# Using Mobile Multi-Camera Unit for Real-Time 3D Motion Estimation and Map Building of Indoor Environment

**Vom Fachbereich Elektrotechnik und Informatik der Universität Siegen**

zur Erlangung des akademischen Grades

**Doktor der Ingenieurwissenschaften**

**(Dr.-Ing.)**

genehmigte Dissertation

von

**M.Sc. Chayakorn Netramai**

1. Gutachter: Prof. Dr.-Ing. Hubert Roth

2. Gutachter: Prof. Dr.-Ing. Anatoliy Sachenko

Vorsitzender: Prof. Dr.-Ing. habil. Otmar Loffeld

Tag der mündlichen Prüfung: 21.01.2011

To my loving family.

# Acknowledgements

First of all, I would like to thank my supervisors. I am deeply grateful to Professor Hubert Roth, head of the Institute of Automatic Control Engineering, University of Siegen, who gave me the opportunity to do this thesis work and for his valuable advice and guidance during this long study. I am also very grateful to Professor Anatoliy Sachenko, head of the Department of Information Computing Systems and Control, Ternopil National Economic University, for his willingness to take me under his guidance and for all his valuable comments.

I would like to express my deep appreciation to Professor Otmar Loffeld and Dr. Stefan Knedlik from the International Postgraduate Programme Multi Sensorics, University of Siegen, for their support and guidance during the past few years at the university of Siegen.

I am grateful to Professor Alois Knoll, Dr. Christian Verbeek and Dr. Matthias Radecker for their support and inspiration during my stay at the Fraunhofer Institute in Sankt Augustin.

Special thanks to all my work colleagues and friends, Fábio Bisogno, Poramate Manoonpong, Chanin Joochim, Oleksandr Melnychuk, Christof Hille, Pakorn Ubolkosold, Seyed Ghobadi, Omar Löpprich, Miao Zhang, Friederike Bergner and also those that I might have failed to mention above for all their encouragements, fruitful comments and shared moments during the hardest and longest study in my life.

Finally, I would like to thank my parents and my sister for their endless love and support. I own them everything that made me what I am today.

# Zusammenfassung

3D-Kartengenerierung und 3D-Bewegungsschätzung in Echtzeit ausschließlich unter Verwendung visueller Daten sind zwei anspruchsvolle Problemstellungen, mit denen sich der Arbeitskreis zum maschinellen Sehen in den vergangenen Jahren intensiv auseinandergesetzt hat. Für die erfolgreiche Erstellung einer 3D-Karte bedarf es der genauen 3D-Bewegungsschätzung mit Hilfe eines Eingangssensors während des Kartierungsprozesses. Bisher waren die meisten Versuche zur Verbesserung der 3D-Bewegungsschätzung vor allem auf die eingesetzten Software-Algorithmen gerichtet. Doch trotz ausgeklügelter Algorithmen wird eine exakte 3D-Bewegungsschätzung weiterhin durch die Grenzen behindert, die durch den verwendeten visuellen Sensor hervorgerufen werden. Eine einzelne Kamera besitzt nur ein kleines Sichtfeld, was zum Problem der Bewegungsmehrdeutigkeit im Fall kleiner Bewegungen führt und damit zu ungenauen Bewegungsinformationen und einer schlechten Kartenqualität.

Diese Doktorarbeit stellt eine neue Multikamera-Hardware vor, die als ein optisches 3D-Messgerät zur Lösung der Probleme der Echtzeit-3D-Bewegungsschätzung und -3D-Kartierung verwendet werden kann. Der Fokus liegt dabei nicht allein auf der Software-Lösung, sondern geht einen alternativen Weg zur Verbesserung der Genauigkeit und Zuverlässigkeit der Bewegungsschätzung, nämlich mit Hilfe eines besseren Hardware-Designs. Das Ergebnis dieser Herangehensweise ist eine Multikameraeinheit (MKE), die auf eine extrem genaue 3D-Bewegungsdetektion ausgelegt ist. Sie besteht aus drei Stereokamerapaaren, die zu einer kompakten, mobilen Hardwareplattform zusammengefügt werden. Die einzigartige Kameraanordnung beseitigt die Mehr-

deutigkeit des Bewegungsfehlers, welche man in Einzelkamerasystemen findet, und liefert so eine präzise Bewegungsschätzung. Das erweiterte Sichtfeld dank mehrerer Kameras ermöglicht außerdem eine einfache, aber genaue Detektion der 3D-Bewegung der Kamera in Echtzeit ohne komplizierte Berechnungen. Die begleitenden Algorithmen, die für die Echtzeit-3D-Bewegungsschätzung benötigt werden, einschließlich der Detektion und des Abgleichs von Merkmalen in Echtzeit sowie Algorithmen zur Unterdrückung von Ausreißern, werden ebenfalls für die MKE implementiert. Darüber hinaus wird der FastSLAM-Algorithmus für die simultane Echtzeit-3D-Lokalisierung und Kartenerstellung implementiert, um eine einheitliche Merkmalpunktkarte und den Standort und die Ausrichtung der MKE beizubehalten. Infolge dessen erbringt das vorgestellte 3D-Bewegungsschätzungs- und 3D-Kartierungssystem mittels der MKE im Vergleich zu den herkömmlichen Einzelkamerasystemen eine höhere Leistung, was durch Simulationsergebnisse und Experimente unter realen Bedingungen bestätigt wird. Dies gilt besonders für die Leistungsfähigkeit der 3D-Bewegungsschätzung, bei der sowohl die rotatorische als auch die translatorische Mehrdeutigkeit des Bewegungsfehlers kompensiert wird. Der wahrscheinlichkeitstheoretische Ansatz für die Generierung von Merkmalpunktkarten zeigt eine hohe Echtzeit-Leistung und Konsistenz mit guter Genauigkeit. Schließlich wird die vorgestellte Multikamera-Hardware für 3D-fotorealistische Kartierungsprojekte verwendet, in denen ein hochwertiges 3D-Modell, welches die Umgebung korrekt repliziert, in Echtzeit erstellt werden kann.

# Abstract

Real-time 3D map building and 3D motion estimation using only visual data are two challenging problems which have been intensively studied by the machine vision community in the past decade. In order to successfully build a 3D map, the accurate 3D motion estimation of the input sensor during the map building process is needed. Up to now, most of the attempts to improve the 3D motion estimation process have been concentrated on the software algorithms used. However, despite the use of sophisticated algorithms, accurate 3D motion information is still hindered by the limitation of the visual sensor used, e.g. a single camera with small field of view which suffers the motion ambiguity problem in the case of small movements, leading to inaccurate motion information and poor map quality.

