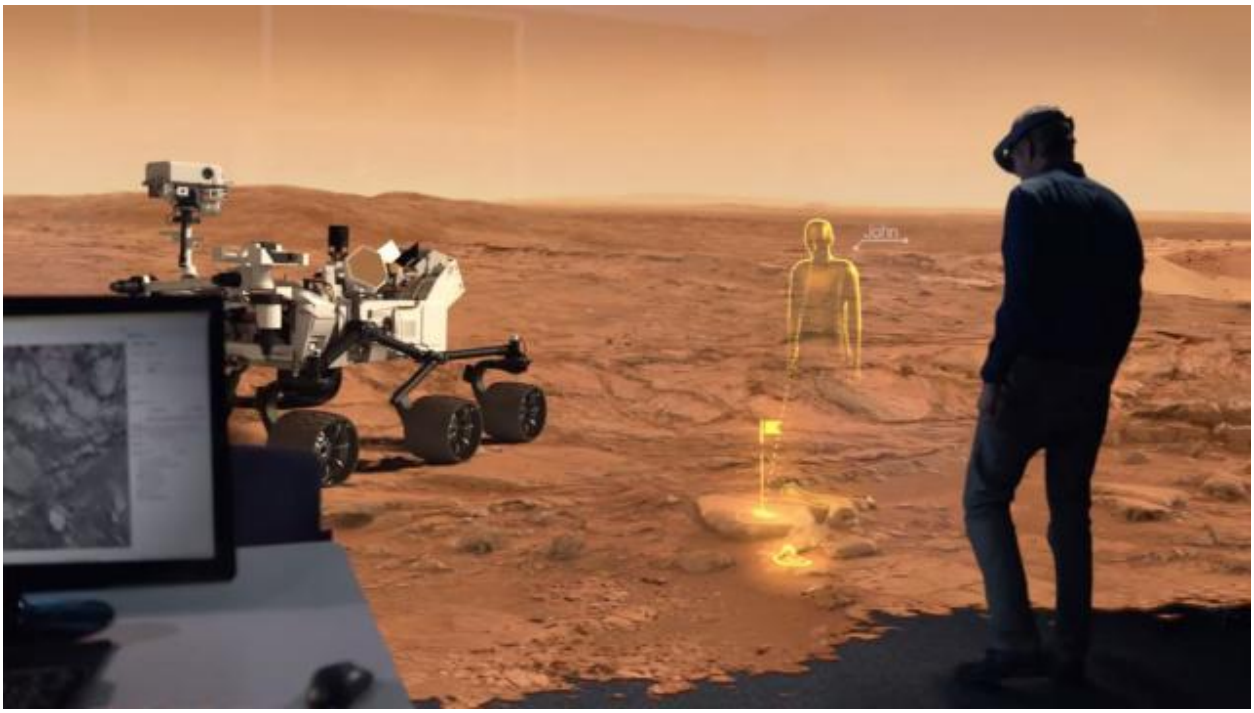


Figure 1.6 Microsoft has collaborated with NASA to recreate a 3D environment of Mars from pictures provided by Curiosity. This environment could then be explored using its newly developed Hololens.

In January 2015 Microsoft announced something similar to the Meta-Pro that would also work with their new operating system Windows 10; the Hololens, Figure 1.6. It hasn't released much technical details but looking at the head band it seems to contain several stereo camera's which are probably used for head tracking. Having such a large company investing in and promoting this technology might very well spur AR development.

1.3 Challenges and difficulties in AR

There is still a long way for AR to become a part of people's everyday life. A lot of challenges and limitations still need to be improved before AR will be able to showcase a wider range of possibilities. Ö. Blissing [7] gave a comprehensive summation of some of these limitations and challenges of which a short overview is given.

1) *Portability*

Recent developments in mobile devices such as smart-phones and tablets have made AR more widely available. Equipped with cameras and additional sensors they form a good platform for AR applications. This trend makes the cumbersome backpacks required for larger devices ultimately obsolete.

2) *Tracking*

Accurate tracking is crucial in AR applications and is a fundamental enabling technology for AR. With the most cited ISMAR paper, and five out of ten of the highest cited ISMAR papers being about this subject [8]. For reliable tracking distinguishable features, such as corners and edges are required in the scene. Large monotonic surfaces therefore pose problems, are hard to be recreated correctly and generally show up as dark spaces in dense depth maps. Changes in illumination or occlusion can cause loss of the tracked feature.

3) *Outdoor use*

Tracking and mapping large amounts of features can take up an increasing amount of memory. This will also cause computational time to increase exponentially when these feature points are computed using a probabilistic framework. Especially in (open) outdoor spaces the risk increases to end up with a large map which is less so the case indoors. These environments can also be a lot more hectic with the scenes becoming highly dynamic with multiple independent motions and frequent partial occlusions. Outlier detectors like RANSAC can in such cases become overwhelmed and fail [9] [10].

4) *Depth perception*

Depth information can be calculated using multiple (consecutive) 2D images, and is needed for pose estimation and 3D map construction. Systems using monocular cameras in this case are not able to obtain a sense of scale without some form of reference. Using calibrated stereo cameras this problem can be overcome using the baseline as reference. But most stereo vision algorithms are still in its infancy and need additional development.

5) *Overload and over-reliance*

In order to be usable, AR systems shouldn't overload the user through its interface and avoid having the user to become overly reliant on the displayed information. This may cause that he or she misses important cues from the environment.

6) *Computational load*

For a realistic experience, real time performance in AR is required. The human threshold of perceiving images as being separate is 10 to 12 frames per second. 24 frames per second is the widely adopted rate in video, meaning the computational time, and thus the algorithm, needs to be fast enough to process and render approximately within this time frame. As mentioned previously, AR requires a high computational load for tracking, pose estimation and map updating. Especially purely vision based systems need complex algorithms to perform accurately and robustly.

1.4 Research goal

As aforementioned in challenge number 3; the moving features can disrupt accurate pose estimation and therefore disrupt mapping and tracking. The goal of this research is to improve the pose estimation of an existing vision based AR framework in dynamic scenes by implementing a motion segmentation method into its tracking process. It might be important to consider that the camera itself is not static, because in such a case simple subtraction of previous frames would already be sufficient. Four steps can be considered to reach this goal;

- Adapt a motion segmentation method from literature.
- Design a proper way to implement it into the AR framework.
- Optimize its performance inside that framework.
- Test performance in dynamic scenes compared to several state of the art methods.

The algorithm we used to make the framework more robust in dynamic scenes is based on a method from literature [11] and is adapted to work with input from a stereo camera [6]. For the implementation a good location inside the AR framework needs to be found. From there the motion segmentation algorithm can filter out features from moving objects.
Parameter settings in this method need to be fine-tuned to make the motion segmentation run effectively while also minimizing computational load. This fine-tuning has to be done experimentally on test-data and also includes trying out some adaptations to improve the segmentation method.
An experimental setup has to be devised to see if the goal of improved pose estimation robustness has actually been achieved and how it compares to other state of the art visual odometry algorithms. This requires ground-truth data of the pose and related stereo images. Since there is no motion segmentation dataset readily available for a stereo setup, it has to be created.

1.5 Thesis outline

The outline of this thesis will be as follows. Chapter 2 will explain the main processes inside a stereo vision based AR program. Some background theory on these processes will be presented along with the AR framework that is used in this research. The motion segmentation that will be implemented in that AR framework will be explained in chapter 3. The integration of the segmentation algorithm is presented in chapter 4. The setup to test the performance of the AR framework with motion segmentation is explained in chapter 5 along with their results. The conclusions that can be drawn from these tests results are then discussed in chapter 6.

2 AR methods and background theory

Before going further into the details of this research, some background theory on the process of pose tracking will be explained.

2.1 Vision based tracking techniques

To estimate the position and orientation (pose) of the AR framework, using visual information, the system needs to detect cues that can be used as reference from the scene. These can be either fiducials or models.

- *Models;*
  In this case the system tracks structures in the scene of which it has a 3D model stored in its memory. This model is then used to align with the geometry as recorded by a camera.
  However, since this method is of no relevance to this research it will not be explained any further in this report.
- *Features;*
  This can be either;
  - *Fiducials*; patterns that have been placed in the environment and match a pattern stored in the system's memory.
  - *Natural features;* points that occur naturally in the scene and can be considered interesting and informative.

2.1.1 Fiducials

Markers are (printable) patterns that can be placed in the environment. A model of the marker is stored in the system's memory so it can be aligned with the visual input. Several examples of AR markers are shown in Figure 2.1.

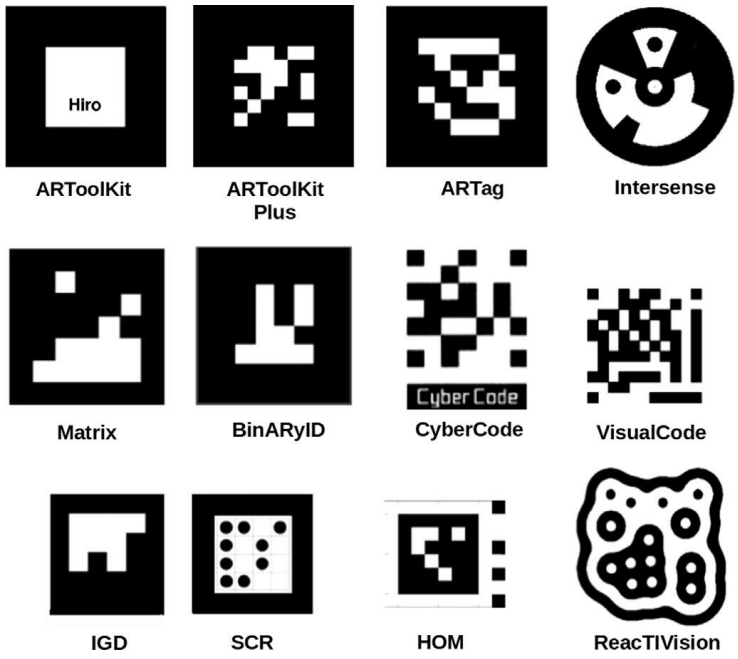


Figure 2.1 taken from [12]; Different AR markers from different frameworks. They all have high contrasting asymmetric patterns.

Some properties most AR markers have in common are:

- *High contrasts*; this helps the system with detecting the marker in the scene.
- *An anti-symmetric pattern*; to determine the correct pose relative to the marker, the system needs to tell the difference between its four sides.

Once the system has detected the marker in the camera frame it will try to align it with the stored marker model. The transformation of that projection can then be related to the pose of the camera.

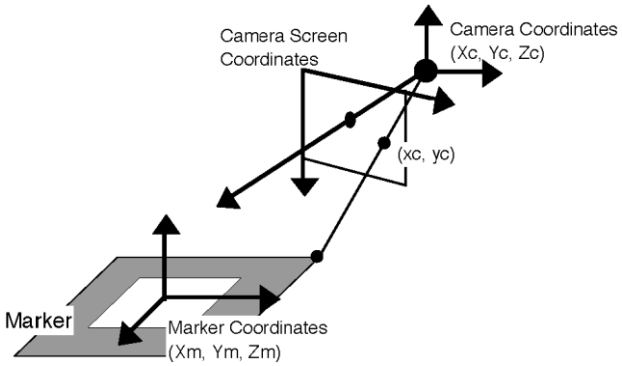


Figure 2.2 A square fiducial marker, in grey, is detected from its four corners. The camera pose can then be estimated by minimizing a re-projection error.

This method is accurate and computationally relatively inexpensive. But a marker must be in view the whole time for the system to be able to estimate its pose. So the camera can either only make small movements, or multiple markers need to be placed in the environment to cover for larger movements. But, when placing multiple markers in the environment their relative positions need to correspond accurately to how the lay-out of those markers is stored in the system, which requires precise measuring.

Edge detectors are often used to find markers in the scene [12]. The error for edge detection depends on the observed line thickness, the thicker a line the smaller the error. This error also varies depending on the evaluation distance of the edge detectors. J. Caarls [13] showed that an evaluation distance of 2 pixels works well for a line thickness of 5 pixels or more, below that an evaluation distance of 1 pixel performs better. He therefore proposed varying this evaluation distance according to the line thickness.

2.1.2 Natural features

A natural feature is an area detected in the scene that is considered interesting. An example of what a feature represents is shown in Figure 2.3. Most feature detectors consider such an area to be interesting if it has high image intensity gradients because this makes it more distinct and informative, which means that it has characteristics by which it can be easily re-detected and matched over multiple frames. Reliable features have high intensity gradients in multiple directions, preferably two, indicating a corner. These gradients are generally also used for classification and description.

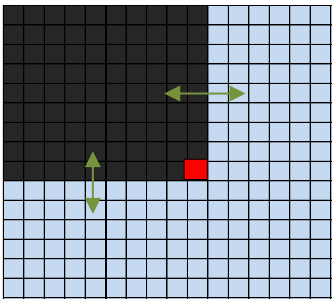


Figure 2.3 A feature patch, the image coordinate of the pixel indicating the feature' s coordinate is marked with a red square. The green arrows indicate the direction of the intensity gradients.

There are a multitude of methods to classify and describe features which is usually done by using the intensity gradients. For example a SIFT feature is described by the average gradient in each sub-patch of the feature patch, as shown in Figure 2.4 taken from [14].

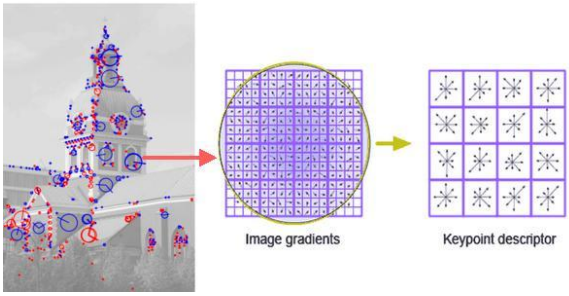


Figure 2.4 A SIFT feature. The arrows indicate the direction of the gradient, their length the magnitude. In this case the feature of 16x16 pixels is divided into 16 4x4 sub-patches for each of which the main gradient is computed. In the end the descriptor for this feature will be a vector of 128 numbers.

The pose can be estimated using the 2D coordinates of the features. In monocular visual odometry methods this is usually done by re-projecting the 3D world coordinates to the 2D coordinates. These 3D coordinates need to be obtained first however by triangulation over corresponding 2D image coordinates.
Using features from the environment avoids the preparation issues that come with fiducials but brings up new ones that are mainly due to problems from continually tracking them over consecutive frames. Just a few of these problems are:

- *Occlusions*; either by other objects or a part of the object on which the feature is detected can move in front of the view. Drift can occur if loss of the feature is unnoticed.
- *Changes in brightness*; As the camera moves, the reflection with respect to the light source can change, causing changes in brightness making feature matching prone to errors.
- *Drift*; matching over consecutive frames can cause incremental tracking errors and slowly change, the characteristics of the feature that initially was tracked.
- *Independent motion of objects*; Reliable pose estimation becomes impossible if the data-points used in its estimation process show different motion patterns. This problem and its solution will be elaborately explained henceforth since it's the research topic of this thesis.

Because the use of natural features is central to this research, the methods and their theory to use it in visual odometry methods, such as natural feature based AR, is explained more in depth in the following section.

2.1.2.1 Optical flow tracking

The use of optical flow is currently the method of choice to efficiently track features over consecutive images. This method was pioneered by B.D. Lucas and T. Kanade [15]. (1) shows the idea behind optical flow; the principle of brightness constancy. This means that the brightness of a pixel patch in a new image I at time $t+1$ at position $x_i + V$ should be similar to the patch in the previous frame at t at position x_i. Here x represent a patch of pixels of size 5x5 $x = \{x_1, x_2, \dots, x_{25}\}$, each pixel x_i with image coordinate $x_i = [\mathrm{u}_i, \mathrm{v}_i]$. And V is the change in pixel location $[\Delta\mathrm{u}, \Delta\mathrm{v}]$, so for ease of notation; $\Delta x \equiv V \equiv [\Delta\mathrm{u}, \Delta\mathrm{v}]$.

$$\mathrm{I}(x,t) - \mathrm{I}(x+\mathrm{V},\mathrm{t}+1) = \sum_{i=o}^{N} \mathrm{I}(x,t) - \mathrm{I}(x+\mathrm{V},\mathrm{t}+1) = 0 \qquad (1)$$

Taking the Taylor expansion it can then be written as in (2) where ∇I is the spatial intensity gradient. The contribution from second and higher order terms O^2 is negligible so they can be left out.

$$\mathrm{I}(x+\mathrm{V},\mathrm{t}+1) = \mathrm{I}(x,t) + \nabla I \cdot \mathrm{V} + I_t + O^2 \qquad (2)$$

$$\nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix} \qquad (3)$$

From (1) it is clear that in (2) the second and third right hand term should add up to zero, as shown in (4). Then to obtain the optical flow we have to solve the motion constraint equation (4).

$$\nabla I \cdot \mathrm{V} + I_t = 0 \qquad (4)$$

Equation (4) can then be solved for V using a linear least squares method as shown in Figure 2.5 over multiple features (a patch) to obtain an over defined equation (6).

$$V = (\nabla I^T \nabla I)^{-1} \nabla I^T \, I_t \qquad (5)$$

$$V = \begin{bmatrix} \Delta\mathrm{u} \\ \Delta\mathrm{v} \end{bmatrix} = \begin{bmatrix} \sum I_x(x_i)^2 & \sum I_x(x_i) I_y(x_i) \\ \sum I_y(x_i) I_x(x_i) & \sum I_y(x_i)^2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} \sum I_x(x_i) \sum I_t(x_i) \\ \sum I_y(x_i) \sum I_t(x_i) \end{bmatrix} \qquad (6)$$

When trying to estimate the motion of a small patch that only has an image gradient in one direction, i.e. an edge or a line, it becomes hard if not impossible to determine its velocity without seeing either end of the edge or a line. This problem is known as the aperture problem. Estimation of the local optical flow can therefore only be done reliably on sufficiently textured patches such as corner features.

A downside of estimating the optical flow using the linear method is that it only holds for small motions. A larger patch size can be used to keep the feature within range of the window. But using a larger window can smooth out the details in the image making the optical flow estimate less accurate. It also can pick up features on other bodies as well so that not all points move as their neighbor.

Using multiple scale pyramids of the image solves the problem with large motions [16]. At each level the optical flow is estimated in an iterative manner and used as initial estimate for the next level of higher resolution. The image difference $I_t(p)$ then becomes;

$$I_t(x) = \mathrm{I}(x) - \mathrm{J}(x + g^L + V_{k-1}) \quad (7)$$

Where V_k is the updated optical flow from the last iteration step or residual displacement since there is already an initial estimate, so it just needs to be refined at each new scale and iteration. This term then also has consequences for the update for each new scale;

$$g^{L-1} = scaling\_factor \cdot (g^L + V^L) \quad (8)$$

The scaling factor is usually set to 2, where the tracking starts at the smallest scale. The image gradient will then be calculated as in (9) using a central difference operator.

$$\nabla I(x) = \begin{bmatrix} I_x \\ I_y \end{bmatrix} = \sum_{i=o}^{N} \begin{bmatrix} \frac{\mathrm{I}(x_i + 1, y_i) - \mathrm{I}(x_i - 1, y_i)}{2} \\ \frac{\mathrm{I}(x_i, y_i + 1) - \mathrm{I}(x_i, y_i - 1)}{2} \end{bmatrix} \quad (9)$$

The solution for each iteration for one scale is near the point where V hardly changes anymore; $\|\Delta \mathrm{V}\| < accuracy\ treshold$. So at each k^{th} iteration at level L $\mathrm{V}_{\mathrm{k}}^{\mathrm{L}}$ is updated by (10) and used to update $I_t(p)$ with (7).

$$\mathrm{V}_{\mathrm{k}}^{\mathrm{L}} = \mathrm{V}_{\mathrm{k}-1}^{\mathrm{L}} + \mathrm{V} \quad (10)$$

The new $I_t(p)$ is then used to update the Right most 2×1 matrix holding $\begin{bmatrix} \sum I_x(x_i) \sum I_t(x_i) \\ \sum I_y(x_i) \sum I_t(x_i) \end{bmatrix}$ is updated at each iteration, the left 2×2 matrix of (6) that needs to be inverted is only updated at each new scale level L. The incremental update of the optical flow ΔV is then calculated as in equation (6). The process is shown in Figure 2.5.