This thesis work proposes a new piece of multi-camera hardware to be used as a 3D visual sensing device for the real-time 3D motion estimation and 3D map building problems. Instead of focusing only on the software solution, this work takes an alternative approach to improve the motion estimation accuracy and robustness by means of a better hardware design. A multi-camera unit (MCU) which is aimed for high accuracy 3D motion detection is constructed. It consists of three pairs of stereo cameras which are put together as a compact, mobile hardware platform. This unique camera arrangement eliminates the motion ambiguity error found in single camera systems and so accurate motion estimation is obtained. The increased field of view by means of multiple cameras also enables a simple but accurate detection of 3D movement of the camera in real-time without any complex calculations. The accompanied algorithms which are needed for the real-time

3D motion estimation including the real-time feature detection and feature matching as well as outlier rejection schemes are also implemented for the MCU system. Moreover, the FastSLAM algorithm for real-time 3D localization and map building approach is implemented in order to maintain a consistent feature point map and the location and orientation of the MCU. As a result, the proposed 3D motion estimation and 3D map building using the MCU system gives a better performance compared to the conventional, single camera systems as confirmed by the simulation results and real world experiments. This is especially the case for 3D motion estimation performances, where the motion ambiguity error is being compensated in both rotation and translation cases. The probabilistic approach for 3D feature point map building shows a strong real-time performance and consistency with good accuracy. Finally, the proposed multi-camera hardware is used for a 3D photorealistic map building task where a high quality 3D model which correctly replicates the surrounding environment can be constructed in real-time.

# Table of Contents

# List of Figures

# List of Tables

# List of Symbols and Acronyms

## List of Symbols

| | |
|---|---|
| $b$ | Stereo baseline |
| $D$ | Disparity value |
| $D_{max}$ | A threshold value for the selection of feature points used in ICP algorithm |
| $i_P$ | Image intensity at point $P$ |
| $n^t$ | Set of data associate variables, $n^t = \{n_1, n_2, ..., n_t\}$ |
| $\mathbf{R}$ | Rotation matrix |
| $\mathbf{s}_t$ | Pose of the multi-camera unit at time $t$ |
| $s^t$ | Path of the multi-camera unit, $s^t = \{\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_t\}$ |
| $s^{t,[m]}$ | Path of the multi-camera unit within the $m^{th}$ particle |
| $\boldsymbol{\mu}_{n,t}^{[m]}, \boldsymbol{\Sigma}_{n,t}^{[m]}$ | The mean and covariance of the $n^{th}$ feature point in the $m^{th}$ particle at time $t$ |
| $\boldsymbol{\mu}_{s_t}^{[m]}, \boldsymbol{\Sigma}_{s_t}^{[m]}$ | The mean and covariance of the multi-camera unit pose in the $m^{th}$ particle at time $t$ |
| $\mathbf{t}$ | Translation matrix |
| $\Theta$ | Feature point map, $\Theta = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, ..., \boldsymbol{\theta}_N\}$ |
| $\mathbf{u}_t$ | The multi-camera unit odometry measurement at time $t$ |
| $u^t$ | Sequence of the odometry measurement, $u^t = \{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_t\}$ |
| $w_t^{[m]}$ | Importance weight of the $m^{th}$ particle at time $t$ |
| $\mathbf{x}_v$ | The multi-camera unit pose vector |
| $z^t$ | Sequence of the feature point measurement, $z^t = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_t\}$ |

# List of Acronyms

| | |
|---|---|
| CRF | Corner Response Function |
| DoF | Degree of Freedom |
| EKF | Extended Kalman Filter |
| FoV | Field of View |
| GUI | Graphic User Interface |
| HSV | Hue, Saturation, Value color model |
| ICP | Iterative Closest Point |
| MCU | Multi-Camera Unit |
| NCC | Normalized Cross Correlation |
| PMD | Photonic Mixer Device |
| RANSAC | Random Sample Consensus |
| RGB | Red, Green, Blue color model |
| SLAM | Simultaneous Localization And Mapping |
| SVD | Singular Value Decomposition |
| ToF | Time of Flight |

# Chapter 1

# Introduction

***Overview — Goals — Contributions — Content of the Thesis***

## 1.1 Overview

### The 3D Map

A map is a material or database that contains useful information of the observed environment including the basic geometric descriptions of objects within the environment such as location, length, size, appearance, etc. A map is usually presented by means of miniature graphics that are directly captured from the real environment. The images captured from different locations and perspectives of the environment are tiled together to present a complete replication of the environment. Figure 1.1 is an example of such a graphical map being made from a set of images captured by the Mars exploration robot. Not only does this map represent the 2D graphical explanation of the remote site in terms of intensity image, but the depth information of the site is also provided, which promotes the usefulness of the map even further.

While 2D graphic maps might be sufficient for many applications, many others require the full explanation of the environment in three dimensions. A 3D map is then

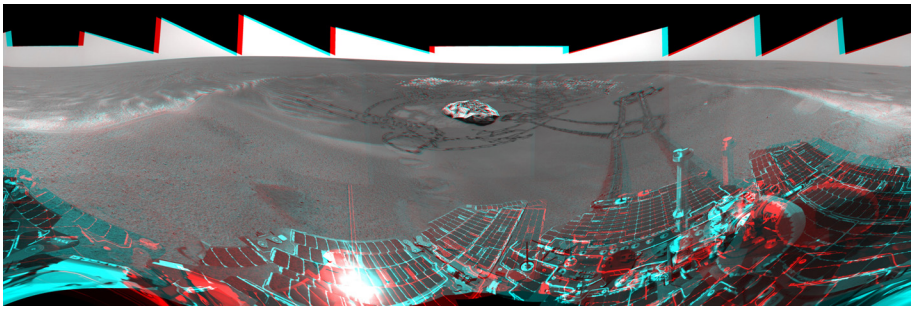

**Figure 1.1: A well-traveled 'Eagle crater', a 3D version of the view from the Mars exploration rover Opportunity on its 56$^{th}$ sol on Mars. The blue and red color in this anaglyph image encodes the depth information of the scene which can be seen when viewed with a pair of two color glasses. (photo credit: NASA/JPL)**

a better candidate since it contains the corresponding volume and fully 3D dimension information of the real world environment. Figure 1.2 shows an example of a 3D map captured by the Wägele platform [Biber *et al.* (2005)]. In this case complete geometry information as well as the texture of the environment is available in the map. Such a map is considered a 3D photorealistic map since an image derived from an arbitrary viewpoint of the 3D map has a similar quality compared to a photo taken from a real scene at the same location.

Some uses of the 3D maps include, but are not limited to:

- remote exploration with autonomous systems, autonomous robot navigation, etc.,
- cultural site preservation, engineering site measurement and
- generation of 3D multimedia contents for the entertainment industry.

## 3D Motion Estimation

The tasks of 3D map building are usually carried out using a 3D visual sensor. For instance, a stereo camera can be used to generate a high quality 3D image in less than a second. However, it is usually impossible to generate a 3D image of the entire environment in one shot due to certain limitations such as limited field of view, limited range of operation, etc. Therefore, to generate a map of a large environment, multiple measurements have to be taken in order to fully describe the environment. Generally the measurement tool is moved around the environment so that the entire space can be discretely observed. In order to join these multiple measurements the *egomotion*

**Figure 1.2: An example of a 3D model built by the Wägele platform. (photo credit: University of Tübingen)**

between each measurement is needed. Figure 1.3 depicts this scenario. Firstly, the first 3D image frame is taken. However the small field of view of the sensor makes it impossible to cover the whole space of the environment and therefore the 3D sensor is moved to a new location in order to make another 3D image of the environment. These two measurements can be easily combined if the 3D motion information between the two frames is available.

In machine vision, the motion information can be estimated using series of images captured by the observer during a movement. From each time step feature points are extracted from the images. The process called *visual odometry* uses these feature points to estimate the 3D motion information at each time step. The resulting 3D motion is a composition of two main components – translation and rotation, which can be described in terms of displacement, velocity and acceleration. The path that the observer has undergone can be described as a time-ordered collection of 3D locations and poses. Such a path is called a *trajectory*. It is also common to define a reference coordinate frame in order to describe the motion according to a certain reference frame.

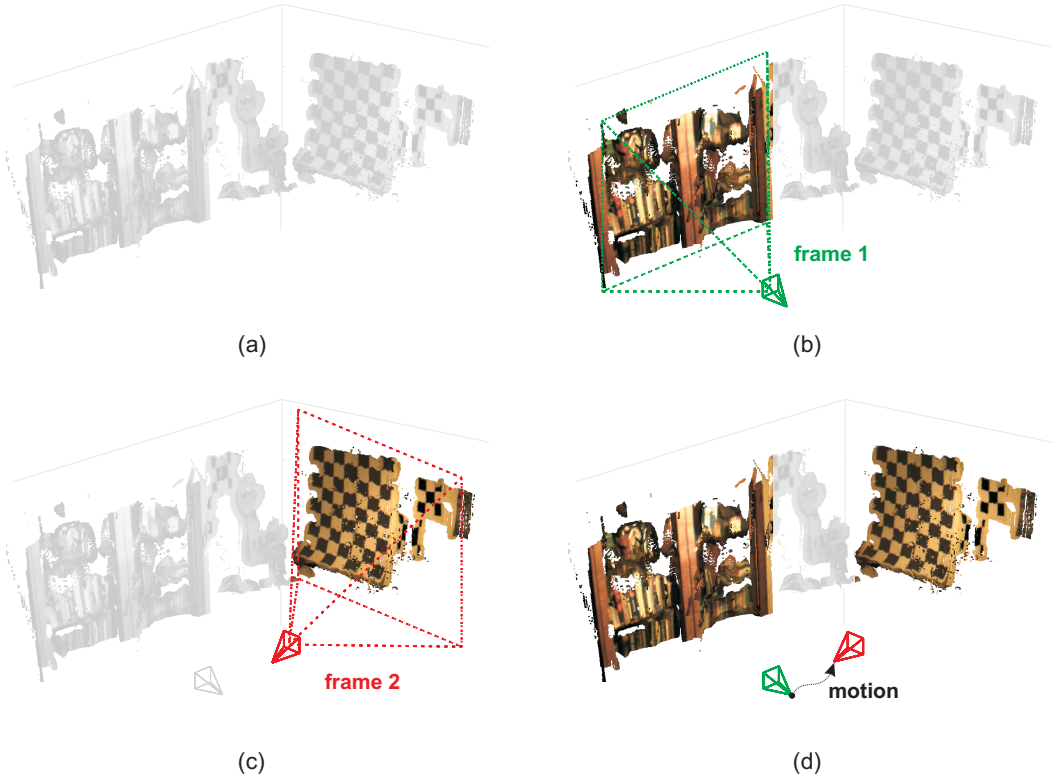


**Figure 1.3: Relationship between motion and map building. (a) target environment (b) image frame 1 is captured (c) camera is moved and image frame 2 is captured (d) two image frames can be put together if the motion is known.**

## 1.2 Goals

The main objective of this thesis work is the realization of a new 3D visual sensor for the 3D motion estimation and map building tasks. The 3D motion information is to be estimated using only visual information from the proposed hardware; no other odometry devices are used. The accompanied software algorithms for the 3D motion estimation and map building processes are to be implemented. Particular emphasis is also placed on the implementation of a probabilistic approach for real-time map building that copes with the uncertainties of the motion estimation and measurement error. Finally, the proposed hardware and its software system should be able to produce a photorealistic 3D map of any indoor environment as well as record the travelled trajectory during the map building process in real-time.

## 1.3 Contributions

The main contributions in this work are:

- Realization of a new mobile, multi-camera unit with the main purposes as a 3D motion estimation and 3D map building device (Chapter 3). Relevant publication: *"Real-Time Photorealistic 3D Map Building Using Mobile Multiple Stereo Cameras Setup"*[Netramai & Roth (2007)].
- Design and implementation of the accompanied algorithms and information usage scheme that tightly integrate and maximize the benefits for the 3D motion estimation process using the proposed multi-camera unit (Chapter 4).
- Provide a complete probabilistic implementation of the six degree of freedom localization and mapping solution using the proposed multi-camera unit as an input device (Chapter 5). Relevant publication: *"Real-Time 3D Motion Estimation and Map Building Using Enhanced Multi-Camera System"*[Netramai & Roth (2010)].