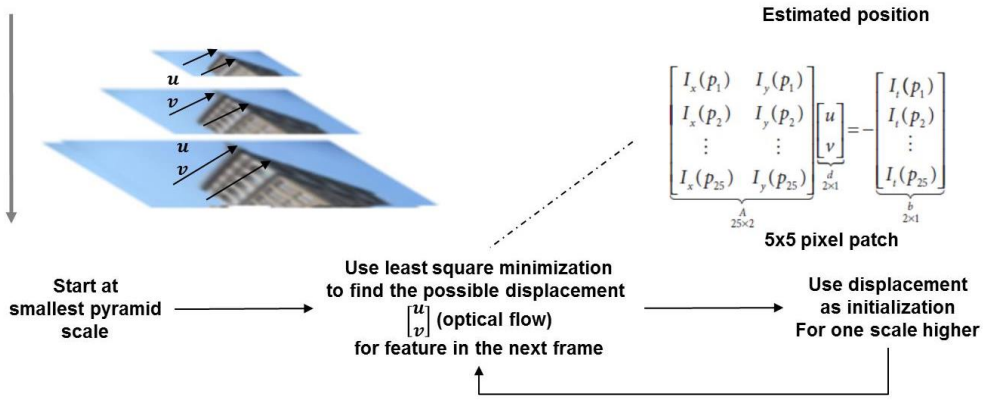


Figure 2.5 Flow chart showing the process to frack a feature using a pyramidal Lucas Kanade optical flow algorithm. It calculates optical flow and uses the scaled up flow vector to obtain an initial estimate of the new location in increasingly higher scales in the new image.

The larger the window size the more accurate it can track (large) motions, but this also comes at a cost of increased computational load.
A shortcoming with optical flow is that it only assumes rigid transformations. In a 3D world pixel displacements from neighbours can vary with depth and rotations.

2.1.2.2 Keyframe tracking

As the camera moves, incremental changes in the tracked patch can cause drift of a feature's location. This can be avoided by tracking features using key-frames.
A key-frame is created when certain conditions have been met, for example when the camera's position has changed significantly at which point new features are detected. This allows a comparison of a feature in the current frame with that in a key-frame which can be seen as comparing it with its former self. This way possible drift can be corrected for.
Because there is a large view-point change between frames the patch from the keyframe needs to be warped so as to align it with the current pose. The process is shown *Figure 2.6*.

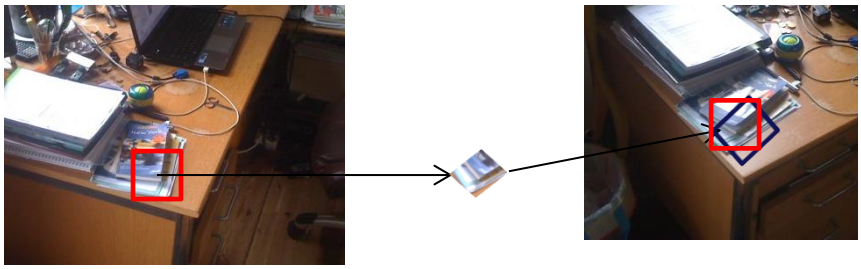


Figure 2.6 The left image shows the key-frame, the image on the right the current frame in which the feature needs to be searched. The large rotation between these frames shows that it becomes difficult if not impossible to match the red squared pixel patches in both frames with each other. Instead the red feature patch from the left image needs to be warped as shown in the middle patch between the arrows to find a good match. The purple square shows how it can be matched in the new frame.

2.1.2.3 Pose estimation

The camera pose can be estimated by finding the transformation $P = [R|T]$ from the 2D image coordinates $\{x_1^c, x_2^c \dots x_N^c\}$ in a frame to their corresponding 3D world coordinates $\{X_1^w, X_2^w \dots X_N^w\}$, as shown in Figure 2.7. This transformation can be found by minimizing the 2D – 3D re-projection error through a least squares method over a selection of N points, so for each i points;

$for\ all\ i\colon\ x_i^c = [R|T]X_i^w$

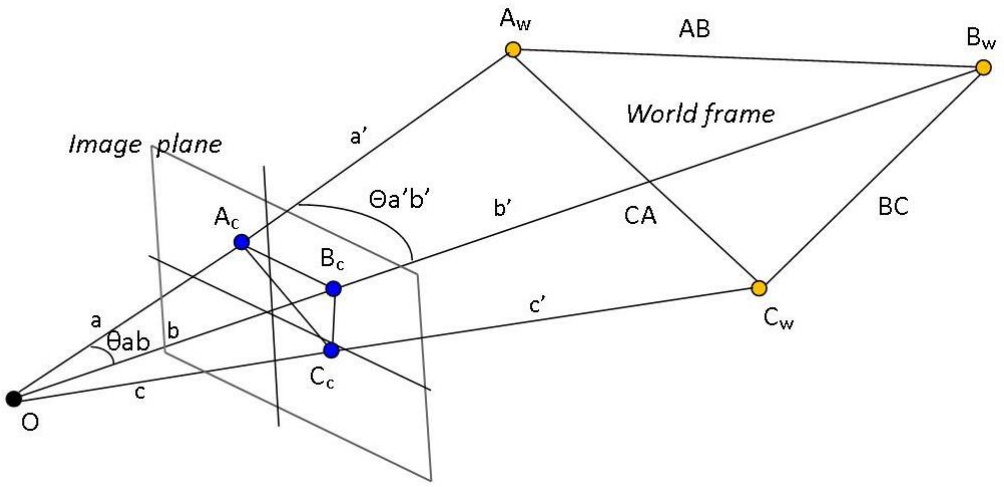


Figure 2.7 The basic idea behind pose estimation. The vectors between the optic center O and the image points (Ac, Bc and Cc) lie collinear to the vectors between the image points and the world points (Aw, Bw and Cw).

A popular method is currently the Perspective-n-Point (PnP) method which was proposed by Fischler and Bolles [17]. The advantage of this method is that it puts a constraint on possible solutions for the pose. In Figure 2.7 it can be seen that the perspective lines from the optic center to the image points a, b and c lie collinear to those between the image points and world points; a', b' and c'. The angle between these lines will therefore be equal, so $\theta_{ab} = \theta_{a'b'}$. Using the cosine rule these constrains can be expressed by (11), (12) and (13).

$$a'^2 + b'^2 - 2a'b'\cos(\theta_{ab}) = AB^2 \quad (11)$$

$$b'^2 + c'^2 - 2b'c'\cos(\theta_{bc}) = BC^2 \quad (12)$$

$$c'^2 + a'^2 - 2c'a'\cos(\theta_{ca}) = CA^2 \quad (13)$$

Since the camera pose is unknown a', b' and c' cannot be known. Rewriting (11), (12) and (13) into polynomial equations and solving for this can return the rotations and translations of the camera in the world frame.

Various methods have been developed to solve for the rotations and translations of the camera. One such method that has proven to be both efficient and effective compared to some of the more complex

PnP methods is the Efficient Perspective-n-Point (ePnP) algorithm [18]. It uses imaginary reference points $\boldsymbol{c}_j$ representing the camera coordinates $\boldsymbol{p}_i^c$ through a weighted sum. The transformation from camera coordinates to image points $w_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}$ is written in (14), where α_{ij} represent the homogenous barycentric coordinates. This can be rewritten into a linear system that can be solved using the SVD to obtain the unknown control points $\boldsymbol{c}_j$. Because this might return different solutions (multiple small Eigenvalues), the solution with the smallest re-projection error is chosen.

$$w_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \boldsymbol{A}\boldsymbol{p}_i^c = \boldsymbol{A} \sum_{j=1}^{4} \alpha_{ij}\, \boldsymbol{c}_j = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^{4} \alpha_{ij} \begin{bmatrix} x_j^c \\ y_j^c \\ x_j^c \end{bmatrix} \qquad (14)$$

2.1.2.4 Pose estimation using M-estimation

STAM and PTAM both use a robust non-linear optimization method to estimate the pose. This is because outliers can emerge during feature tracking which violate the Gaussian noise assumption of a least squares method. This skews the result of such a method thereby reducing its accuracy.

M-estimators can use a cost function $\rho(r_i)$ (15) that is continuous, symmetric and monotonically increasing. This gives the possibility to bind the influence of large errors and outliers or completely eliminate them, as shown in Figure 2.9. While the residual of measurements with small errors that correspond to the Gaussian noise principal are included in the minimization process.
In the cost function (15), r_i is the residual error; the difference between the i^{th} observation and its fitted value, the parameter vector is represented by $p = [p_1,\ p_2 \dots p_n]$, in the case of pose estimation this represents the pose.

$$\sum_{i=1}^{n} \rho\,(r_i) \qquad (15)$$

The solution can be found by taking its derivative and finding the minimum of (16).

$$\sum_{i=1}^{n} \frac{d\rho(x)}{dx} \frac{\partial r_i}{\partial p_j} = 0, \qquad for\ j = [1 \dots n] \qquad (16)$$

Introducing a weight function $w(x) = \frac{d\rho(x)}{dx} \frac{1}{x}$ the equation becomes (17). Figure 2.8 shows several types of such functions.

$$\sum_{i=1}^{n} w(r_i) r_i \frac{\partial r_i}{\partial p_j} = 0\,, \qquad for\ j = [1 \dots n\,] \qquad (17)$$

This can then be solved using a Gauss Newton method. There are different types of cost functions for $\rho(r_i)$. In [6] the Tukey's biweight was selected (18) because the cost function levels off and the weight function goes to zero after some residual error $|r| > a$, where a is a predefined constant.

$$\rho(r) \quad \begin{cases} \frac{a^2}{6}\left[1-\left(1-\left(\frac{r}{a}\right)^2\right)^3\right] & |r| \leq a \\ \frac{a^2}{6} & |r| > a \end{cases} \qquad (18)$$

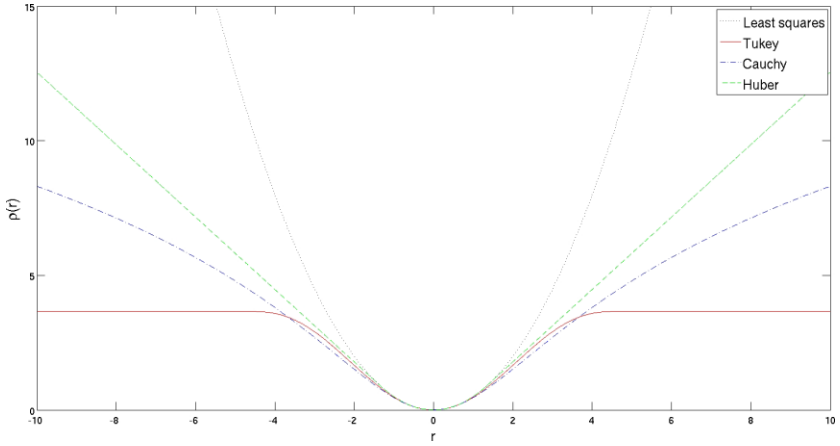


Figure 2.8 Different cost functions $\boldsymbol{\rho(r)}$ showing their response to a residual. Tukey levels off after the residual becomes larger than some set constant $\boldsymbol{a}$ which in this case is set to 4.6851, for example in the case of outliers. This figure is taken from [6].

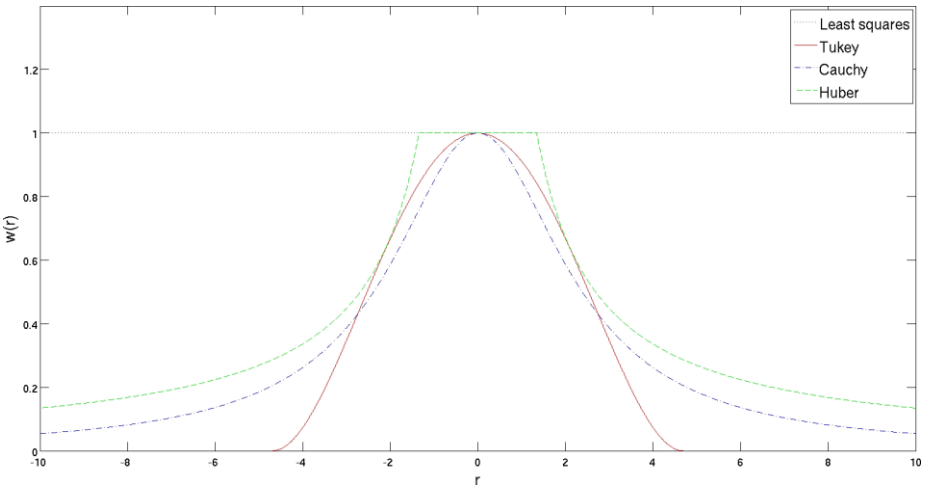


Figure 2.9 from [6]. It shows several weight functions that can be used with the M-estimation method.

The pose estimation can then be solved iteratively by updating point $\boldsymbol{p}$ till it converges to a solution;

$$\boldsymbol{p}_{t+1} = \boldsymbol{p}_t - (J^T W J)^{-1} J^T W b \qquad (19)$$

For pose estimation the residual error r_i will be the re-projection error of the 3D map point onto the image in which the feature is being tracked.
J is the Jacobian of the re-projection error evaluated at point $\boldsymbol{p}$ and W is a diagonal matrix $diag(w_1, w_2 \dots w_n)$ corresponding to the weights.

2.2 Stereopsis

In this research a stereo camera will be used to calculate the 3D location of a world point. With monocular vision the 3D location has to be obtained through triangulation; solving the least square error for corresponding image coordinates over two or more camera poses. The accuracy of this estimation, however, depends also on the accuracy of the pose estimation process.
With a stereo setup on the other hand the 3D coordinates can be calculated algebraically, as shown in (23), (24) and (25), using the left and right (rectified) camera frame whose relative pose can be accurately calibrated offline. This location will also be known to scale since a real world reference, the baseline, is known.

Figure 2.10 shows the geometry of such a stereo setup which is also known as the epipolar geometry [19] with c_l and c_r being the left and right optic centers. This epipolar geometry indicates how the epipolar plane, identified by P, c_l and c_r, intersects the left and right images at a line in either image called the epipolar line; $\boldsymbol{E}_l$ and $\boldsymbol{E}_r$ for the left and right image respectively. Point P is projected into the left and right images at the intersection of lines p_l and p_r. If the projection of P onto the left image is known its matching location in the right image will lie on $\boldsymbol{E}_r$, also known as the epipolar constraint. This reduces the stereo matching from a 2D to a 3D problem.

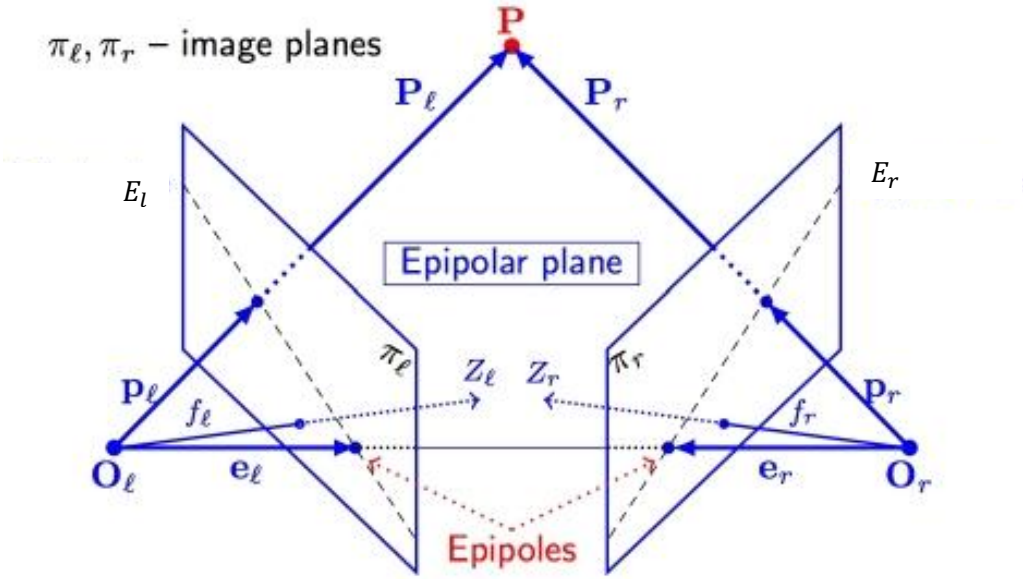


Figure 2.10 The epipolar geometry.

The relation between a world point P at $\{X_w, Y_w, Z_w\}$ observed in the left and right camera frame at image coordinates x_l and x_r respectively can be written as;

$$x_r = R(x_l - T) \qquad (20)$$

Since a point can be viewed from both cameras, it can be thought to reside on a plane lying tangent to both image planes. So for the left image plane the co-planarity condition can be written as;

$$(x_l - T)^T T \times x_l = 0 \qquad (21)$$

Substituting (21) into equation (20) gives;

$$(R^T\ x_r)T^T \times x_l = 0 \qquad (\ 22\ )$$

After some rewriting (22) can be solved in a least squared error equation to obtain the intrinsic and extrinsic parameters using multiple (generally a minimum of eight) corresponding points from the stereo images.

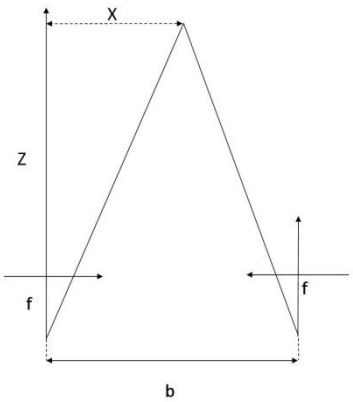


Figure 2.11 A world point observed by a left and right camera into rectified images.

Looking at Figure 2.11 it can be easily worked out how to calculate the 3D coordinates $[X, Y, Z]$ from the rectified left $[x_l, y_l]$ and right image $[x_r, y_r]$ coordinates using the camera parameters. These parameters are the focal length f, optical centers c_x and c_y and baseline b. These have to be determined in advance through camera calibration.

$$X = \frac{(x_l - c_x)b}{x_l + x_r} \qquad (\ 23\ )$$

$$Y = \frac{(y_l - c_y)b}{x_l + x_r} \qquad (\ 24\ )$$

$$Z = \frac{fb}{x_l + x_r} \qquad (\ 25\ )$$

2.3 Parallel tracking and mapping (PTAM)

The novelty of this method is to make use of parallel processers that have become commonplace in modern computers. This allows having the tracking and mapping processes to run in parallel threads and gives PTAM better real-time performance [1]. Figure 2.14 gives an overview of how this process works.

To make tracking and pose estimation more robust it uses keyframe tracking, which is explained in section 2.1.2.2. Features are first tracked using the optical flow after which the pose is estimated using an M-estimator, as explained in 2.1.2.4. The tracked coordinates are improved using key-frame tracking and can then be used to refine the pose with an M-estimator.

Because it uses a monocular camera setup it needs to find a baseline first, meaning; the user needs to translate the camera roughly 10 centimeters for the system to initialize a map and find a plane on which it can project virtual objects.

While the tracking process is running, the map points are continually being updated by local or global bundle adjustment. Local bundle adjustment is done over the five most recent key-frames to avoid exponentially increasing complexity with each new keyframe.

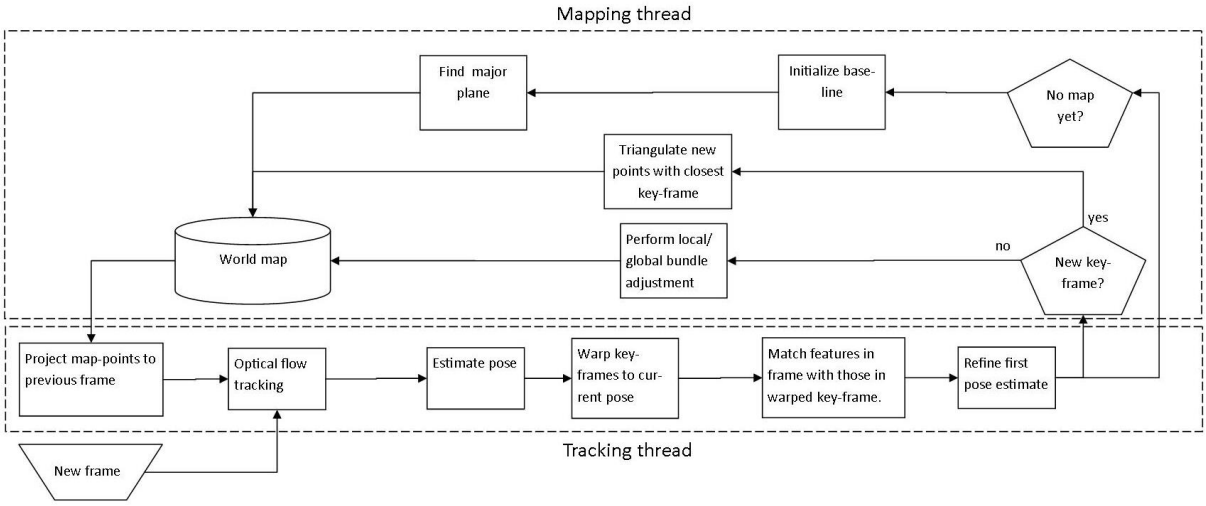


Figure 2.12 PTAM parallel tracking and mapping frame work. The two separate tracking and mapping threads are bounded by different dashed rectangles.

PTAM has proven to outperform EKF SLAM as shown in Figure 2.13 where both their performances are tested on synthetic data. It can be clearly seen that PTAM stays a lot closer to the ground truth compared to the EKF-SLAM method.

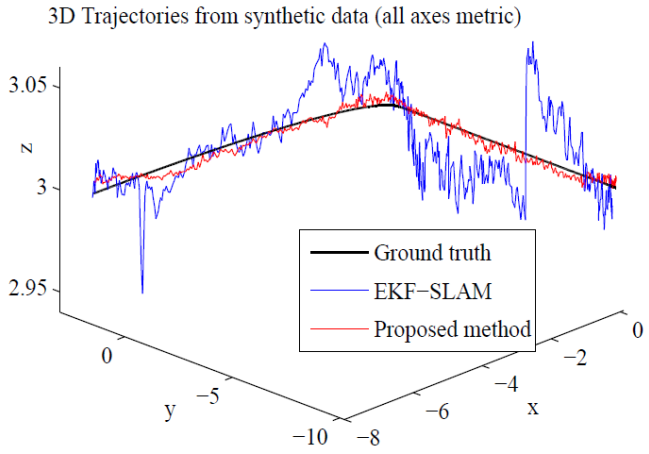


Figure 2.13 taken from [1]. Performance comparison of PTAM (the proposed method) and an EKF-SLAM method.

2.4 AR framework with STAM

STAM [6] is an extension of PTAM [1] that uses a stereo setup in favor of a monocular setup. This way it doesn't need to initialize a baseline and all coordinates can be represented in a real world scale.
It also uses the features tracked from the second camera in the pose estimation, which is done using an M-estimator, this method is further explained in section 2.1.2.4.

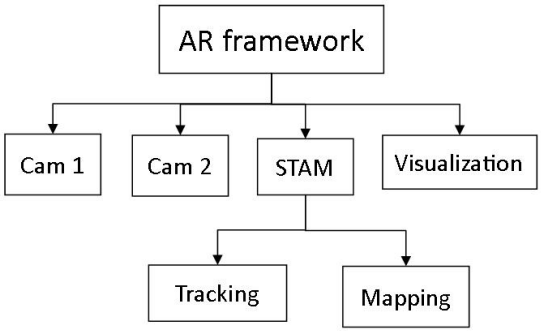


Figure 2.14 An overview of the AR framework. It has 5 threads, one for each camera, two for the tracking and mapping and one for the visualization part.

Next to adapting PTAM to a stereo setup other improvements have also been implemented to increase performance and robustness;

- Instead of detecting Features from an Accelerated Segment Test (FAST) STAM uses an Adaptive and Generic Accelerated Segment Test (AGAST) [20] detector because it is more adaptable to various scenes while being equally efficient in detection. It uses the same corner criterion as FAST but can adapt the detector dynamically during the process. AGAST's adaptability is due to its training on different environments (indoors and outdoors) while FAST has only been optimized for one.
- An extra test to find outliers is done by comparing the displacement of points that are connected through a Delaunay [21] mesh. If a point does not have at least two neighbours that show similar motion within 5 pixels difference, it is marked as an outlier.
- To improve the robustness of the estimated pose even further the ePnP [22] method inside a RANSAC routine is used to refine the pose after the first stage. Because the computational complexity for the ePnP algorithm grows linearly it can be used effectively on a small number of points inside RANSAC.

2.5 Fast Semi-Direct Monocular Visual Odometry (SVO)

Semi-Direct Monocular Visual Odometry (SVO) [23] is an efficient method for pose estimation that can run at 55Hz and has proven to outperform PTAM in terms of accuracy, as shown in Figure 2.15.

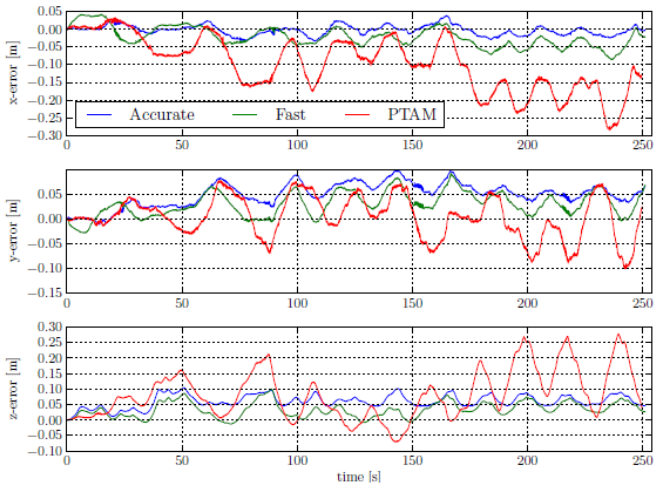


Figure 2.15 Accuracy of pose estimated by SVO compared by PTAM for the x, y and z translations. The accurate and fast are the two different modes SVO can operate in.

It consists of a motion estimation and mapping part that both run in different threads, similar to [1]. The camera motion estimation is done in three stages, as shown in Figure 2.16. The three different stages will be explained next.

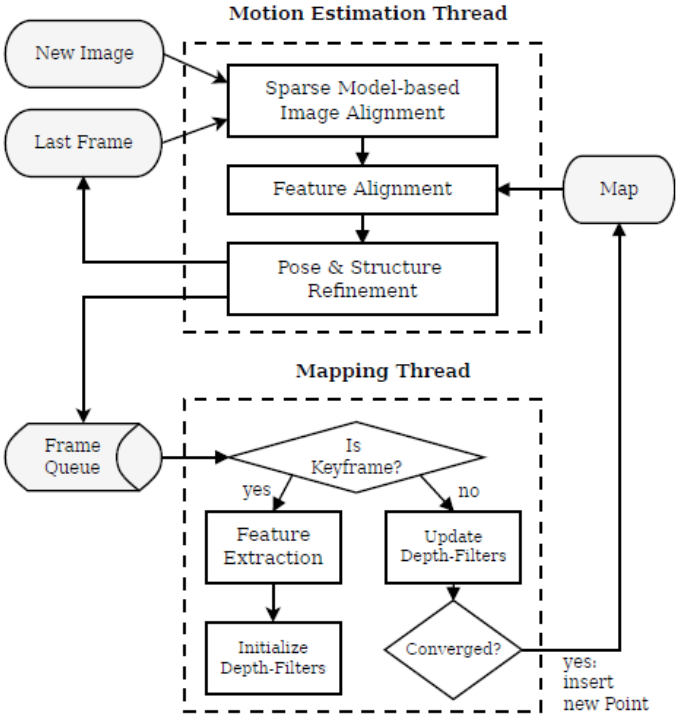


Figure 2.16 Taken from [23]. SVO works by running the motion estimation and mapping process in different threads. The motion estimation consist of three steps to refine the motion estimation. The estimated motion is then used to update the map.

Motion:

1) First the photometric error between the 4x4 feature patches from the previous image and the new image are minimized with a Gauss-Newton procedure.
2) Second, to reduce drift, the result from the first stage is used as an initial guess to project the 3D map points in the current frame. The features of those re-projected points that are visible in the current frame are compared to their respective feature in the key-frame that has the closest viewing angle compared to the current one. This error is minimized by affine warping using the inverse compositional Lucas-Kanade algorithm [24]. This is a relaxation step that violates the epipolar constraints to achieve higher correlation between feature patches.
3) The last step optimizes the pose to minimize the re-projection residuals, and is solved using an iterative non-linear least squares method, also known as motion only bundle adjustment [25]. The 3D points can also be optimized through structure only bundle adjustment, this is however omitted in the fast parameter settings of the algorithm.

Mapping:

A depth filter is used in the triangulation of 3D coordinates for each point. The advantage of this is that it has fewer outliers because the filter makes many measurements until it converges. Depth even converges in highly similar environments because erroneous measurements are explicitly modeled by a Gaussian and Uniform mixture model distribution [26].

3 Motion segmentation

This chapter will introduce the method that is developed during this research to segment moving parts from the scene. First some of the previous work in the field of visual motion segmentation that has been examined during a literature research will be discussed.

In motion segmentation features are separated according to their motion patterns. As in any segmentation method the result depends both on the segmentation method used as well as how the various similarities within a population are modeled and compared.

3.1 Previous work

Quite some research has been done in the field of feature based motion segmentation, most of which with (autonomous) automotive applications in mind, since it is important that such a system has awareness of other road users in order to be able to avoid accidents.

To distinguish between dynamic and static (background) points [27] assumes motion constraints on dynamic objects like independent motion, constant heading and the planarity constraint (objects residing on the ground plane). In [28] objects that are part of the environment's background (trees, billboards etc.) are assumed to be relatively large and/or are also expected to violate the planarity constraint. Features from tree branches that have been clustered without the trunk for example might appear to the system to float in the air, while clustered features from a car should be close to the ground.

The velocity of a point is calculated over up to six consecutive images using backward differences with weighted coefficients; more recent measurements will be weighed more heavily. These are then clustered according to their Mahalanobis distance to account for noise and errors that are expected to increase with the distance from the camera. Due to the correction on the increasing noise level in the depth direction, the detection of cars and pedestrians at close and far range of up to 60m worked well for this method in a rural environment. In a city environment this method worked well for up to 50m. Turning cars were properly detected due to the continuous velocity assumption, though it failed for sharply turning ones. Also pedestrians and bicyclists aren't always detected in their entirety. This method, however, mainly segments translations, which is fine for vehicles describing a translating trajectory. But people wearing AR helmets will rotate their heads resulting in rotations that this method does not segment well.

A motion segmentation method that uses multi-body segmentation is described in [29]. Individual moving bodies are detected and segmented according to their individual affine rigid body motion models using a robust algebraic segmentation (RAS) method [30]. RANSAC is used with an eight-point algorithm to find the fundamental matrix in order to detect and reject outliers within each group. The use of efficient geometric constraints and a probability framework which propagates the uncertainty in the system allows for extra robustness and capability to segment complex motions. It appears that a pose estimator in conjunction with RANSAC is a popular method to create multiple motion hypotheses. A 3 point linear algorithm in conjunction with RANSAC is also used in [9] to create motion hypotheses from disparity space. The largest consensus group, which is considered to be the static background, is then used to estimate the camera motion using a Levenberg-Marquardt algorithm. Tracking is further improved by using blob extraction which creates macro blobs that grow on a regional disparity similarity assumption.

Motion segmentation can also be done using the Euclidean distance to determine the connectedness measure [31] of pixels in a point cloud. Building on the local convexity criterion, based on the idea that objects generally have a convex outline, every border between objects will have a concave outline and thus will not be grouped.
Motions are modeled using a probabilistic framework. Different objects that show similar motion models will be grouped and those that have unique motions will create new tracks. Although there is some measure of over-segmentations due to errors, this method allows for good motion estimation even in highly dynamic scenes. An interesting improvement that can help overcome over-segmentation is from [32] where edge scoring is also used in the clustering process so that objects are better separated. Motion detection from row vectors relies on good background motion assumptions. And since no robust statistics are needed it will not suffer from outliers that can distort the optimization algorithms for pose estimation. Its effectiveness does however depend on the texture richness of dynamic objects. For example people's clothing can have very few detectable features. It also takes several frames to detect and track objects to obtain a reliable row vector, hence segmentation is not instantaneous.

3.2 Method chosen from literature

The method we consider to be most effective for our AR framework is from [33] and will be explained here separately. This method is basically a RANSAC method where the best data set is chosen depending on the spatial distribution of the image coordinates and the corresponding inlier ratios.
Its motion segmentation involves the following steps;

1) Put the feature points in bins, these can be of size 10×10 pixels for example. The reason for this is to avoid densely populated areas that might skew the scoring result (26) which depends on position covariance (27); densely packed inliers can result in a low covariance for example.
2) A sample set is taken from the bins at random. The probability that a sample is picked from a bin depends on its inlier ratio $\varepsilon_i = \frac{\sum inliers}{N}$ which is truncated into the range of [0.2 – 1] for $\tilde{\varepsilon}_i$.
   - The probability that a point is selected from a bin is $p_i = \frac{\tilde{\varepsilon}_i}{\sum_j \tilde{\varepsilon}_j}$
3) First only one point per bin is tested on the new hypotheses.
   - If there are enough inliers it will recalculate a new hypotheses over all the inliers and test it on all the points.
4) The score of the hypotheses is calculated by;

$$s = \sum_i \varepsilon_i \frac{\pi\sqrt{\det(C)}}{A} \qquad (26)$$

The covariance C is calculated by (27), where x_i is the centre of the bin B_i and $\overline{x}$ the weighed mean position is calculated using equation (28).

$$C = \frac{\sum_i \varepsilon_i}{(\sum_i \varepsilon_i)^2 - \sum_i {\varepsilon_i}^2} \sum_i (x_i - \overline{x})(x_i - \overline{x})^T \qquad (27)$$

$$\overline{x} = \frac{1}{\sum_i \varepsilon_i} \sum_i \varepsilon_i x_i \qquad (28)$$

5) The stopping criterion is fulfilled if; $(1-(\sum_i p_i\,\varepsilon_i)^m)^{K_s} < \eta$, where m is the initial sample size, K_s the number of iterations done and η an experimentally set constant.
6) Create a probability map which is used for the selection of a new sample in the next frame. Points that lie in a bucket that contained more inlier points in the previous frame have a higher probability to be selected for estimating an initial model. A probability map is visualized in Figure 3.1.

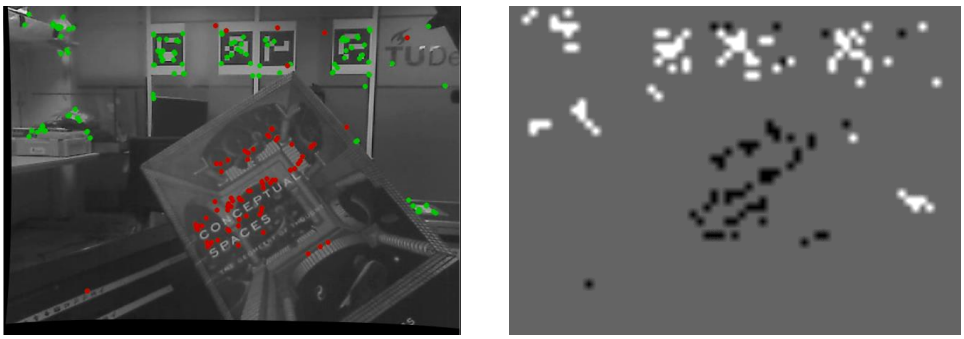


Figure 3.1 The probility map (right) obtained from the identification of inliers and outliers (left). The inliers (green)in the left image result in bins more likely to be picked in the next frame (white).

3.3 Proposed motion segmentation algorithm

The proposed motion segmentation method to be implemented into STAM is based on the algorithm explained in 3.2. It will be adapted to work with a stereo camera setup, as is shown in Figure 3.2. So consequently the motion model can be estimated using stereo matched features, as explained in 3.3.3. The benefit of using a stereo setup over a monocular setup is that for each feature its 3D coordinates can be estimated individually in camera coordinates using equations (23), (24) and (25). This means that fewer points (≥ 3) are needed to create a camera motion hypothesis with 6 degrees of freedom (DOF). A monocular setup on the other hand needs at least eight (in case of the eight point algorithm). The fewer points are needed to initialize a motion hypothesis, the smaller the chance that this initial sample contains outliers; the better the sample the quicker a good motion model with corresponding inliers is found.

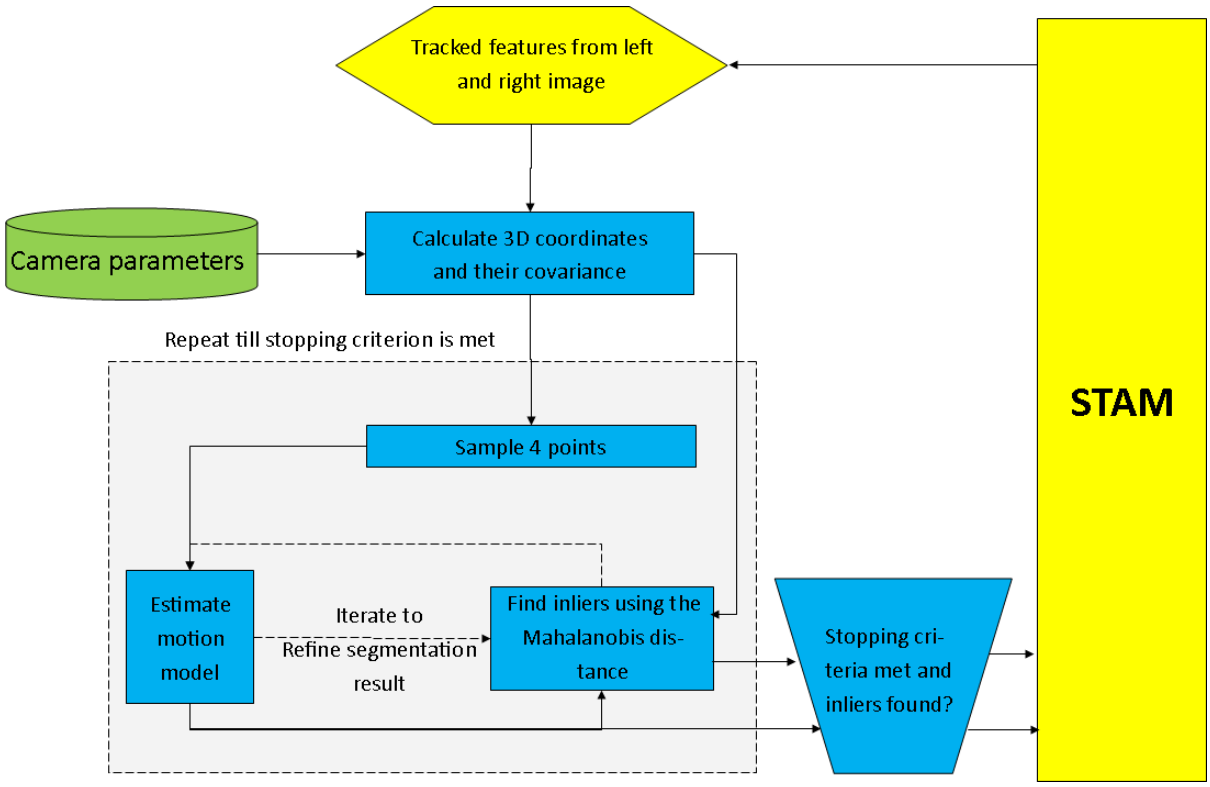


Figure 3.2 The process to filter out independently moving features using the stereo tracked coordinates. It consists of two main steps, the first calculates the data needed for motion segmentation. The second is the actual motion segmentation process that filters out the independent motions. For outlier detection it also needs to estimate the camera motion which can be used to update the camera pose in STAM, which can thereby skip the Gauss-Newton process that would normally do this update in the first pose tracking stage.