## 1.4 Content of the Thesis

The content of the thesis is organized as follow:

- **Chapter 1** contains this introduction.
- **Chapter 2** gives an overview of the different types of sensors. These include stereo camera, 3D laser scanner and PMD camera. Brief information, working principles and capability of each sensor are discussed. A comparison between each type of sensor is also given in order to show the advantage and disadvantage of each sensor, which is important for the selection of the 3D visual sensor for the multi-camera unit.
- **Chapter 3** introduces the multi-camera unit, its state of the art, hardware description, software description, hardware simulation results and calibration technique.
- **Chapter 4** describes the process of real-time 3D motion estimation using the multi-camera system. The algorithms that are used within the whole process will be explained. The motion estimation process which includes 3D motion detection using only 2D data and final 3D motion estimation are explained.

- **Chapter 5** describes the real-time 3D map building process. A probabilistic approach for the localization and map building is explained. A simulation of the multi-camera system is given in order to observe the performance of the map building algorithm. A method to generate a 3D photorealistic map is also explained.
- **Chapter 6** presents some experiment results by using the multi-camera system for real-time 3D motion estimation and map building. Several tests including motion estimation using pre-defined motions and paths are conducted and the obtained results are discussed. Several 3D photorealistic maps gathered using the multi-camera system are also presented at the end of the chapter.
- **Chapter 7** contains the conclusion of the work. Insights into future improvements to the system are discussed at the end of the chapter.

# Chapter 2

# 3D Visual Sensors

***Introduction — 3D Visual Sensors — Single-Beam Laser Range Finder — PMD Camera — Stereo Camera — Comparison of 3D Visual Sensors — Selection of 3D Visual Sensor for the Multi-Camera Unit — Conclusion***

## 2.1 Introduction

This chapter gives a quick overview of the current 3D visual sensor technology including the basic understanding about how the 3D sensors work and their usage for the task of 3D motion estimation and 3D map building. The introduction includes three kinds of 3D sensors, namely single-beam laser range finder, PMD camera and stereo camera. A brief technical review is provided, the strengths and weaknesses of each sensor are listed and compared. Finally, the key reasons for selecting the stereo camera as the current 3D sensor for the multi-camera hardware are provided.

## 2.2 3D Visual Sensors

A 3D visual sensor is a visual-based, non-contact sensor that is able to percept the three-dimensional geometry information and other physical properties such as texture

and color of the surrounding environment. There is a wide range of 3D sensors for indoor and outdoor usage and they can be classified by the techniques being used to gather depth information. Generally, there are two major techniques that are widely used to percept 3D information: *time of flight* and *triangulation*. The time of flight sensor is usually known as a laser range finder where a laser beam is used to determine the depth of the scene. The PMD camera, which is a newly developed time of flight 3D camera, is also becoming another candidate for short and medium range 3D vision applications. The triangulation visual sensor is mostly known as a stereo camera where images from two cameras are used for the calculation of the depth information. Since these sensors have different working principles they each have certain advantages and disadvantages. The following sections provide the explanations and comparison of the different types of 3D sensors used in the 3D motion and map building task in detail.

## 2.3 Single-Beam Laser Range Finder

The single-beam laser range finder is an optical sensing device that uses a laser beam to determine the distance to the objects within its sweeping angle and operational range. The time of flight (ToF) principle is used for the determination of the distance or range information. An example of a single-beam laser range finder that is being used in robotic and navigation applications is the SICK LMS200 (Figure 2.1). The LMS200 can detect objects within a 180 degree sweeping angle and 80 meter range. Due to the fast scanning frequency of up to 75 Hz it is one of the most widely used sensors for real-time navigation and obstacle avoidance for automated vehicles and autonomous robots. Table 2.1 lists the specifications of this particular hardware model.

### 2.3.1 Working Principle

The ToF principle is straightforward. First the sender, i.e. laser diode, emits a light signal which travels from the sender to the object. This light signal hits the object and reflects back to the receiver. By measuring the round-trip time, the distance between the sensor and the object can be calculated. The relationship between the round trip time and the distance is defined by:

$$distance = \frac{(ct)}{2} \tag{2.1}$$

where $c$ is the speed of light and $t$ is the round-trip time of the light signal between the sensor and the object. The ToF principle is illustrated in Figure 2.2.


**Figure 2.1: SICK LMS200 laser range finder. (photo credit: SICK AG)**

**Table 2.1: Technical specification of the SICK LMS200 laser range finder. (source: SICK AG, June 2009)**

| | |
|---|---|
| **Light source** | Infrared (905 nm) |
| **Field of view** | 180° |
| **Scanning frequency** | 75 Hz |
| **Operating range** | 0 - 80 m |
| **Angular resolution** | 0.25, 0.5 and 1° |
| **Resolution** | 1 mm |
| **Statistical error** | 5 mm |
| **Interfaces** | Serial (RS-232, RS-422) |
| **Power** | 24 VDC, 20 W |
| **Weight** | 4.5 kg |
| **Dimensions** | 156 × 155 × 210 mm |

In many implementations the single-beam laser range finder operates by shooting a laser beam on a rotating mirror which sweeps it across the scanning area. Figure 2.3 illustrates this operation in a few steps. First, the laser beam sweeps counter-clockwise from one side of the scene ($a$) to the other side of the scene ($b$ and $c$ respectively) where the scene and an object are scanned. The red line represents the laser beam, the green circles represent the scene and the obstacles and the figures with blue dots represent the data points gathered by the laser range finder.

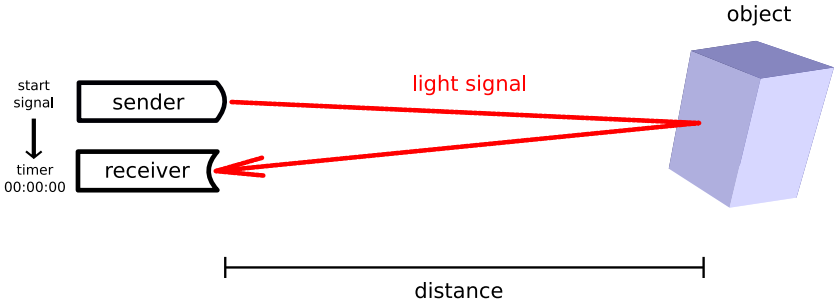


**Figure 2.2: The time of flight principle.**

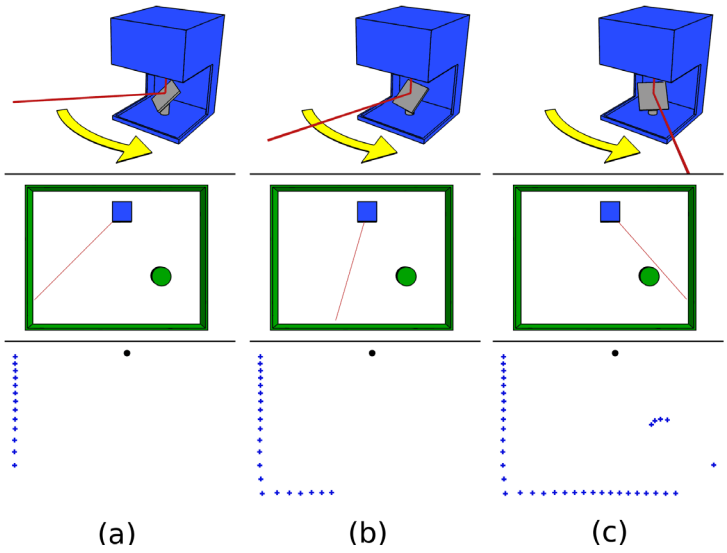


**Figure 2.3: The laser range finder in operation.**

For a 3D map building task, the laser range finder is usually equipped with an extra actuator such as a servo motor so that it can be tilted within the vertical direction in order to extend the scanning range. As a result, the full 3D scan of the scene is possible. Figure 2.4 illustrates the tilting mechanism for the 3D volume scanning.

### 2.3.2 Implementations and Applications

Many 3D map building applications were successfully implemented using a single-beam laser range finder due to its high accuracy and long operation range. Surmann et al. did extensive research and produced a number of scientific papers about six degrees of freedom simultaneous localization and mapping (SLAM) with the SICK laser range

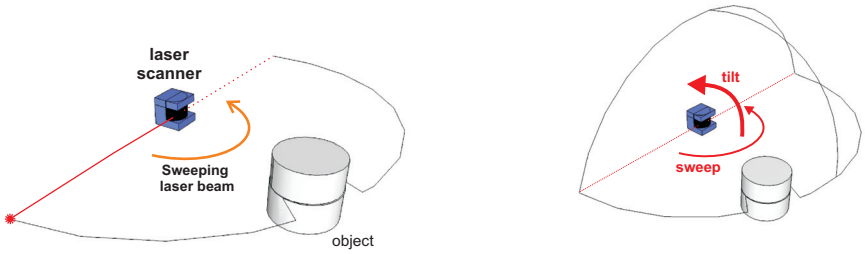


**Figure 2.4: A normal sweeping plane of a 3D laser range finder (left) and the extended vertical scanning using a tilting mechanism (right).**

finder [Surmann *et al.* (2006), Nüchter *et al.* (2005)]. The laser range finder with a tilting mechanism is fitted on a mobile robot (Figure 2.5) that drives in the indoor and outdoor environment. Some sample scans derived from such systems are shown in Figure 2.6.

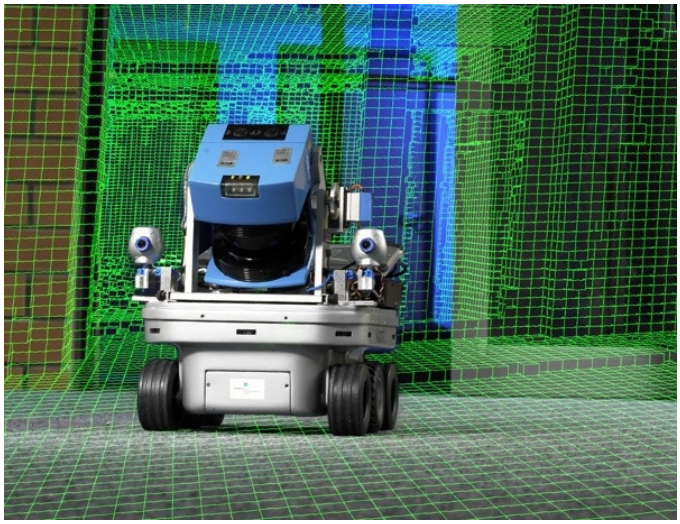

**Figure 2.5: A mobile robot equipped with a SICK laser range finder and a tilting mechanism. (photo credit: Fraunhofer Institute AIS)**

Beside the use of the single-beam laser range finder among the robotic research community, there are also commercially available products that are specifically designed for large size 3D scanning of buildings and interiors. These scanners provide more sophisticated solutions for detailed 3D scans by sweeping the laser scanning plane across

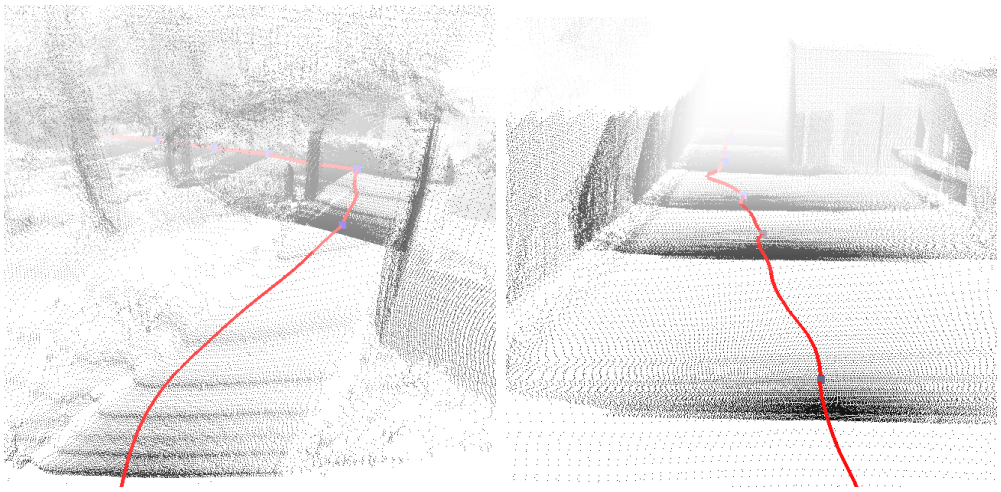

**Figure 2.6: Sample scenes acquired by the laser range finder. (photo credit: Fraunhofer Institute AIS)**

a full 360 degree rotation. An on-board computer is integrated into the scanner for controlling and data logging purposes. The color information that is not available from the laser range finder can be inserted to the acquired 3D model using a high resolution digital camera. By acquiring several 360 degree scans of a scene from several locations within the environment, a complete 3D model of the environment can be constructed using specific post-processing software. An example of such a commercial laser range finder product is the FARO Photon laser scanner shown in Figure 2.7. It has a scanning range of 120 meters with an error of $\pm 2$ mm and it completes a full 360 degree scan in 233 seconds. Figure 2.8 shows an example of a 3D model of a cathedral built by the FARO Photon scanning system. The cloud of 3D points is collected by the laser scanner and a digital camera is used to provide the texture and color information to the finished model.