The original paper [33] didn't exactly specify what method was used to create a model by which to segment dynamic features. Quite possibly this could have been a pose estimation method such as PnP, which only needs 3 points for a monocular setup, as explained in section 2.1.2.3. PnP however also requires corresponding 3D world coordinates for those points, making it reliant on the quality of the mapping process.

Measurement uncertainty will be taken into account to improve static feature detection from the 3D camera coordinates. While features are tracked their coordinates are perturbed by a certain amount of noise [34]. Points lying further from the camera cannot be described as accurate as those lying closer by because the resolution deteriorates with distance with the effect that the accuracy of the 3D location also decreases.

Incorporating the uncertainty of the 3D locations into the testing of the motion model on each point can be done by using the Mahalanobis distance $\Delta(X_1, T(X_2))$ (29), which will be further explained in 3.3.1. $T(X_2)$ is the transformation of the 3D coordinate X_2 in the second frame so it matches X_1 in the first frame using the motion model. Ω is selected for the notation of the covariance over its more conventional sign Σ to avoid confusion with the summation signs in this thesis.

$$\Delta(X_1, T(X_2)) = \sqrt{(X_1 - T(X_2))\Omega^{-1}(X_1 - T(X_2))^T} \qquad (29)$$

Normally to check the fitness of the motion model, the sum of squared errors is taken between the measured and estimated position through a transformation using the motion model. But the squared error should be transformed using the covariance matrix to correct for the distribution of measurement errors.

This is also done in a similar way in [28] in order to compare the motion of individual points with each other and cluster them if the Mahalanobis distance is within some error margin. Another example that is closer to our use for hypotheses testing using the Mahalanobis distance is [35], which includes the log of the determinant of the covariance in order to regularize the Mahalanobis distance (30). This way data points are not penalized when the inverse of their covariance is relatively well aligned with the error $X_1 - T(X_2)$.

$$\Delta(X_1, T(X_2)) = \sqrt{(X_1 - T(X_2))\Omega^{-1}(X_1 - T(X_2))^T + \log(\det(\Omega))} \qquad (30)$$

3.3.1 The Mahalanobis distance

One can take the Euclidean distance from a point to the mean of a data set to test how well this point fits that data set. Or, when the data is normally distributed one can find out how many standard deviations the point is removed from the set's mean. Since the population percentage for each standard deviation is known a sense of a standardized reference frame can be employed. This however requires the data to be univariate so the variables can be normally distributed.

The Mahalanobis distance [36] is a generalization of this idea of a standardized reference frame for multivariate data, where the distribution is represented using a covariance.
Mahalanobis compared his theory to that of Galilean transformations which maps all coordinates into a frame of reference and thus ensures that apples are compared to apples, using a common expression. This makes complex problems a lot more trivial because they can be solved within a frame of reference. With the Mahalanobis distance the standard normal distribution is the frame of reference [37].

To transform a random variable $x \sim (\Omega, \mu)$, with covariance Ω and mean μ, into a standard normally distributed random variable r' it has to be transformed with the transformation $T(x)$.

$$T(x) = \Omega^{-1/2}(x - \mu) \qquad (31)$$

The Mahalanobis distance is the Euclidean norm of the transformation in (31). To compare two data points with the same covariance for example this becomes;

$$\Delta(x, y) = ||T(x) - T(y)|| \qquad (32)$$

When writing out (32) using (31) μ cancels out and gives equation (33).

$$\Delta(x, y) = \sqrt{(x - y)\Omega^{-1}(x - y)^T} \qquad (33)$$

Figure 3.3 shows the difference in analyzing data using the Euclidean and Mahalanobis distance. It can be seen that along the second principal axis y' of the data the Mahalanobis distance is a lot stricter because the variance along that axis is also smaller, while it is more generous along the primary principal axis x'. These vectors can be found from the eigenvectors of the covariance, the eigenvalues then represent a sense of scale.

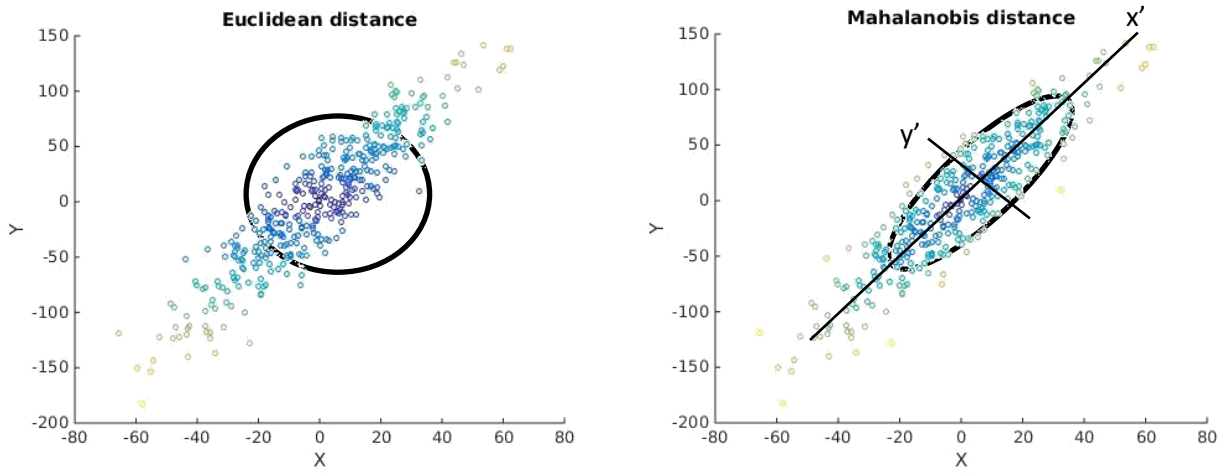


Figure 3.3 The difference between the Euclidean distance (left) and the Mahalanobis distance (right). Both images display exactly the same dataset. The color represents how well each individual data point fits in the set (dark blue is closer as light green).

Using the Mahalanobis distance the dissimilarity measure between two vectors with similar covariance can thus be calculated while taking their distribution into account. For the segmentation method in this research the dissimilarity between a point's estimated position and its measured position needs to be compared. The covariance is employed to adjust for the error distribution of each 3D position.

3.3.2 First order error propagation, or why we need to calculate the Jacobian to obtain the covariance.

To calculate the Mahalanobis distance the covariance needs to be calculated. This is done using equation (34), also known as the error propagation law, where J is the Jacobian of the input signals and S the matrix holding their variances. To better understand the reasoning behind this formula this section, which is a summery from [38], will explain a bit more about why the covariance Ω is calculated according to equation (34).

$$\Omega = J\,S\,J^T \qquad (34)$$

The covariance can be defined as the measure by which the change of one variable influences the change of another variable. Or; the degree to which two sets of variables deviate from their expected values. An intuitive explanation for the use of the Jacobian J could therefore be because J also describes the (local) behavior of the system caused by its variables.

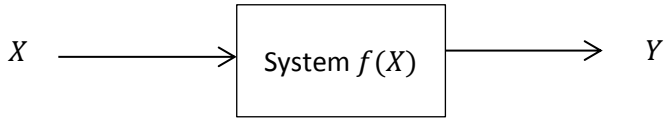


Figure 3.4 A single input single output system $f(X)$

Figure 3.4 shows a system with a single input X and output Y. Figure 3.5 shows how the normal distributions of X can be projected onto the output Y through the linearization of the system $f(X)$. If the distribution on the X axis would be projected onto the Y axis through the non-linear function $f(X)$ itself, the y axis distribution would be asymmetric and thus no longer be Gaussian.

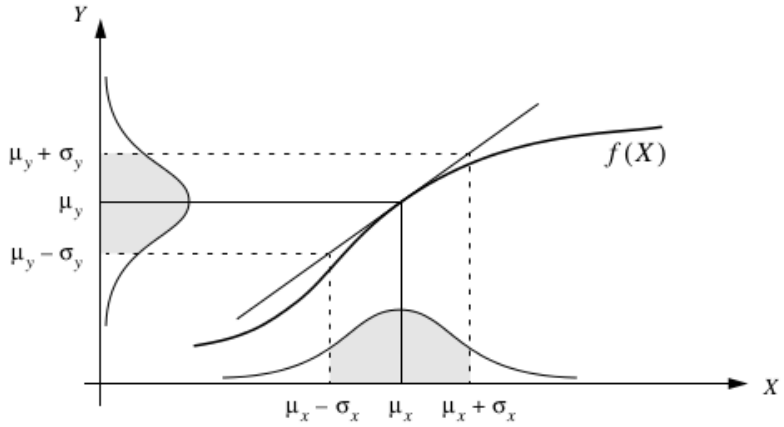


Figure 3.5 One dimensional case of a non-linear error propagation problem through the system $\boldsymbol{f}(\boldsymbol{X})$. The distribution for $\boldsymbol{Y}$ is obtained through the linearization of $\boldsymbol{f}(\boldsymbol{X})$ around the mean of $\boldsymbol{X}$.

The linear relationship can be obtained by describing $f(X)$ by a first order Taylor expansion around the mean of X; μ_x.

$$Y \approx f(\mu_x) + \frac{\partial f}{\partial X}_{x=\mu_x} \cdot (X - \mu_x) \quad (35)$$

The mean and standard deviation of Y, μ_y and σ_y respectively, can be described as a function of X.

$$\mu_y \approx f(\mu_x) \quad (36)$$

$$\sigma_y = \frac{\partial f}{\partial X}\ \sigma_x\ = \frac{\partial f}{\partial X}_{x=\mu_x} \cdot (X - \mu_x) \quad (37)$$

$$E[Y] = \mu_Y \quad (38)$$

This reasoning can be expanded to a more complex system with multiple inputs and one or more outputs as shown in Figure 3.6.

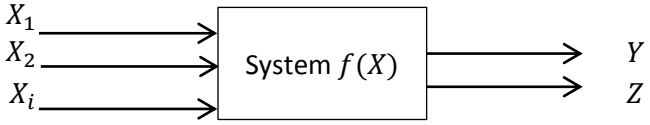


Figure 3.6 A system with multiple inputs and multiple outputs.

The output for a system with multiple inputs can be written using a first order Taylor expansion as in (35) for both Y and Z;

$$Y \approx f_1(\mu_i, \mu_i, \dots, \mu_n) + \sum_{i=1}^{n} \left[\frac{\partial f_1}{\partial X_i} (\mu_i, \mu_i, \dots, \mu_n) \right] \cdot (X_i - \mu_i) = \mu_Y + \sum \frac{\partial f_1}{\partial X_i} [X_i - \mu_i] \quad (39)$$

$$Z \approx f_2(\mu_i, \mu_i, \dots, \mu_n) + \sum_{i=1}^{n} \left[\frac{\partial f_2}{\partial X_i} (\mu_i, \mu_i, \dots, \mu_n) \right] \cdot (X_i - \mu_i) = \mu_z + \sum \frac{\partial f_2}{\partial X_i} [X_i - \mu_i] \quad (40)$$

The distribution for Y, σ_Y^2, is shown in (41), the distribution for Z, σ_z^2, can be done similarly.

$$\sigma_Y^2 = E[(Y - \mu_Y)^2] = E\left[\frac{\partial f_1}{\partial X_i} (\mu_i, \mu_i, \dots, \mu_n) \cdot (X_i - \mu_Y)^2 \right] \quad (41)$$

In the case of a system with multiple inputs and multiple outputs Y and Z, their relationship between their variance, the covariance σ_{YZ} is;

$$\begin{aligned} \sigma_{YZ} &= E[(Y - \mu_Y)(Z - \mu_Z)] \\ &= E\left[\left(\mu_Y + \sum \frac{\partial f_1}{\partial X_i} [X_i - \mu_i] \right) \cdot \left(\mu_z + \sum \frac{\partial f_2}{\partial X_i} [X_i - \mu_i] \right) \right] - \mu_y \mu_z \end{aligned} \quad (42)$$

Similar to how the distribution for Y from (35) can be described in (37), so can the covariance for (41) and (42) be rewritten into (43) and (44). The derivation to come to (43) and (44) is quite lengthy so the reader is advised to read [38] for a more complete explanation. But it boils down to writing out the variance in expected values (38) using the variable and mean over multi-input multi-output systems.

After rewriting (41) and (42) μ_Y and μ_Z cancel out in both equations. So after a lengthy derivation these equations can be written as

$$\sigma_{YZ} = \sum \frac{\partial f_1}{\partial X_i} \frac{\partial f_2}{\partial X_i} \sigma_i^2 + \sum \sum_{i \neq j} \frac{\partial f_1}{\partial X_i} \frac{\partial f_2}{\partial X_j} \sigma_{ij}^2 \quad (43)$$

$$\sigma_Y^2 = \sum \left(\frac{\partial f_1}{\partial X_i} \right)^2 \sigma_i^2 + \sum \sum_{i \neq j} \frac{\partial f_1}{\partial X_i} \frac{\partial f_2}{\partial X_j} \sigma_{ij}^2 \quad (44)$$

Let's now write out the structure of the equation for the covariance in matrix form as

$$J\,S\,J^T = \begin{bmatrix} \sigma_{Y_1}^2 & \sigma_{Y_1 Y_2} \\ \sigma_{Y_2 Y_1} & \sigma_{Y_2}^2 \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial X_1} & \cdots & \frac{\partial f_1}{\partial X_n} \\ \frac{\partial f_2}{\partial X_1} & \cdots & \frac{\partial f_2}{\partial X_n} \end{bmatrix} \begin{bmatrix} \sigma_{X_1}^2 & \sigma_{X_1 X_2} & \cdots & \sigma_{X_1 X_n} \\ \sigma_{X_1 X_2} & \sigma_{X_2}^2 & \cdots & \sigma_{X_2 X_n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{X_n X_1} & \sigma_{X_n X_2} & \cdots & \sigma_{X_n}^2 \end{bmatrix} \begin{bmatrix} \frac{\partial f_1}{\partial X_1} & \frac{\partial f_2}{\partial X_1} \\ \vdots & \vdots \\ \frac{\partial f_1}{\partial X_n} & \frac{\partial f_2}{\partial X_n} \end{bmatrix} \quad (45)$$

The comparison can be seen when writing out an entry from the matrix $J\,S\,J^T$. For each diagonal entry the resulting equation will look like (44), while those off-diagonal will look like (43). Again for the extended proof and comparison the reader is referred to [38].

3.3.2.1 Modeling the noise

The noise for the 3D position obtained over the stereo images can be modeled by taking the partial derivative over (23), (24) and (25) to obtain the Jacobean for (34). During this research we initially looked at [28] as an example for how to construct the Jacobean. In that paper the partial derivative of the 3D coordinate vector $[X\ Y\ Z]^T$ is taken over the left image coordinate along the x axis, u_l and the y axis v_l and also the disparity $d = u_l - u_r$ to obtain (46), where b is the baseline and a_i the weight for frame i.

$$J = \frac{\partial [X\ Y\ Z]^T}{\partial u_l \partial v_l \partial d} = \mathrm{a}_i \cdot \begin{bmatrix} \frac{b}{d_i} & 0 & \frac{(u_{l_i} - c_u)\cdot b}{d_i^2} \\ 0 & \frac{b}{d} & \frac{(v_{l_i} - c_v)\cdot b}{d_i^2} \\ 0 & 0 & \frac{f\cdot b}{d_i^2} \end{bmatrix}, \qquad i = 0 \ldots 5 \qquad (46)$$

The covariance of this method is used for the clustering of the weighted position difference over 6 frames, which returns a 3×18 Jacobian matrix. Here the difference is weighed using weights a_i. Because this method clusters motion vectors with 3 degrees of freedom (DOF) it might not be very applicable to a motion segmentation method based on a 6 DOF motion model.
When this 3D position covariance using the Jacobean from (46) was used in our motion segmentation method it did perform quite well. Even though (46) was meant to be used to cluster 3DOF motions, while we cluster 6DOF motions.

Alternatively the 3D position $[X\ Y\ Z]^T$ can be differentiated over the 4 image coordinates from the left and right frame; $u_l,\ v_l,\ u_r\ and\ v_r$ [34]. This means that the disparity is differentiated over its two variables as opposed to be considered a variable upon itself.

$$J = \frac{\partial [X\ Y\ Z]^T}{\partial u_l \partial v_l \partial u_r \partial v_r} = \begin{bmatrix} \frac{b}{(u_l - u_r)} & -\frac{(u_l - c_u)\cdot b}{(u_l - u_r)^2} & 0 & \frac{(u_l - c_u)\cdot b}{(u_l - u_r)^2} & 0 \\ \frac{(v_l - c_v)\cdot b}{(u_l - u_r)^2} & \frac{b}{(u_l - u_r)} & 0 & \frac{(v_l - c_v)\cdot b}{(u_l - u_r)^2} & 0 \\ -\frac{f\cdot b}{(u_l - u_r)^2} & 0 & 0 & \frac{f\cdot b}{(u_l - u_r)^2} & 0 \end{bmatrix} \qquad (47)$$

It can be seen in both (46) and (47) that the variance is determined by the distance from the optic center, but more so by the disparity; the larger the disparity, the closer a point is to the camera, the smaller the error distribution. The same goes for points detected closer to the optic center, where measurements are also more accurate.
It can be argued that since the disparity is a function of u_l and u_r, it seems more logical to differentiate over the four image coordinates as in (47).

3.3.3 The RANSAC algorithm

This is probably the most popular method to detect outliers in a dataset. It was developed by Fishler and Bowels, who in that same paper [17] also presented the PnP algorithm, section 2.1.2.3.

Figure 3.7 shows a flow chart of the method. It works by initially picking a small sample and then testing that sample over the entire population. Data that fits into the hypothesis within some threshold is considered an inlier. Some predetermined measure or function can then determine the hypothesis that is the most successful, along with the corresponding inliers. It can generally work robustly in a data set that contains up to 50% outliers.