**Figure 2.7: The FARO Photon laser scanner which is specifically designed for the capturing of large scenes with optional color enhancement using a digital camera. (photo credit: FARO Technologies Inc.)**

**Figure 2.8: An example of a 3D model of a cathedral using the FARO Photon laser scanner. Note that the chairs and other small objects within the model are inserted manually afterwards using simple 3D polygons. (photo credit: FARO Technologies Inc.)**

## 2.4 PMD Camera

The Photonic Mixer Device or the PMD camera represents an important technological advancement in 3D visual sensing. It realizes a 3D scanning method using the time of flight concept. Instead of a rotating mirror solution found on the laser range finder hardware, the PMD camera illuminates the whole scene with its modulated light source and percepts the bounced back signal with a 2D array of diodes or PMD pixels (Figure 2.9). This enables the PMD system to percept a 3D volume without the need of moving mechanisms, which greatly reduces the hardware size and weight. An example of a PMD camera, PMD[vision]® CamCube, is shown in Figure 2.10 and its technical specification is listed in Table 2.2. An example output of a PMD camera is shown in Figure 2.11

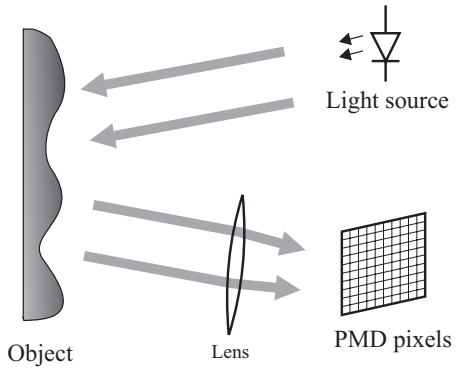


**Figure 2.9: PMD camera working principle.**

### 2.4.1 Working Principle

The PMD system consists of the sender, called illumination modules, and the receiver, which consists of an array of PMD pixels. The capture process starts by emitting light pulses from the illumination modules; the light pulses then hit the object, bounce back through the optical lens and are projected on the PMD pixels. The distance to the object is then determined according to the phase shift of the light signal. The phase shift which is proportional to the distance to the object is determined by the following equation:

$$\varphi = arctan(\frac{A_1 - A_3}{A_2 - A_4}) \tag{2.2}$$

**Figure 2.10: An example of a PMD camera (PMD[vision]® CamCube) equipped with illumination modules. (photo credit: PMDTechnologies GmbH)**

**Figure 2.11: An example of a 3D image acquired by a PMD camera. (source: PMD Technologies GmbH, 2008)**

where $A_1$, $A_2$, $A_3$ and $A_4$ are four samples from the PMD pixel measurement with 90 degree phase shift from each other. The distance $d$ is then calculated by:

$$d = \frac{c\varphi}{4\pi f_{mod}} \tag{2.3}$$

where constant $c$ is the speed of light and $f_{mod}$ is the modulation frequency of the illumination module. With the wavelength of the modulation frequency $\lambda_{mod}$, the maximum range of detection for the PMD camera $d_{max}$ is determined by:

$$d_{max} = \frac{\lambda_{mod}}{2} \tag{2.4}$$

**Table 2.2: Technical specification of the PMD[vision]® CamCube. (source: PMDTechnologies GmbH, 2009)**

| | |
|---|---|
| **Light source** | Infrared |
| **Sensor resolution** | 204 × 204 pixels |
| **Field of view** | 40° vertical |
| | 40° horizontal |
| **Scanning frequency** | 25 Hz |
| **Operating range** | 0.3 - 7 meters |
| **Distance resolution** | <3 mm |
| **Interfaces** | USB2.0 |
| **Power supply** | 12 VDC |
| **Weight** | <1 kg |
| **Dimensions of camera module** | 194 × 80 × 60 mm |

Although the PMD camera provides an elegant and robust solution for 3D measurement tasks, the current hardware still has some considerable hindrances. Two main drawbacks of the PMD camera at the moment are the low resolution of the PMD sensor and the lack of color information. Currently, the resolution of a PMD camera can be as low as $32 \times 64$ pixels. However, the resolution has been increased rapidly due to the growing demand for the PMD sensor in the past few years. The PMD camera with a VGA resolution of $320 \times 240$ pixels or higher can be expected to be available soon.

### 2.4.2 Implementations and Applications

The PMD camera is currently gaining more acknowledgement in the industry and automotive applications, where size and performance are the important factors. The PMD camera is also gaining more attention among machine vision researchers. In the work conducted by Joochim *et al.* (2009), a PMD camera is mounted on a mobile robot (Figure 2.12) and is used as a core sensor to gather 3D information for a map building process. Figure 2.13 shows the result of a 3D model derived using the PMD camera, which was enhanced with color information obtained from an accompanying 2D camera.