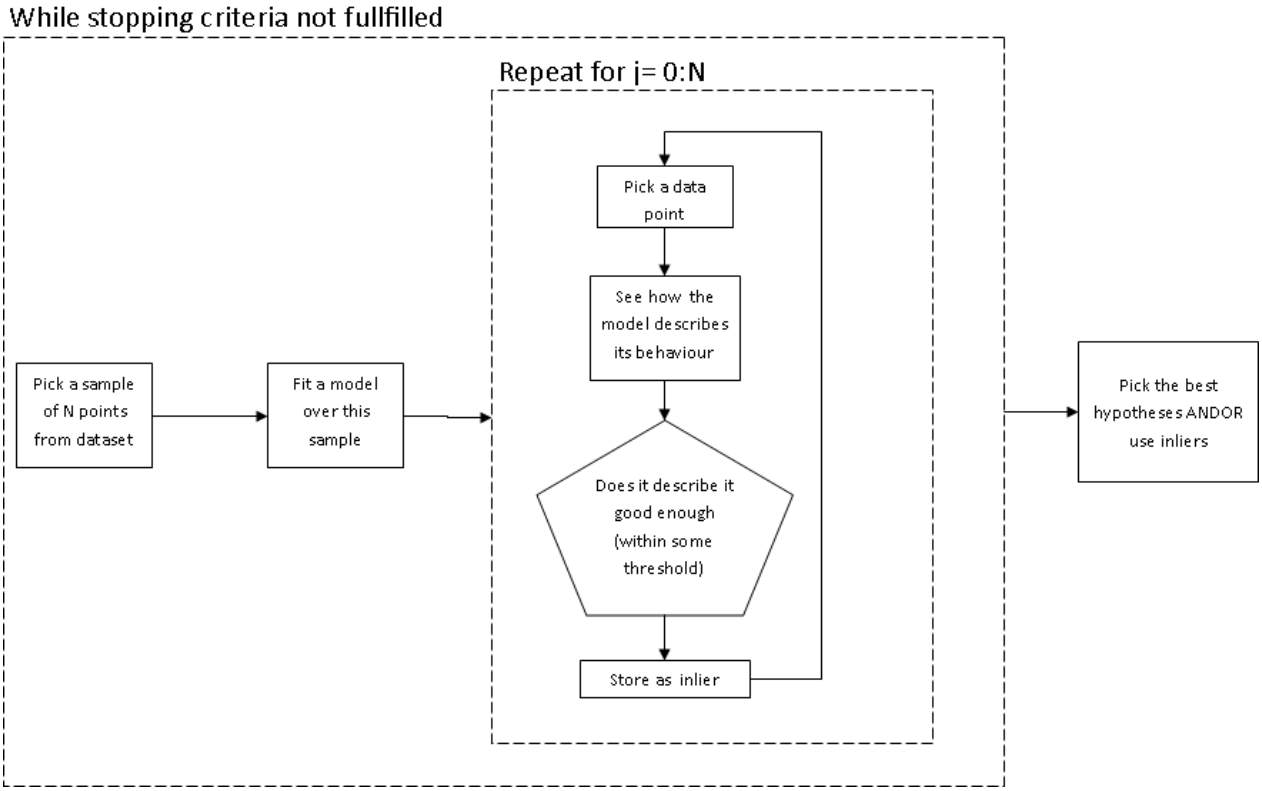


Figure 3.7 The RANSAC algorithm

3.3.4 Estimating the 3D motion

For this research a simple linear method was chosen that could quickly generate a motion hypothesis to test in a RANSAC iteration. The motion will be described by a rotation R and translation T element that is obtained by solving (48). The corresponding coordinate vectors are described by X_t at consecutive frames at time t.

$$X_{t+1} = R \cdot X_t + T \qquad (48)$$

To solve for R and T a minimum of 3 points are required. Following are the steps on how to obtain these from (48).

- Find the centroid, removing the translation component, of the 3D points $X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

$$centroid_t = \frac{1}{N}\sum_{i=1}^{n} X_{i,\ t} \qquad (\ 49\ )$$

$$centroid_{t+1} = \frac{1}{N}\sum_{i=1}^{n} X_{i,\ t+1} \qquad (\ 50\ )$$

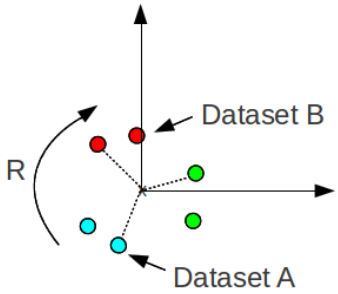


Figure 3.8 Figure taken from [39]. *To obtain pure rotations, the translation component needs to be removed by finding the centroid of each dataset.*

- Accumulate the centered vectors in a $3x3$ covariance matrix H.

$$H = \sum_{i=1}^{n} (X_{i,\ t+1} - centroid_{t+1}) \cdot (X_{i,\ t} - centroid_t)^T \qquad (\ 51\ )$$

- Use the Singular Value Decomposition (SVD) to obtain the singular matrices of H. The rotations are along H's principal axis which can be found with the SVD.

$$[U\ S\ V] = SVD(H) \qquad (\ 52\ )$$

  The left and right singular matrices *U* and *V* need to be multiplied to obtain the rotation matrix R.

$$R = V \cdot U^T \qquad (\ 53\ )$$

  - It can happen that R is a reflection matrix. To check this take R's determinant; If $\det(R) < 0$: *Multiply 3rd column R with -1*

- Finally the translation can be obtained by rotating the centroid from time t to the frame at $t + 1$ and adding that centroid.

$$T = -R \cdot centroid_t + centroid_{t+1} \qquad (\ 54\ )$$

3.3.4.1 Updating the pose

The camera motion that this method estimates can also be used to update the pose at the first stage of the key-frame tracker in STAM, which currently uses an M-estimator for pose-estimation. Bypassing the need for this computationally intensive method could negate some of the extra computationally load that is required by this segmentation method.

It can happen that after 80 iterations no suitable motion hypotheses have been found because none has provided enough inliers. In that case the M-estimator from STAM will need to provide the pose update. The minimum number of inliers is set at $\frac{N}{\tau} < inliers$, where N is the total population size and τ a fixed number, which has been experimentally set at 5. It can happen that hardly any features comply with the set error margin if there is an exorbitant amount of (motion) disturbance.

The rotation and translation components of the camera pose are updated separately; these are represented by $P(R)$ and $P(T)$ respectively. In (55) the translation component $P(T)$ is updated by first transforming the translation, which was estimated in the camera's local coordinate frame, into the global coordinate frame using the rotation $P(R)$ at time t. The rotation component $P(R)$ is updated in (56).

$$P(T)_{t+1} = P(R)_t^{-1}T + R \cdot P(T)_t \quad (55)$$

$$P(R)_{t+1} = R \cdot P(R)_t \quad (56)$$

3.3.4.2 Finding a better fitting motion model

STAM uses an M-estimator for the pose, as is explained in section 2.1.2.4. It improves robustness partly by using weights on each measurement so that good fitting measurements, that have a smaller error, have a larger influence on the result.

The idea that will be explained in this section is that a small improvement in our motion estimation might be made by lending this idea of weighing the measurements from the M-estimator.

Because this method re-estimates the model after the first round of inlier detection, see the dashed line in Figure 3.2, the resulting errors from the inliers can be used as weights in the re-estimation and can thereby improve it.
This is also known as Wahba's problem [40] which tries to estimate a satellite's attitude A by using weighed w_i unit vectors of observations u_i and v_i in their local coordinate frames using a least squares error (57).

$$A_2 = \frac{1}{2}\sum_{i=1}^{N} w_i \|u_i - A_1 v_i\|^2 \quad (57)$$

The weight w_i could in this case be found using a weight function with the Mahalanobis distance as input, section 3.3.4.3 will go into more detail on that subject.
The solution to Wahba's problem used in this research is basic and only requires a small adaptation to the method explained in 3.3.4. The covariance matrix H from (51) is multiplied by the weight for each vector before finding its SVD;

$$H = \sum_{i=1}^{n} w_i (X_{i,\ t=1} - centroid_{t=1}) \cdot (X_{i,\ t=0} - centroid_{t=0})^T \quad (58)$$

The rest of the steps to estimate motion remain as they are.

Because the motion estimate will also be used to update the pose, a third and final weighted re-estimation can be done to further refine it.

3.3.4.3 The weight function

The weight function for w_i to be used in (57) could be similar to that used for robust estimators.
The weight function that might work best is Tukey's; Figure 3.9 shows a plot of this function ε is the error and c the threshold for when a point is still considered an inlier.

$$w(\varepsilon) = \begin{cases} \left(1 - \left(\frac{\varepsilon}{c}\right)^2\right)^2 & if\ |\varepsilon| \leq |a| \\ 0 & if\ |\varepsilon| > |a| \end{cases} \quad (59)$$

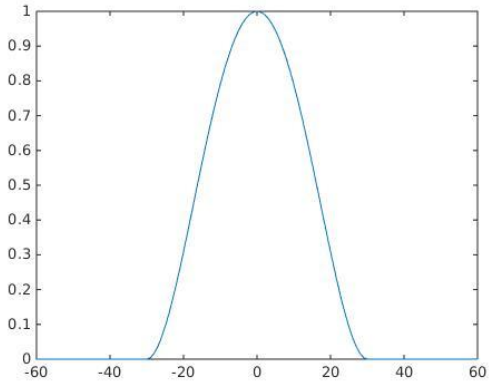


Figure 3.9 Tukey weight function, c is set to 30 in this case. Errors larger than the threshold c (30) are discarded as outliers, similar to setting the cutoff of $|\varepsilon| > |a|$.

The zero gradient at zero error means that most small errors are weighed equally strong. The almost linear descent means larger errors are weighed relative to their error. The lack of a root in this function also means it is easier to compute.
Another weight function is the "Fair" weight function, shown in equation (60) and Figure 3.10, which has a steeper initial gradient and converges to zero slower.

$$w(\varepsilon) = \left(1 + \frac{|\varepsilon|}{a}\right) \quad (60)$$

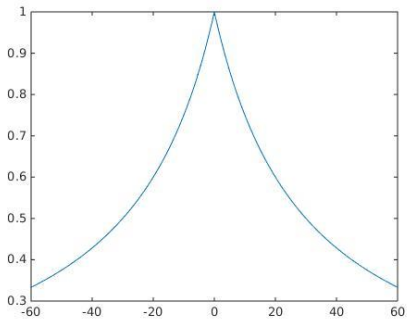


Figure 3.10 The 'Fair' weight function.

The slope is a lot steeper which means that this function has a higher preference towards good fitting points. It has to be tested what function would work best since there are many different weight functions. Four different weight functions are shown in Figure 2.9.

3.3.5 Sample preselection using sceneflow

The speed of the algorithm largely depends on how quickly a good initial sample is picked from the set of 3D vectors, since this affects the moment at which a correct camera motion can be estimated. So a procedure allowing a more selective initial picking of vectors for the sample might help to accelerate this.

In this case vectors are selected using the similarity of their sceneflow, since this can be calculated for each point individually. Although the sceneflow can only describe three of the six degrees of freedom present in the scene, it can make sure that all samples contain points whose motion is not too dissimilar. A large enough margin for the sceneflow difference should make sure that this pre-selection does not exclude points that actually have a similar 6 DOF motion, but due to rotations and a large spatial distribution (a long rotating stick for example) have a 3 DOF motion that is considered too dissimilar.

3.3.6 Region growing clustering

The motion segmentation method spends a lot of time testing motion models that are rejected when they do not fit the displacement of a sufficient number of points. Each motion model is tested on nearly all points requiring a lot of computation time. So the faster the algorithm can notice by itself that the motion model does not fit sufficient points, and stops testing it, the better.

It can be considered that points that belong to the same object and lie side by side are more likely to have a similar motion. Using this notion a crawler can be designed that goes through the set of points methodically, following a mesh which is created using Delaunay triangulation. That way a new iteration can stop earlier if all connected points in that mesh show dissimilar motion.

The region growing process involves the following steps;

1) Mesh points using the Delaunay triangulation method.
2) Crawl through this mesh from multiple starting points in the sample, (so with a sample size of four, there are four starting points)

Figure 3.11 shows how this works out; connected points with dissimilar motion are connected with red lines and similar motions are connected with green lines. After it notices that all inliers connect to outliers it stops. In Figure 3.11 it can be seen that about halve the points are not tested, which saves computation time on a wrong motion model. The red lines connect to points that don't fit the motion model, so the testing stops there. The green dots do fit the model and the routine therefore continues to crawl along the lines to test new points. It can be seen that it stops at the book, which is moving. A starting point in the bottom right that started on the book did not find any fitting points, so it stopped after three points. The points in black were not tested since the function could not grow further. This saved on computations for a model that was probably not a very good fit to begin with.

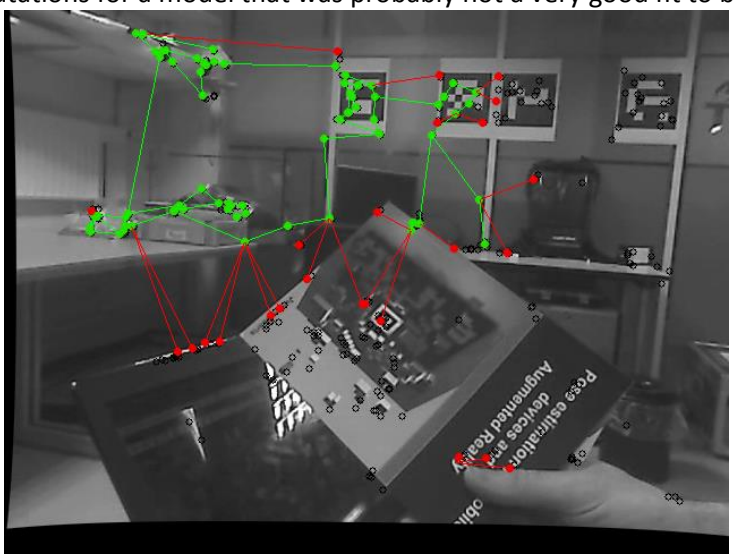


Figure 3.11 A visualization of the path followed by the region growing process and where it stops.

The downside of region growing is that if the scene contains a lot of outliers, groups of similar points can become separated by them. The effect might be that the method won't be able to find a solution at all. Plus it only makes sense in large data-sets, say larger than 40, because the creation of the mesh also takes time which might negate the savings of region growing.

3.3.7 Improving the scoring function

The set of static features and the corresponding motion model is selected using a scoring function (26). This scoring function depends on the image coordinate covariance C which is calculated using the inlier ratio of each bin (27). This adds a weight to the calculation of the covariance to reduce the effect of bins with few inliers.

However, since the feature space is quite sparse each bin generally holds just one point. So the inlier ratio for each bin usually returns either a 1 (the bin contains only inlier(s)) or a 0 (no inlier(s)).
The inlier ratio per bin is simply the number of inliers per bin divided by the total number of points in that bin $\frac{\sum b_i}{N_b}$. So if there is only one point in a bin, then the inlier ratio will be either 0 or 1.

But this makes it less fair for better fitting models that have a smaller chance of also (incidentally) including points on a moving object. So to make this ratio a bit more accurate it might be better to use a weight function, such as Tukey's (59). This measure to grade the fitness of a point with respect to a model more precisely might help the algorithm to come up with a more accurate motion model.

3.4 Conclusion

This chapter introduced the method that will be used to segment dynamic features from a stereo camera feed. This is an adaptation from a monocular segmentation algorithm made compatible for a stereo camera setup.
The use of a stereo setup allows for easier motion estimation that can be done independently from other processes like mapping. Noise and measurement errors still persist, however, in the calculated 3D position. So a noise model is therefore used to adjust for this when comparing a point's displacement with the hypothesized camera motion.

Some other additions to the process have also been presented with the intention to improve efficiency and effectiveness of this algorithm. Their level of effectiveness will be demonstrated in chapter 5 and discussed. Because this algorithm estimates the camera motion, it can also replace the first pose estimation step which is currently done using an iterative Gauss-Newton method explained in 2.1.2.4. It is therefore important that the motion estimate from our method is as accurate as possible.

4 Integration

This section will explain how the segmentation method is integrated into the pose tracking process of STAM. The segmentation method needs to receive tracking data generated by STAM from both the right and left camera frame. This information will be used to determine which features are outliers. The result of this process is then send back to be processed by STAM to account for the outliers in its further pose estimation process. This exchange in data between STAM and the motion segmentation process is also shown in Figure 3.2.

Implementing the motion segmentation process into STAM meant making alterations to its code. It was attempted to implement this with as few adjustments to STAM as possible.

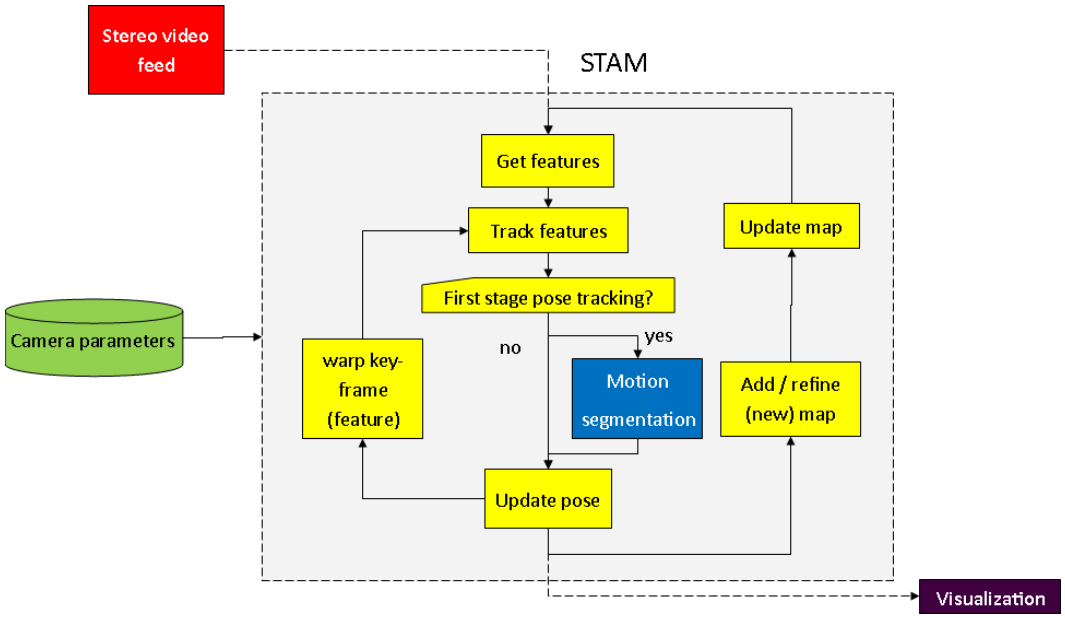


Figure 4.1 A simple overview of the AR framework. The grey dashed box represents STAM. The blue box is where the motion segmentation will be implemented in STAM. The colors correspond to those processes also shown in Figure 3.2.

Figure 4.1 shows where in the pose tracking process of STAM the motion segmentation method needs to be implemented. This is at the first stage of the process; after the world points have been tracked in the current image and before STAM performs the first estimation of the pose.

This means that the segmentation process needs to run in series with the pose tracking process. The simplest solution is therefore to implement the motion segmentation algorithm as a function that can be called from the pose tracking process.

4.1 OpenCV library

In this project a lot of use has been made of the OpenCV library to avoid writing code for processes such as image processing. The Mat type variable is also an easy to use way to do matrix operations and store and retrieve data. The downside is that it is not very efficient compared to arrays. A small test showed that it can take 10 times longer to do a matrix operation with the cv::Mat type compared to mimicking this process in a for loop with a floating point array.

OpenCV is an open source computer vision library developed by Intel in 1999 and released to the public in 2000. It was mainly meant to speed up the development of their CPU-intensive applications. (http://opencv.org/)

There is currently a large community supporting it, so it can be assumed that most of the functions in the library are bug-free. It also includes the more popular computer vision algorithms and provides variable types that are very useful in working with computer vision processes.