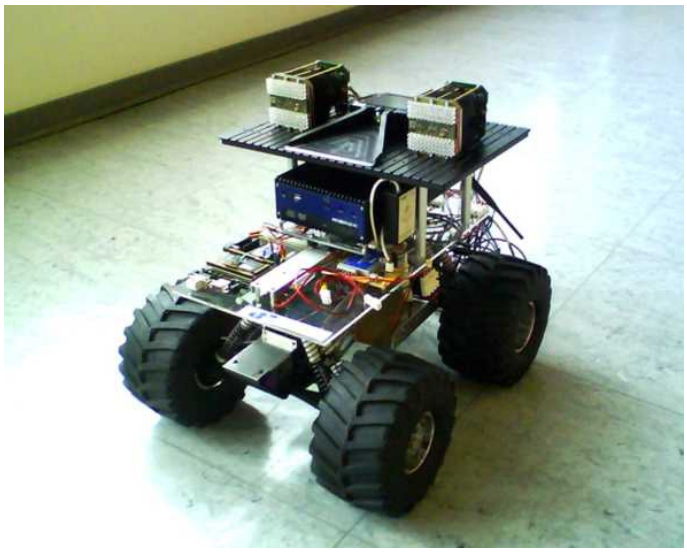

**Figure 2.12: PMD[vision]® A2 mounted on a mobile robot.**

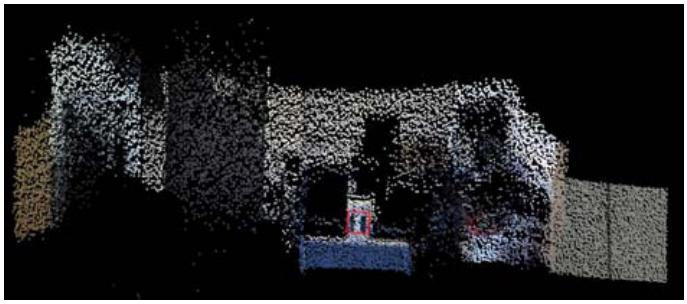

**Figure 2.13: A 3D model obtained using depth information from a PMD camera.**

## 2.5 Stereo Camera

A stereo camera is hardware that produces 3D images using the triangulation technique. It consists of two cameras whose optical axes are parallel to each other and are separated from each other by a certain offset distance called *baseline*. The baseline distance makes two cameras see the same object from two different viewpoints. As a consequence the *disparity value* can be determined. The disparity value together with the other information such as focal length can then be used to calculate the depth information. This whole process is known as *triangulation*. Figure 2.14 shows some example images acquired by a stereo camera.

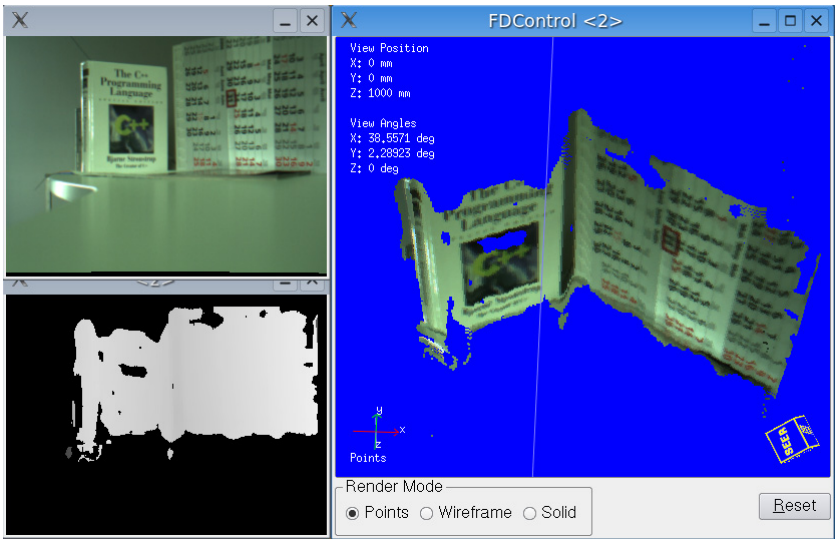


**Figure 2.14: Some example images acquired by a stereo camera. The upper left image is the color image, the lower left image is the disparity image and on the right hand side is the 3D image.**

### 2.5.1 Working Principle

The principle of stereo triangulation is shown in Figure 2.15. Two cameras are placed alongside each other with the baseline $b$ between their optical centers. A general point $P(X, Y, Z)$ appears on the image plane of each camera at two different positions $(X_1, Y_1)$ and $(X_2, Y_2)$. The distance between these two points is called disparity and it is inversely proportional to the distance $Z$ of the point $P$. By assuming that the x- and z-axes of both cameras are parallel and the optical center of both cameras are located on the same horizontal baseline, the disparity value $D$ can be calculated as:

$$D = X_1 - X_2 \tag{2.5}$$

Consider similar triangles, $PF_1F_2$ and $PC_1C_2$

$$\frac{Z - f}{Z} = \frac{b - (X_1 - X_2)}{b} \tag{2.6}$$

thus

$$Z = \frac{fb}{X_1 - X_2} = \frac{fb}{D} \tag{2.7}$$

where

| | |
|---|---|
| $f$ | focus length |
| $b$ | baseline |
| $C_1$ | location of point P projected on image plane 1 |
| $C_2$ | location of point P projected on image plane 2 |
| $F_1$ | focal point of image sensor 1 |
| $F_2$ | focal point of image sensor 2 |

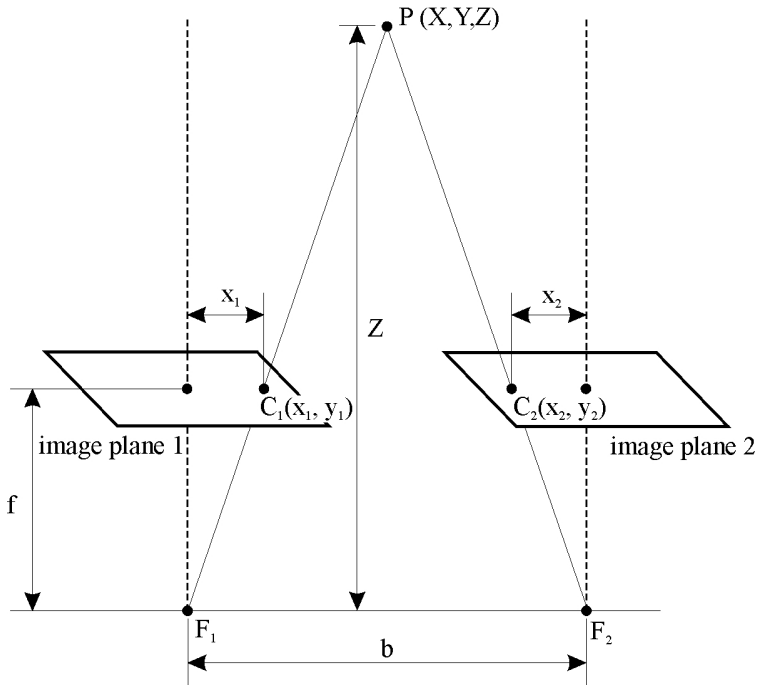


**Figure 2.15: The working principle of the stereo triangulation technique.**

By using the triangulation technique as described, the recovery of the depth information is straightforward. The actual complicated and time consuming part is the determination of the exact position of the points $C_1$ and $C_2$ on the left and the right image. This problem is known as the *correspondence problem*. Several techniques are used to deal with this problem and one of the techniques that can be efficiently implemented for a real-time system is the area-based correlation technique. A brief introduction to this technique is given in the following section.

## Area-based Correlation

One widely used technique for solving the correspondence problem is the area-based correlation technique. Instead of comparing a single point or pixel in the scene, small searching windows among images are used for the correlation process. This technique provides an effective way to solve the correspondence problem where noticeable visual features or highly contrasting features are not available. Figure 2.16 illustrates the working principle of the area-based correlation operation. Firstly, the area of interest in the first image defined by a search window is chosen at one fixed position. Then on the second image another search window is translated to every position within the search area. At each position the pixel intensities within two search windows are compared for their similarity and the correspondent point is located where the maximum similarity is found.