The algorithm initially took on average 40 milliseconds (ms) per frame to detect moving features, with the maximum number of RANSAC iterations set to 60. This might prove to be a bottle neck for use in an AR application which has to process frames in real-time. A standard camera running at 30 frames per second leaves the AR application 33 ms between frames to do its processes. STAM takes on average 26 ms to do a full pose tracking process on a single frame, which leaves about 7 ms to perform the motion segmentation process.

The motion segmentation process repeats a lot of matrix operations, for instance to estimate the camera motion and to test for each feature whether its 3D motion is reasonably similar to that of the camera. By performing these matrix operations using floating point arrays instead of cv::Mat types these processes can be performed 10 times faster.
Further optimizations where made by declaring variables that have large amounts of memory allocated to them only once at initialization, instead of each time the function is called. Overall these optimizations combined decreased computation time by about 85%.

4.2 What data is send to the motion segmentation process

- Since the segmentation process needs to find outliers in the tracked features, it receives the left and right image coordinates (that are tracked using the Pyramid Lucas Kanade method) from STAM over the previous and current frame.
- To calculate the 3D coordinates of the stereo matched features relative to the left camera, the calibration parameters of that camera are also needed. Because these are constant and available from a data file, the program retrieves these from that data file only at initialization.
- For visualization of the results the rectified images of the left camera frame is also obtained from STAM. These aren't required for the segmentation process but are useful while developing and debugging the segmentation process.

4.3 What data is send to STAM

- The list of which feature are considered outliers is sent back as a vector of integers. Each integer corresponds to the entry of the feature vector that has been found to be dynamic, so these features can be identified in STAM by that number. It is sorted by entry number to make it easier to delete these respective entries from the feature vector using a for loop.
- The estimated camera motion is also returned because it can be used to update the pose.

4.4 Changes made inside STAM

The algorithm is implemented as a function to be called from STAM. To process the returned information (so STAM can ignore dynamic features) requires the following changes:

- The features are tracked in two stages, first using LK optical flow and afterwards key-frame tracking. This happens at different locations in the code using different variables. Therefore the dynamic features need to be excluded from further processing in two different variables at different locations in the pose tracking code.
  This is done using the vector holding the entries of the tracked features that are considered to be dynamic.
- A routine to update the pose using the estimated camera motion in the first tracking stage was also implemented. This process is also explained in section 3.3.4.1. But tests have shown that it does not work well.

5 Experiment setup and test results

In this section the results of the tests of the motion segmentation algorithm will be presented. To measure the effectiveness of the pose estimation combined with the motion segmentation algorithm it is tested in a dynamic environment and compared to a ground truth to obtain the error.

The performance of STAM with the motion segmentation algorithm will also be compared to what can be considered to be currently the state of the art in (monocular) visual odometry; SVO [22]. A short explanation of this program was given in section 2.5.

Next to the accuracy of the pose estimation, the computation time will also be an important performance indicator. The reason for this is that with decreased lag the virtual objects can blend into the real world more smoothly and make the AR experience more convincing to the user.

5.1 Research setup

The ground truth is obtained using a marker based visual odometry program Aruco [12], which is explained in Appendix A. The choice was made for Aruco because it has shown to outperform almost all other freely available marker based visual odometry programs.

Since it uses markers, it will not be disturbed by the moving objects in the scene and should therefore return the ground truth pose reliably enough for testing. These markers can become occluded when an object or person moves in front of them. Placing multiple markers in the environment should avoid occlusions as much as possible but not completely. In such cases Aruco returns an error message indicating it does not know the camera pose. The frames in which full occlusions occurs therefore have to be skipped in the evaluation of the pose accuracy.

Figure 5.1 Multiple markers are used to avoid Aruco becoming 'lost' as much as possible. It is important that their relative distance, corresponding to a file read by Aruco holding these distances, is accurate enough for good camera pose estimation.

Because the markers are recorded from a distance they needed to be large enough to be recognized by a 0.3 mega-pixel camera which is why they are printed on A3 paper. A fiducial with three markers was created and printed on three A3 sheets as shown in Figure 5.1, and placed along the background. The file holding the information on the spatial configuration of the markers was then altered to correspond to their real world placement.

There are four different video sequences used for testing. These should resemble slightly different scenarios of an object or person moving through the scene with the background at varying distance. Because the covariance used in calculating the Mahalanobis distance creates a bias towards features further from the camera we would also like to check if the algorithm works with points lying at relatively the same distance, which is the case when the camera is close to the wall or pointed at a desk. These scenes are filmed with a simple stereo camera. This setup is shown in
Figure 5.2. Each camera makes 30 frames per second (fps) and has a resolution of 680 x 480 pixels.

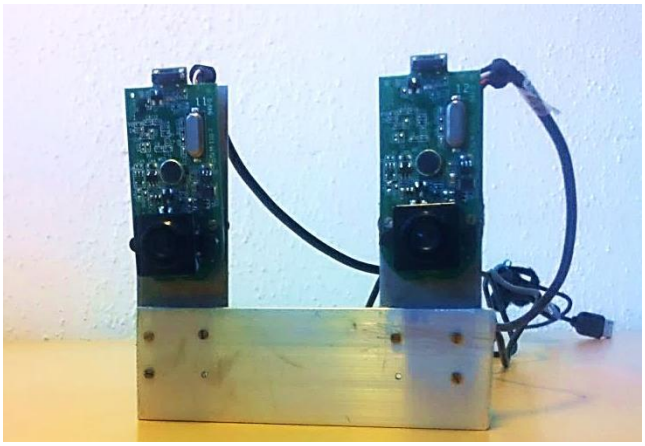


Figure 5.2 The stereo camera setup.

5.2 Varying results due to optimization routines

Because the pose and 3D location of the points are estimated using a Gauss-Newton optimization process and are in a multi-threaded environment the result of the estimated pose can differ on each run. The mapping process runs in a different thread as the tracking process and will improve the map using global or local bundle adjustment while the tracking process runs in another thread. If the tracking process needs information of map coordinates it can interrupt the bundle adjustment and use whatever is optimized at that moment. Because each thread will also constantly be interrupted by the operating system and other background processes the outcome will always vary to a certain extent. Figure 5.3 and Figure 5.4 show the pose error of multiple runs of STAM.

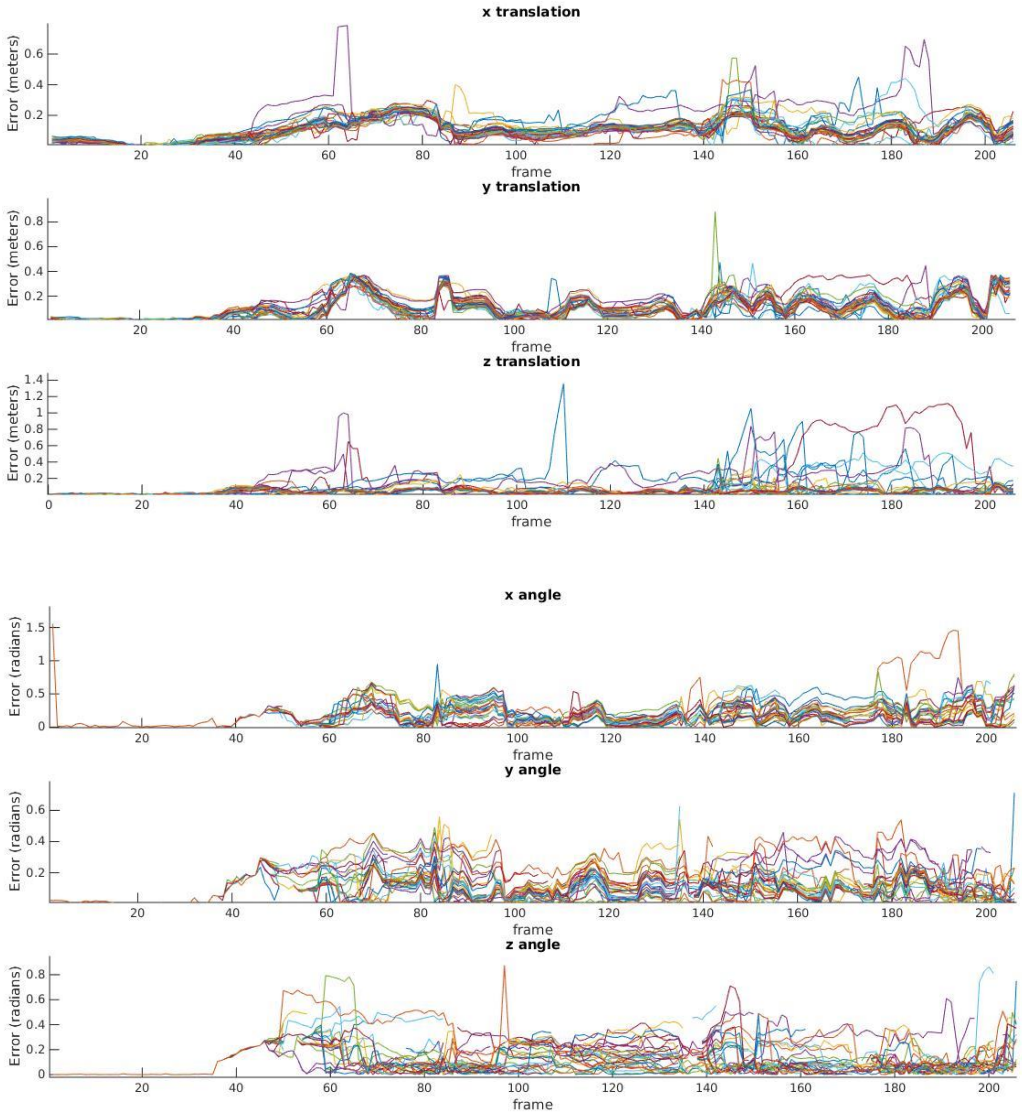


Figure 5.3 Segmentation results of 30 different runs showing the pose error from STAM compared with the ground truth provided by Aruco using markers.

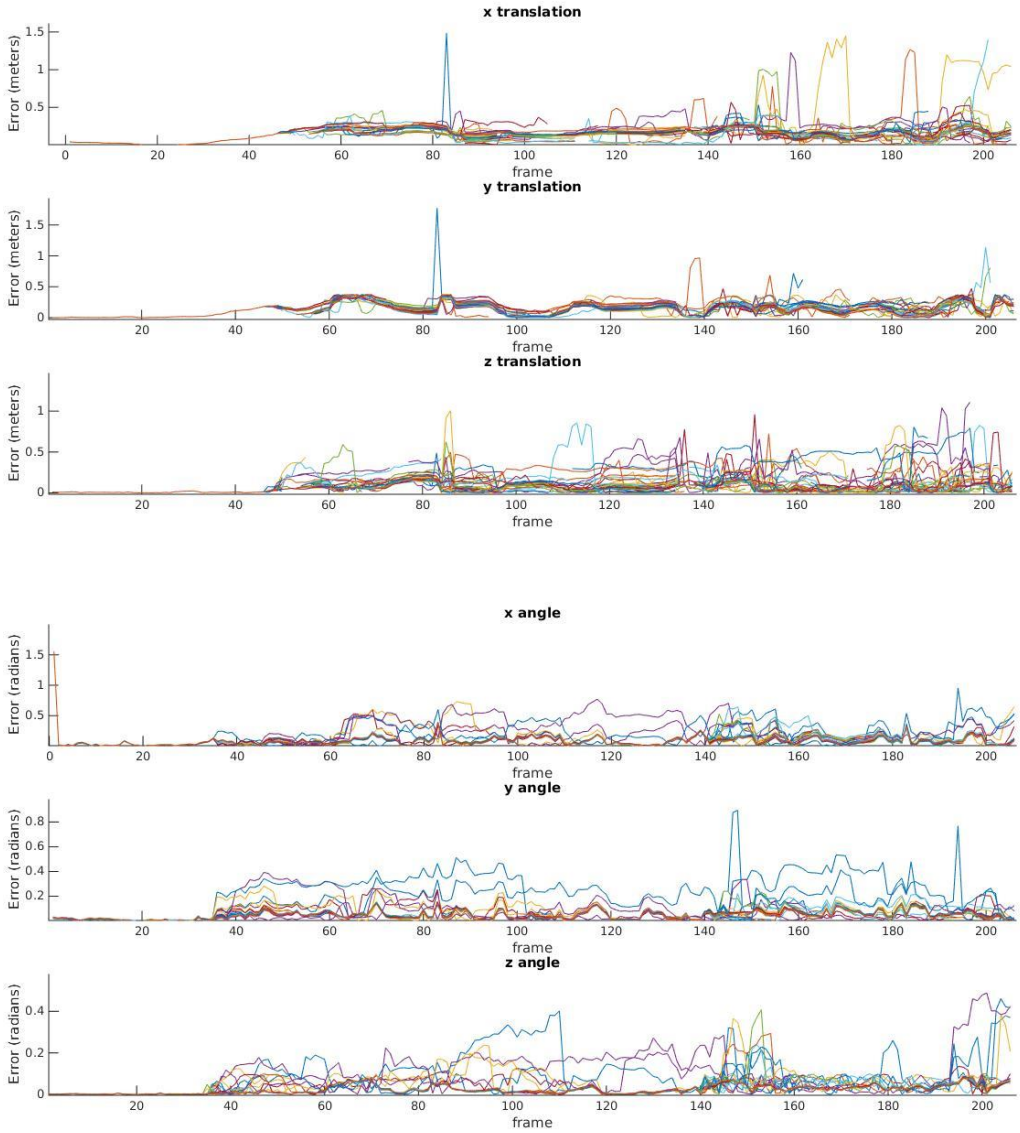


Figure 5.4 30 runs for STAM on a test video. The error is obtained by comparing the pose returned by STAM with the pose obtained by Aruco using AR markers.

It can be seen from Figure 5.3 and Figure 5.4 that although the settings of STAM are constant, the results vary to some extent. Except for some outliers the majority of the lines do follow roughly the same path, shown by a thick cluster of lines.

We might therefore assume that the distribution of the results of each run is Gaussian. So according to the law of large numbers enough runs should average out to the expected value of the estimated pose. There is a debatable rule of thumb that a sample set of 30 can be considered large enough. During some trial runs the results already seemed to average out after 20 runs, so 30 are considered to be enough for this experiment.

5.3 Scoring of the data

For easy comparison each run is given a score. There are two metrics by which we will judge the performance of the algorithm, these are;

- Pose error; This is the main benchmark for the performance of the algorithm. The estimated pose from STAM is compared to the ground truth obtained with Aruco. We will measure the pose at the second (final) stage of the pose tracking process.
- Computation time. The run of the pose tracking process is timed in milliseconds (ms).

The pose error is represented by the absolute mean error (ME) (61), and the root mean sum of the squared error (RMSE) (62). Because 30 runs are done for each test it is preferred to take the average of the errors over every run.
The RMSE can be considered more important for this research since the main purpose of this method is to avoid large errors which can be more noticeable to the user. They might for example make the virtual object go out of view. Because the aim of this research is to improve the user experience of the AR framework, large errors should be weighed more heavily.
At run i $pose_j$ is the estimated pose at frame j and $grnd_j$ the ground truth for the corresponding frame.

$$ME = \frac{\sum_{i=0}^{runs} \sum_{j=0}^{fames} |pose_j - grnd_j|}{runs \cdot frames} \quad (61)$$

$$RMSE = \sqrt{\frac{\sum_{i=0}^{runs} \sum_{j=0}^{fames} (pose_j - grnd_j)^2}{runs \cdot frames}} \quad (62)$$

The errors are summed separately over either the three rotations or the three translations. So for each run we end up with 2 ME's and 2 RMSE's, one for the rotations and one for the translations.

At times STAM is not able to return a measurable pose estimate but instead returns a non-representable value (NaN), which makes it impossible to calculate the error. Because an unknown pose is also not a preferable result it will be penalized with an error score of 1. This number was chosen arbitrarily, but looking at the error plots this more or less corresponds to the largest errors measured during testing.

5.4 What needs to be tested

The proposed motion segmentation algorithm is a modification of the algorithm described in [11]. The alterations made to it were explained in the subsections of 3.3. We would like to see what the effects of these alterations are on the performance of the motion segmentation. The experiment should therefore answer the following questions;

1) Does the algorithm improve the accuracy of the estimated pose? This is the most important test because it is central to this research.

2) What improves the motion segmentation more, the Mahalanobis distance or the Euclidean distance? In this research the choice was made to use the Mahalanobis distance because it should cope better with decreased location accuracy of points removed further from the camera. Short trial runs already showed an improvement in segmentation but this had not been accurately validated.
3) The algorithm gives each found set of inliers a score that to a large extent is based on its spatial distribution. In this research it was chosen to use the 3D world distribution rather than the 2D image distribution, as is explained in section 3.3. But does selection of the best set of inliers improve when using a 3D distribution?
4) Does giving weights to the points at the second motion estimation stage improve the selection of inliers? An essential part for good motion segmentation is the estimation of the camera motion. Using weights for each feature point should improve the motion estimate, as is explained in section 3.3.4.2.
5) Can the estimated camera motion be used to update the pose? The algorithm estimates the camera motion from frame to frame. As explained in section 3.3.4.1 this information can be used to update the pose so we can skip the first Gauss-Newton pose update and save on computation time.
6) Because the 3D location for each feature can be calculated individually we can calculate the translations to improve the initial sampling as explained in section 3.3.5. Does this selective sampling using the known translational velocity then speed up the motion segmentation process?
7) As explained in section 3.3.6, static features can be considered to be neighboring each other. Going through the set of features more methodically might decrease computation time since they can be rejected quicker. So does region growing speed up motion segmentation?

The results of these tests will be presented in this chapter and discussed further in chapter 6.

5.5 Creating the test data

Several issues were encountered during the creation of the test data which are mentioned here;

Decreasing motion blur

Fast camera movements can blur the image because photons coming of a part of the scene can be picked up by multiple pixels from the image sensor while the shutter is opened. This can affect feature tracking which relies on sharp contrasts in the images for reliable matching over consecutive frames. To avoid this as much as possible the shutter time of the stereo camera needs to be decreased. But decreasing the shutter time leaves the sensor with less time to receive sufficient photons to create the image, resulting in a darker image, the extent of how dark depends on the quality of the image sensor. Increasing gain and brightness can negate this effect to some extent, though increasing the gain also increases the noise. Increasing the brightness will worsen the glaring effects of places that are already quite bright by themselves.

In the end an optimal setting was found that had as little motion blur as possible by setting the fastest shutter speed without having to increase the gain and brightness to such extent it would greatly decrease the image quality.

Problem unsynchronized stereo video
Problems were encountered when creating the stereo camera recordings.
An essential part of the application is stereo matching. This is performed along the epipolar line so it is vital that objects align along it. This requires the time difference between both images to be as small as possible. A time difference between images of tens of milliseconds can already create feature coordinate differences, depending on its velocity, too large for reliable stereo matching.
A multi-threaded program is therefore used to record video for both cameras (almost) simultaneously. As it turned out the video buffer required some time to process a frame, which was not always given. The effect was that from either camera a frame could be dropped. So when playing back that video the left video would be at frame 100, for example, while the right was already displaying frame 101 or 102. This meant a difference between frames of several tens of milliseconds. Enough for a moving scene to misalign all features along the epipole and disrupt the stereo matching process significantly, making the segmentation process fail for all cases.
The solution was simply to add a sleep cycle of just a few milliseconds to give time for the video buffer to run its processes and store the frame.

5.6 Test results
The results are presented for each video separately.
There are four different videos;

1, 2) In two videos an object (book) moves through the view of the camera. One of these videos starts by translating the camera horizontally to allow SVO to obtain a baseline.
3) A person holding a board moving through the view. The board is to make sure enough features will be detected.
4) The camera looks at a desk with an object moving through the scene to test its performance when the variation in depth is negligible.

5.6.1 A book moving through the scene
The scene for this experiment is shown in Figure 5.5 and its results in Figure 5.6. The various additions are tested only in this video because it was the most dynamic, meaning it had a lot of movement in it.

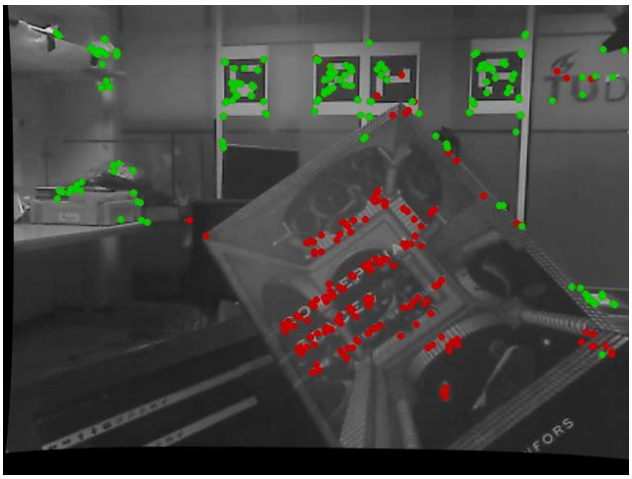


Figure 5.5 Motion segmentation on a book moving through the camera view. The red dots are identified as moving points which should not be used for pose estimation. The green points are considered to be static points.

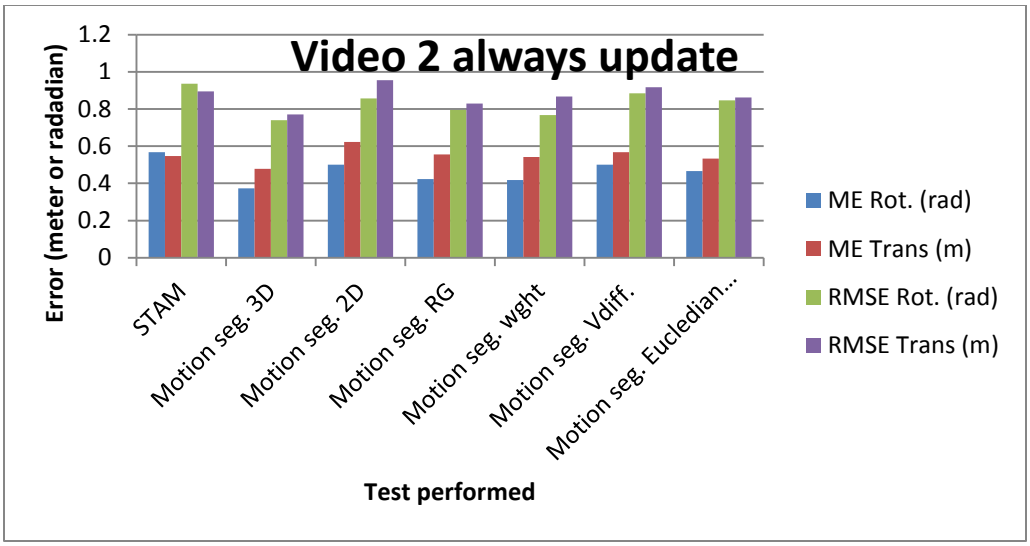


Figure 5.6 This graph shows the average error over 30 runs of the program on the video. The abbreviations along the x axis are explained here;

- ***STAM**; AR framework running without motion segmentation.*
- ***Motion seg. 3D**; STAM with motion segmentation with the score for a set of inliers is calculated using the 3D distribution of the inliers.*
- ***Motion seg. 2D**; STAM with motion segmentation with the score for a set of inliers is calculated using the 2D image distribution of the inliers.*
- ***Motion seg. RG**; STAM with motion segmentation using the 3D distribution for scoring and region growing.*
- ***Motion seg. Wght**; STAM with motion segmentation using the 3D distribution. Weights are calculated for each inlier with which the motion is estimated.*
- ***Motion seg. Vdiff;** Selecting the sample based on difference in lateral velocity.*
- ***Motion seg. ED**; STAM with motion segmentation using the Euclidean distance instead of the Mahalanobis distance.*

For the motion segmentation method using the 3D distribution the decrease in error for the translation is 14% and 21% for the rotations.
It can be seen that using the 3D real world distribution rather than the 2D image distribution of inliers increases performance on motion segmentation.
Selecting the sample using the criterium that the translational velocity should be comparable (Vdiff.) decreases the accuracy. Similar as when using a weighted motion estimation (Wght.).

The motion segmentation algorithm requires at least 3 features to estimate a motion hypothesis; an extra fourth is included to counter noise. So if there are less than 5 features there are simply not enough features to segment and the motion segmentation can be skipped.
During implementation of the algorithm the code was changed so the estimated camera motion could be used to do a pose update. This resulted in a bug that caused a skip of the first stage pose update if there were fewer than 5 features tracked in the new frame, which improved performance considerably. The reason for this effect is that a pose estimate using very few features can hardly be performed reliably since the M-estimator will have trouble determining a good reference to decide on which the inliers are and which the outliers.
The result of selectively skipping a pose update if there are too few features is shown in Figure 5.8.

Using the estimated motion to update the pose significantly decreased the accuracy of the pose tracking process.
To further investigate the effect of selective pose updates STAM also ran with a selective update of the pose. This showed significant improvements using only this small alteration. Motion segmentation improves it even further and shows a decrease in RMSE of 62% for rotations and 53% for translations over the unmodified STAM.

Figure 5.7 shows the results for the various configurations of the program, the abbreviations are similar to those explained for Figure 5.6. For this test the program is run 30 times over, similar to the tests on pose estimation accuracy. The average time is taken over these 30 runs.

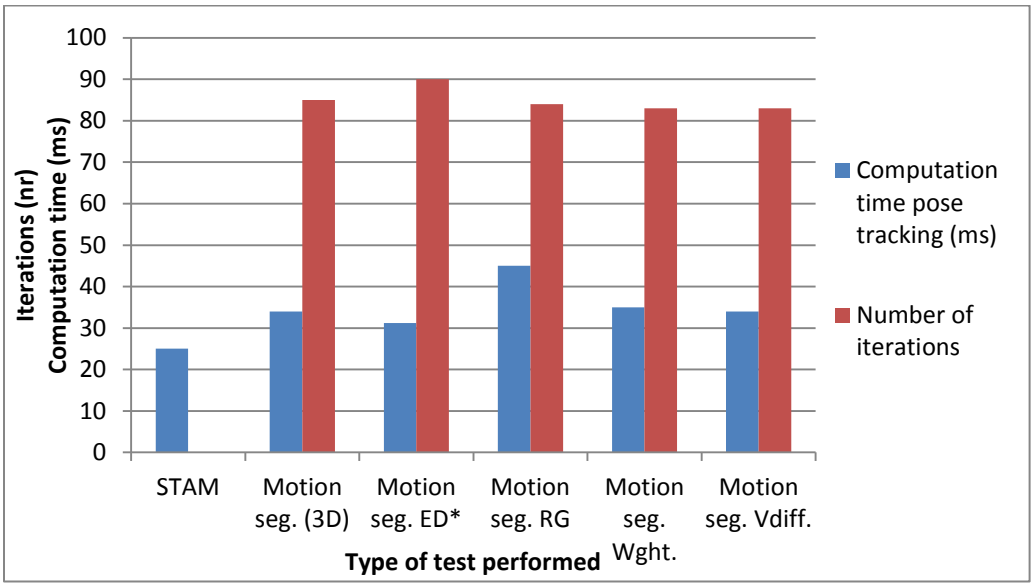


*Figure 5.7 Compuation time of the pose tracking process and the number of iterations for some different configurations of the motion segmentation algorithm. *ED; Euclidean distance. The maximum number of iterations was set to 120.*

The average computation time for the pose tracking process including the motion segmentation algorithm is 34 ms, while without motion segmentation the pose tracking takes about 25 ms. So the motion segmentation itself takes about 9 ms. The computation time for frames for which the algorithm could not segment motion due to a lack of tracked features are not added to the total. The process to detect motion needed on average 82 iterations to detect outliers.
The maximum number of iterations was set to 120 but this could be set lower. The motion segmentation algorithm did not return a significantly better result if the number of iterations was set to more than 60. At 60 iterations the algorithm ran a few milliseconds faster, about 6 ms on average.
This was tested on a x86 64 bit quad core 2.3 GHz Intel i7 computer.

Region growing did not improve outlier detection or computation time. On the contrary; creating the mesh takes extra time, as does traversing it. It might be because the region growing process can stop before having traversed through all the points that good points are more likely to be skipped. This can

decrease the likelihood of finding the best set of inliers, which can decreases pose estimation performance in STAM.

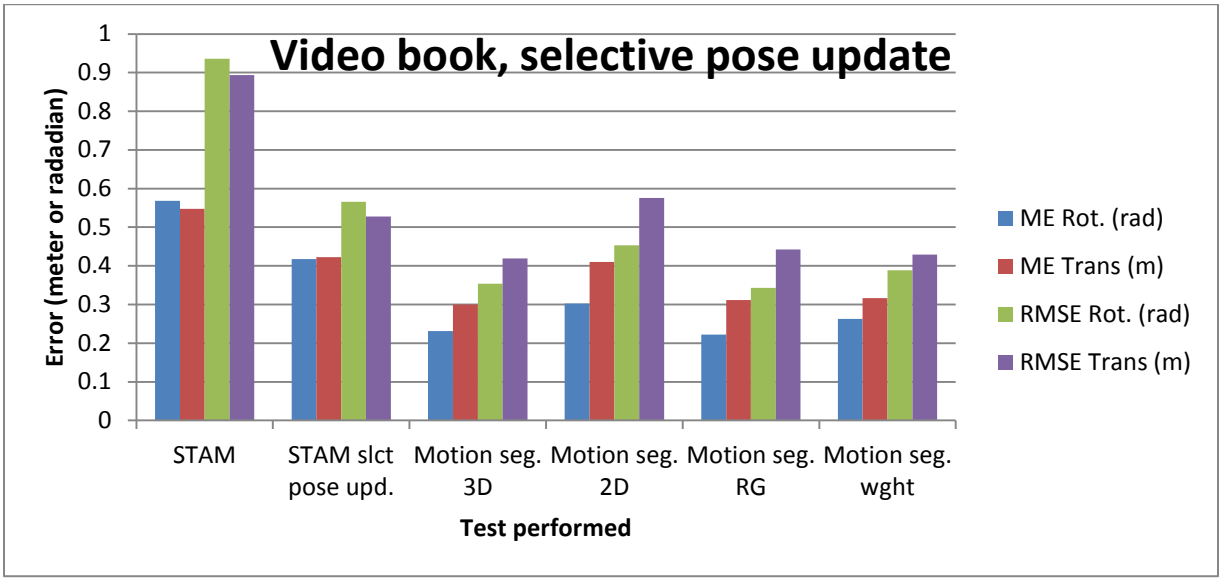


Figure 5.8 This table shows the errors (ME and RSME) of STAM with and without motion update when the first pose estimate is skipped if there are too few features in the current frame. The first graphs marked with 'STAM' is placed for comparison and always does a pose update, also when there are too few (0 to 4) features available. The abbreviations are the same as in Figure 5.6 and also explained in that figure's caption.

5.6.2 Comparison to SVO

In this test STAM with and without motion segmentation will be compared to SVO. This scene is shown in Figure 5.9.

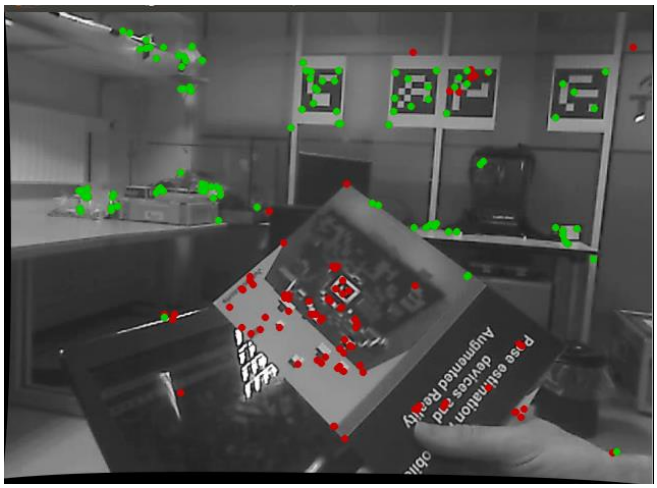


Figure 5.9 A still from the scene in which STAM with motion segmentation can be compared with SVO.

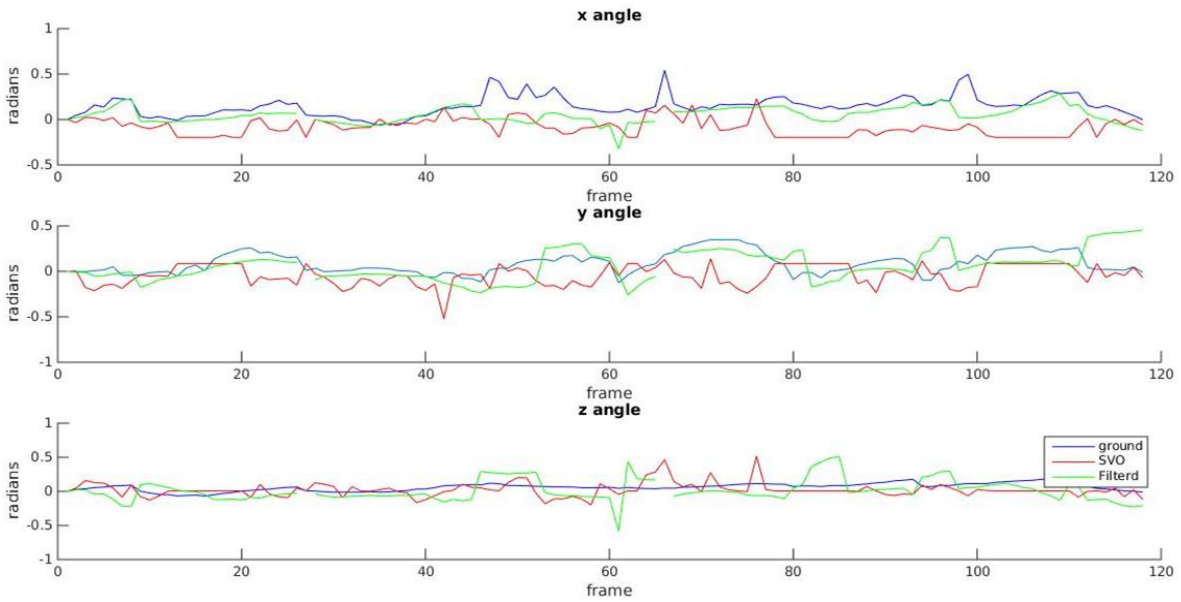


Figure 5.38 The estimated angles over consecutive frames of the ground truth obtained using Aruco (blue line), STAM with Motion segmentation (green line) and SVO (red line). Because SVO is monocular it does not have a real sense of scale. The translations are therefore left out because they cannot be compared well.

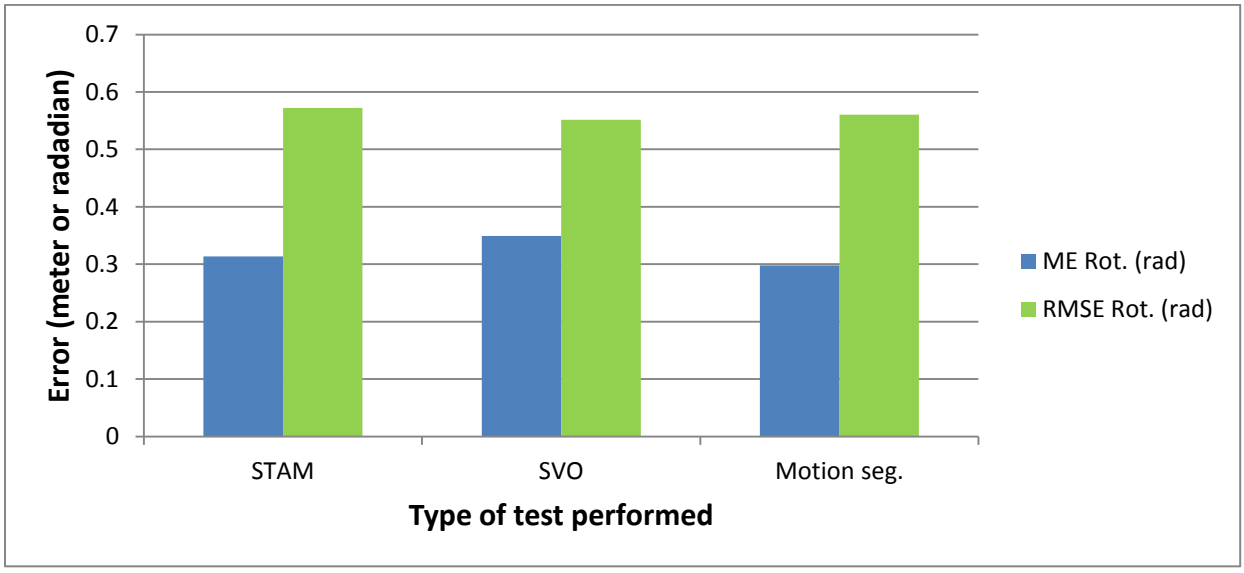


Figure 5.10 The ME and SMSE error scores for STAM, SVO and the motion segmentation algorithm. Because the SVO uses a monocular camera the scale of the translations of SVO are unknown and are thus left out of the comparison.

Because SVO is monocular it needs to initialize a baseline before it can start mapping the environment. , Translations cannot be compared because SVO's world map scale is **arbitrary**. Rotations on the other hand can be compared; these results are shown in Figure 5.10.

In this scene it performed best and slightly outperformed STAM. There is however hardly any difference with the motion segmentation algorithm. SVO required a lot of configuring to make it run properly. The pace at which it creates new key-frame needed to be increased significantly for SVO not to lose its tracked features as soon as the camera made just a slight movement.

5.6.3 Video of a person with a marker board

In this video a person is moving a marker board through the scene as shown in Figure 5.11. This marker board was used because it has a lot of good features on it that can be tracked. This marker board is not used by Aruco so it does not interfere when it obtains the ground truth.

Figure 5.11 A still from the video where a person is moving through the scene holding a marker board which was only used to make sure that enough features are tracked on the moving object.

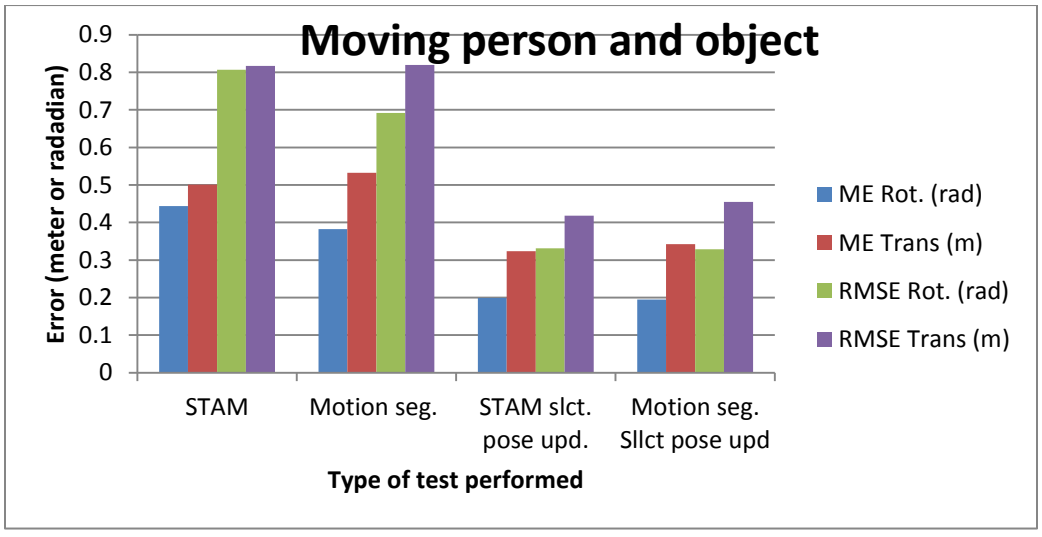


Figure 5.12 The test result showing the averaged errors over all frames.

It can be seen that for the rotations the motion segmentation makes an improvement of 14% over STAM. For translations there is no improvement. Again, skipping pose updates when there are too few features significantly improves the results.

5.6.4 Video of a desk scene

A scene of a desk is filmed with a magazine moving over it. The goal of this test is to see whether the Mahalanobis distance might still be beneficial if points are located at equal distance from the camera.

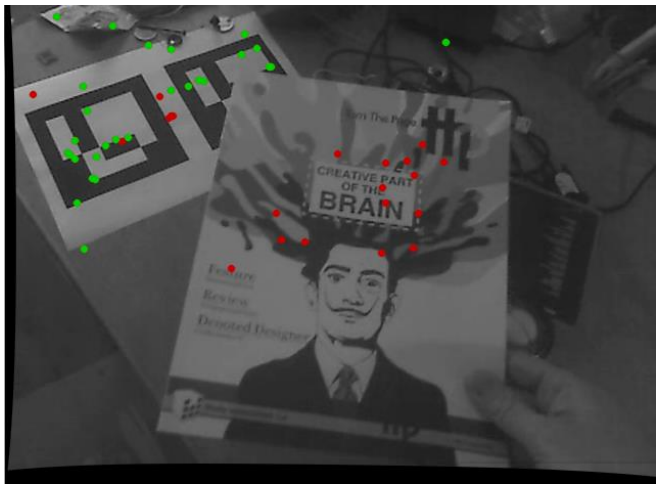


Figure 5.13 A still from the video where all objects are close by to test if the depth bias of the algorithm might have negative consequences.

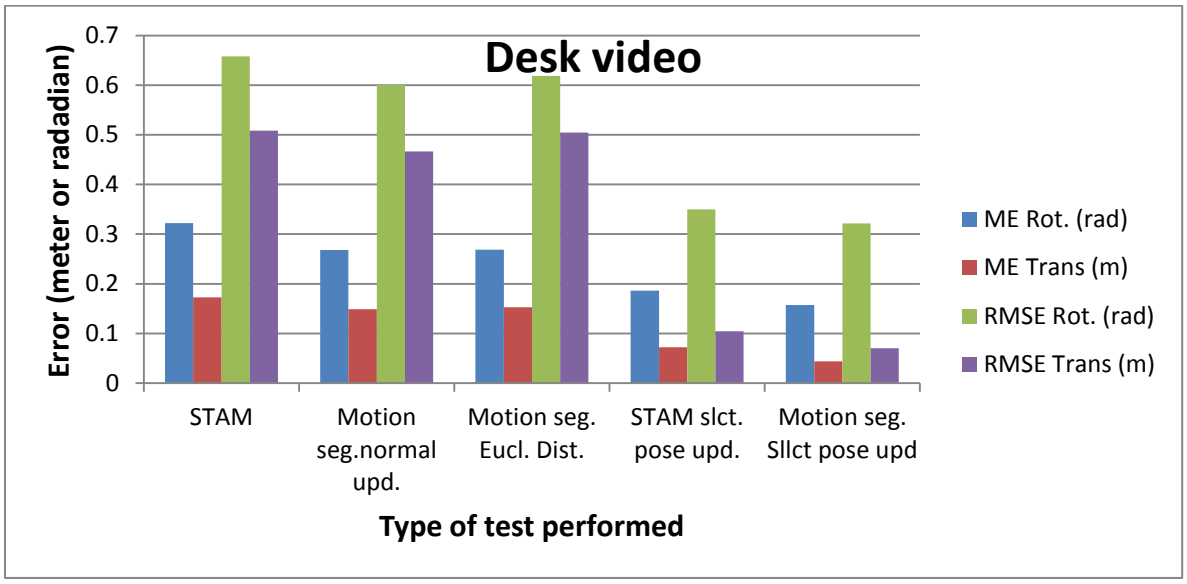


Figure 5.14 Results of the tests. The first set of columns marked 'STAM' are from STAM without motion segmentation. The second are from STAM with motion segmentation using the Mahalanobis distance while third (Motion seg. Eucl. Dist) uses the Euclidean distance. The fourth and fifth are from STAM with and without motion segmentation but doing a selective pose update.

Using the Mahalanobis distance instead of the Euclidean distance in the segmentation process improves its performance. So the covariance, which increases with distance, still improves performance when all points lie at about an equal distance.
When we skip a pose update when there are too few features (less than 5) it greatly improves performance by itself as can be seen in Figure 5.14 under *'STAM slct. Pose upd.'* which improves rotations by 47% and translations by 80%. When the motion segmentation is included in STAM it improves another 4% and 6% for the rotations and translations respectively.

5.6.5 Pose update

As explained in section 3.3.4.1, it is also attempted to update the pose using the estimated camera motion from our segmentation process. This however caused significantly larger pose errors, mainly in the translational directions. In Figure 5.15 the pose estimates of STAM (red) and the estimated motion (green) is shown, the ground truth is represented in blue. The estimated motion is not used to update the actual pose of STAM but shown as a virtual pose update to evaluate the estimate from the motion segmentation algorithm. Inspection shows that the rotations seem to be estimated reasonably accurate. The translations on the other hand behave a lot more erroneous. When using the estimated camera motion in the pose update of STAM during test runs, the largest errors were found in the estimated translations.

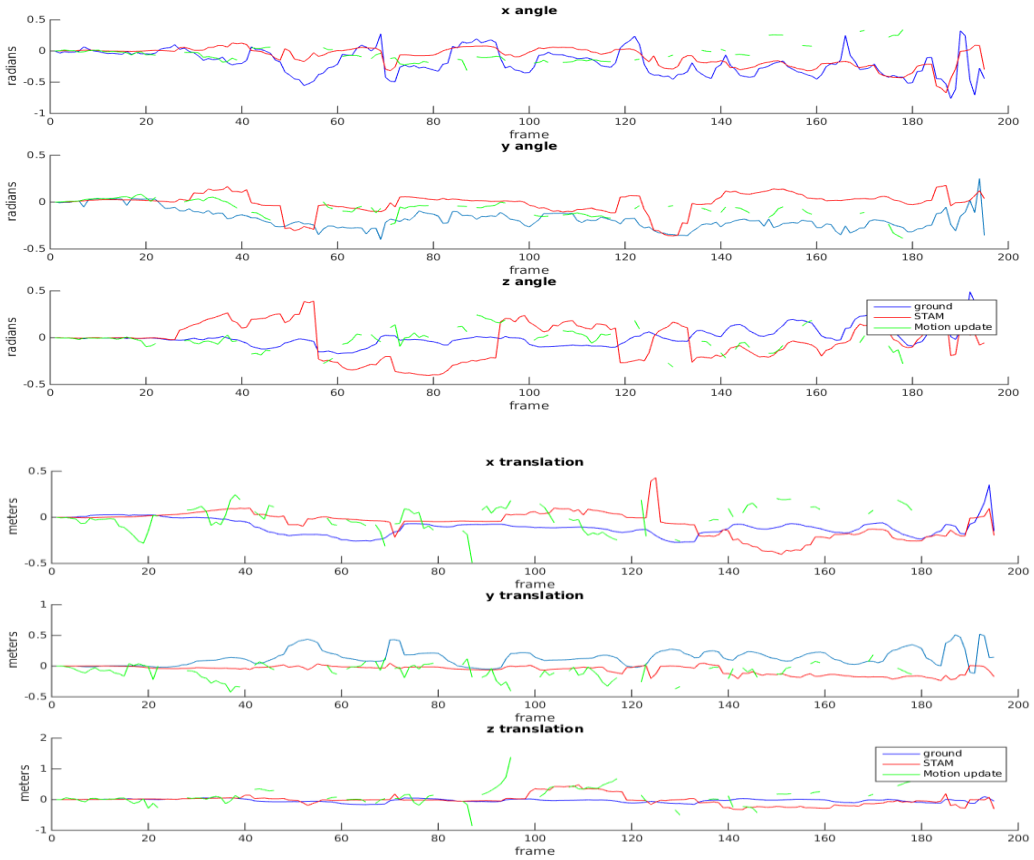


Figure 5.15 The estimated camera motion (green) from the motion segmentation process. This is done separately from the actual pose update and is not used in the actual pose estimation of STAM.

6 Conclusions

In this thesis we have shown how a motion segmentation algorithm designed for a monocular visual odometry program could be adapted to one for a stereo camera. It could benefit from using the extra visual information made available by the second camera to model the error distribution of tracked features. This error distribution was used to improve the clustering process by using the Mahalanobis distance instead of the Euclidean distance. When comparing the use of these two error metrics the Mahalanobis showed an improved performance over the Euclidean distance.

Also using the 3D world distribution for scoring a set of found inliers, rather than the 2D image distribution, improved the selection of the correct set of inliers.
Since dynamic features tend to lie closer together and generally only one side of an object faces the camera, the features residing on the moving object are usually arranged in a plane like shape. This means that generally features on the side of a moving object which the camera can see are at relatively equal depth, decreasing their 3D distribution, while the static features are more spread out. So even when a moving object obscures much of the view the algorithm will still prefer feature sets that are also more distributed along the depth of the scene.

The accuracy of the pose estimation deteriorated significantly when the estimated camera motion was used to update the pose in the first stage of the pose tracking process. This might indicate that the estimated camera motion is not accurate enough because noise and outliers negatively influenced the motion estimate of the linear method. Using a weighted method did however not seem to improve this.

By doing matrix operations with arrays instead of the OpenCV Mat type, the computation time could be decreased by 50%. Because the matrix operations are 10 times faster using floating point arrays compared to openCV Mat type, more iterations can be done which might improve the accuracy of the motion segmentation process. Other optimizations decreased the computation time further with an extra 35%. This has made real-time performance possible, making it a useful addition to the AR framework. Further optimization might shave off a few more milliseconds.

The overall estimated pose improves significantly when frames that have too few tracked features are skipped. The reason for this is that robust pose estimation requires a sufficiently large set of features to make a statistical distinction between inliers and outliers. The reason why features are not tracked well in certain frames might be due to too much motion blur. Skipping such frames thus avoids erroneous pose estimations based on a poor data set.

We did not succeed in updating the pose using our outlier detector's estimated camera motion because this significantly increased the pose error for STAM. Inspection of the estimated motion, isolated from the further pose updating process, shows that the estimated translations are not very accurate. So the motion estimation process can use some improvements.

7 Appendix

7.1 ArUco

A recently developed library for camera pose estimation using fiducial markers is Aruco [12]. It has proven to outperform other popular systems, such as ARToolKitPlus and ARTag as shown in Figure 7.1. It has also shown to run at almost 100 Hz on a single core of a 2.40 GHz dual-core processor.
It creates its own fiducial marker with the number of markers for each fiducial to be set by the user. One such marker is shown in Figure 7.2. Using a fiducial with multiple markers makes it more robust against occlusions. When creating its own dictionary of markers, based on requirements in size and number of markers, it tries to maximize the inter-marker distance and number of bit-transitions. The more bits, which the system uses for identification and error detection and correction, the better. These markers should however not become too small to remain detectable.

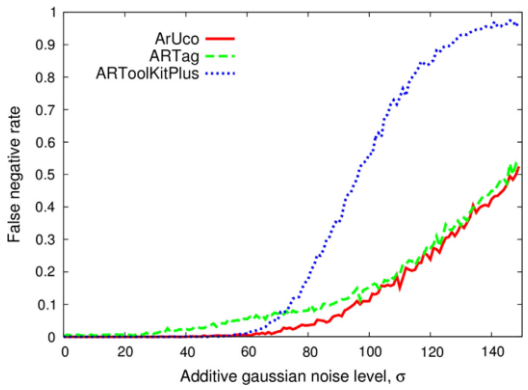


Figure 7.1 Taken from [12] it shows the comparative robustness of ArUco against added noise to 100 images containing markers from different viewpoints.

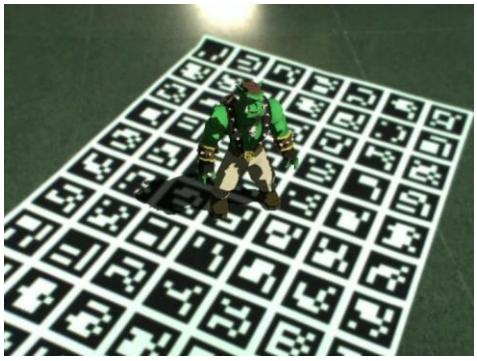


Figure 7.2 A marker that is generated by Aruco. It uses printable boards with multiple markers to increase accuracy and make it more robust against occlusions.

8 Bibliography

[1] Klein, Georg, and David Murray, "Parallel tracking and mapping for small AR workspaces.," in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on. IEEE*, 2007.

[2] Azuma, Ronald T., "A survey of augmented reality," *Presence 6.4 ,* pp. 355-385, 1997.

[3] Sutherland, Ivan E., "The Ultimate Display," in *Proceedings of IFIP Congress.*, 1965.

[4] Feiner, Steven, Blair Macintyre, and Dorée Seligmann, "Knowledge-based augmented reality," *Communications of the ACM ,* pp. 53-62, 1993.

[5] "Meta," [Online]. Available: https://www.spaceglasses.com. [Accessed 2014].

[6] O. Akman, Robust augmented reality, Delft: TU Delft, 2012.

[7] B. F. B. a. J. Ö. Blissing, *Augmented and Mixed Reality as a tool for evaluation of Vehicle Active Safety Systems,* 2013.

[8] F. Zhou, H. B.-L. Duh, and M. Billinghurst, "Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR," *ISMAR'08: Proc. 7th Int'l Symp. ISMAR'08: Proc. 7th Int'l Symp. on Mixed and Augmented Reality Cambridge, UK, Sep. 15-18 2008. IEEE CS Press. ISBN,* p. 193–202.

[9] M. K. K. a. L. I. Agrawal, "Real-time detection of independent motion using stereo," *Application of Computer Vision, 2005.,* Vols. WACV/MO-TIONS'05 Volume 1. Seventh IEEE Workshops on. Vol. 2. IEEE, 2005.

[10] Ess, Andreas, et al., "Moving obstacle detection in highly dynamic scenes," *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on. IEEE,* 2009.

[11] Shimamura, Jun, Masashi Morimoto, and Hideki Koike, "Robust vSLAM for Dynamic Scenes," in *MVA.*, 2011.

[12] Garrido-Jurado, S., et al. , "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition ,* vol. 47.6, pp. 2280-2292, 2014.

[13] J. Caarls, Pose estimation for mobile devices and augmented reality, Delft: TU Delft, 2009.

[14] Lowe, David G, "Distinctive image features from scale-invariant keypoints," *" International journal of computer vision 60.2 ,* pp. 91-110., 2004.

[15] Lucas, Bruce D., and Takeo Kanade, "An iterative image registration technique with an application to stereo vision," *IJCAI,* vol. 81, 1981.

[16] Bouguet, Jean-Yves, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," Intel Corporation 5 , 2001.

[17] Fischler, Martin A., and Robert C. Bolles., "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM ,* vol. 24.6, pp. 381-395, 1981.

[18] Lepetit, Vincent, Francesc Moreno-Noguer, and Pascal Fua, "Epnp: An accurate o (n) solution to the pnp problem," *International journal of computer vision,* vol. 81.2 , pp. 155-166., 2009.

[19] Trucco, Emanuele, and Alessandro Verri, Introductory techniques for 3-D computer vision. Vol. 201., Englewood Cliffs: Prentice Hall, 1998.

[20] Mair, Elmar, et al, "Adaptive and generic corner detection based on the accelerated segment test," *Computer Vision–ECCV 2010. Springer Berlin Heidelberg,* pp. 183-196, 2010.

[21] B. Delaunay, "Sur la sphere vide."," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk ,* Vols. 1-2, pp. 7.793-800, 1934.

[22] Moreno-Noguer, Francesc, Vincent Lepetit, and Pascal Fua, "Accurate non-iterative o (n) solution to the pnp problem," *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on. IEEE,* 2007.

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

[24] Baker, Simon, and Iain Matthews, *Baker, Simon, and Iain Matthews. "Lucas-kanade 20 years Lucas-kanade 20 years on: A unifying framework,* Robotics Institute, Carnegie Mellon University, 2002.

[25] Strasdat, Hauke, J. M. M. Montiel, and Andrew J. Davison. , "Real-time monocular slam: Why filter?," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on IEEE*, 2010.

[26] Vogiatzis, George, and Carlos Hernández, "Video-based, real-time multi-view stereo," *Image and Vision Computing 29.7 ,* pp. 434-441, 2011.

[27] K. E. a. L. V. G. Ozden, "Background recognition in dynamic scenes with motion constraints," *Computer Vision and Pattern Recognition, 2005.CVPR,* vol. 1, no. IEEE, p. IEEE Computer Society Conference on, 2005.

[28] Lenz, Philip, et al., "Sparse scene ow segmentation for moving object detection in urban environment," in *Intelligent Vehicles Symposium (IV), 2011 IEEE. IEEE*, 2011.

[29] Kundu, Abhijit, K. Madhava Krishna, and C. V. Jawahar, "Realtime motion seg-mentation based multibody visual SLAM," in *Proceedings of the Seventh Indian Con-ference on Computer Vision, Graphics and Image Processing. ACM*, 2010.

[30] Rao, Shankar R., et al, "Robust algebraic segmentation of mixed rigid-body and planar motions from two views," *International journal of computer vision 88.3,* pp. 425-446., 2010.

[31] Moosmann, Frank, and Thierry Fraichard, "Motion estimation from range images in dynamic outdoor scenes," *Robotics and Automation (ICRA), 2010 IEEE International Conference on. IEEE,* 2010.

[32] Muller, D., Mirko Meuter, and S-B. Park. , ""Motion segmentation using interest points," *Intelligent Vehicles Symposium, 2008 IEEE. IEEE,* 2008.

[33] Tan, Wei, et al, "Robust monocular SLAM in dynamic environments," in *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on. IEEE*, 2013.

[34] Sünderhauf, Niko, and Peter Protzel, "Stereo odometry—a review of approaches," *Chemnitz University of Technology Technical Report,* 2007.

[35] C. L. Winter, " Normalized Mahalanobis distance for comparing process-based stochastic models," *Stochastic Environmental Research and Risk Assessment ,* no. 24.6, pp. 917-923, 2010.

[36] P. C. Mahalanobis, "On the generalised distance in statistics," *Proceedings of the National Institute of Sciences of India,* p. 49–55, 1936.

[37] J. Ekström, "Mahalanobis' Distance Beyond Normal Distribution," 2011.

[38] K. O. Arras, " An Introduction To Error Propagation:Derivation, Meaning and Examples of Equation FCF," the Autonomous Systems Lab, Institute of Robotic Sys-tems, Swiss Federal Institute of Technology Lausanne (EPFL), 1998.

[39] N. Ho, "Nghia Ho," [Online]. Available: http://nghiaho.com/?page_id=671. [Accessed 2014].

[40] G. Wahba, "A least squares estimate of satellite attitude," *SIAM review 7.3 ,* pp. 409-409, 1965.

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

[42] Sinha, Sudipta N., et al, "GPU-based video feature tracking and matching," *EDGE, Workshop on Edge Computing Using New Commodity Architectures,* vol. 278, 206.