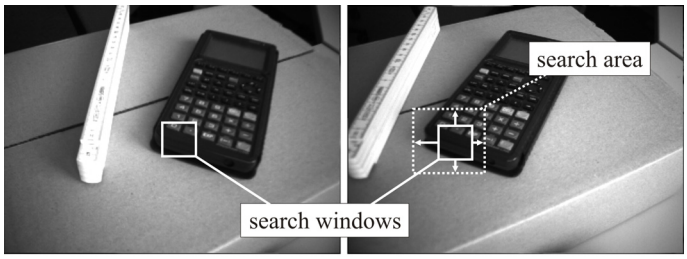


**Figure 2.16: The search windows and the search area of the area-based correlation technique.**

In many implementations an epipolar line is used. The epipolar line is a line that presents the possible location of the correspondent feature on the second image in respect to a distinctive point found in the first image. Thus the search dimension is reduced from two to one. This technique is widely used to increase the stereo vision performance and reduce the computational time. Figure 2.17 shows the geometry of the epipolar line for a stereo camera.

### 2.5.2 Implementations and Applications

Several works present the results of 3D map building using a stereo camera. In Sáez & Escolano (2004) a stereo camera is used to observe feature points within the test

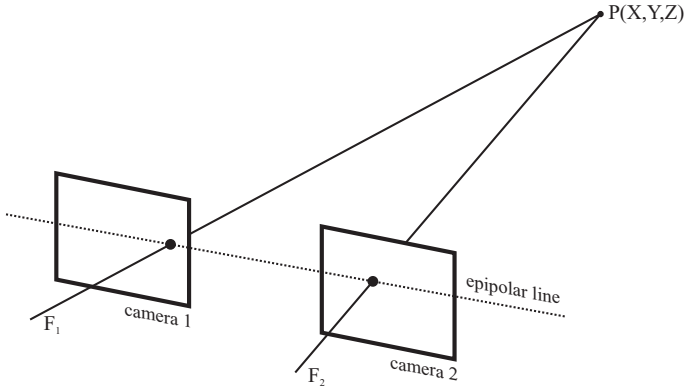


**Figure 2.17: An epipolar line spanning two image planes.**

scene which are used for the indoor map building of an environment with flat terrain. In Garcia & Solanas (2004) a stereo camera is used for the map building task where the motion of the camera can be estimated and some results of the captured 3D model are presented.

## 2.6 Comparison of 3D Visual Sensors

Three types of 3D visual sensors mentioned in this chapter use different kinds of technology and possess different working principles. This section compares these sensors according to certain aspects including the coverage area, precision, visual detail, mobility, real-time performance, affordability and the energy saving factor. A short analysis about each aspect is given below.

### Coverage Area

The coverage area or the field of view indicates the size of the 3D volume that can be acquired by the 3D visual sensor. This ability is important for map building tasks since gathering of the map data is faster when the sensor can see a large portion of the environment at a time. The coverage area is mainly defined by the working principle of the sensor. In Figure 2.18, it can be seen that the observed area of the stereo camera and the PMD camera are defined by the field of view of the optical lens whereas the observed area of the laser range finder depends on the sweeping and the tilting angles.

- **Laser range finder:** The coverage area of a laser range finder depends on the sweeping angle and the tilting mechanism.

- **PMD camera:** For the PMD camera the coverage area depends on the field of view of the equipped optical lens.
- **Stereo camera:** For the stereo camera the coverage area depends on the field of view of the equipped optical lens.

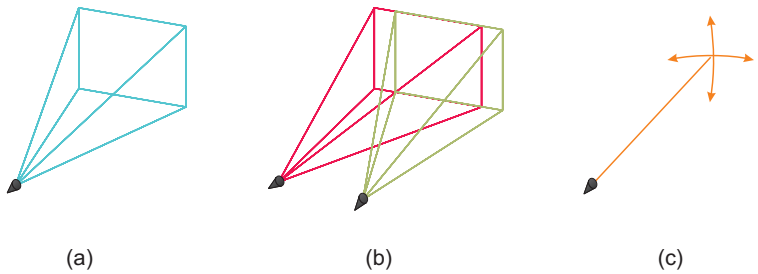


**Figure 2.18: Comparison of the working principles of the 3D visual sensors (a) PMD camera (b) stereo camera and (c) laser range finder.**

## Precision

The measurement principle has a big effect on the precision of the depth measurement. The time of flight sensor can sense the depth of the scene directly at high accuracy using its own light source while the stereo camera depends on the triangulation, which relies heavily on the texture and the light conditions of the observed environment.

- **Laser range finder:** The precision of the measurement depends on the quality of the laser beam signal and the timing circuit.
- **PMD camera:** The precision of the measurement depends on the quality of the light signal which is directly affected by the power of the illumination module and the ambient light.
- **Stereo camera:** The precision depends on the resolution of the camera sensor and the algorithm used to determine the depth information as well as the light conditions.

## Visual Detail

The visual detail indicates the ability of the 3D sensor to acquire information about the geometric description and the texture of the environment, which is important in an application such as photorealistic map building where these information need to be acquired at high resolution.

- **Laser range finder:** The resolution of the geometric scan depends on the finest movement of the rotation and the tilting mechanism. Moreover, since texture information cannot be acquired directly by the laser range finder a digital camera is needed to capture the texture and color of the objects in the scene. Therefore the quality of the texture depends on the image resolution of the additional digital camera.
- **PMD camera:** The resolution of the geometric scan and texture image depends on the spatial resolution of the PMD sensor. At the moment the resolution of the PMD sensor is still low and is also lacking of color information.
- **Stereo camera:** The resolution of the geometric scan and texture depends on the spatial resolution of the image sensor. Commercial stereo camera hardware offers a high resolution sensor and therefore good geometric resolution can be obtained using sophisticated algorithms at the expense of calculation time. The texture information is directly available from the high resolution color image acquired from the left and right camera of the stereo camera.

## Mobility

Mobility indicates the size and portability of the 3D visual sensor. This is important if the target system needs high mobility and a small hardware size.

- **Laser range finder:** The laser range finder has the largest size and weight compared to the other 3D visual sensors described in this work. It also contains moving mechanical parts, which is undesired by many applications.
- **PMD camera:** The PMD needs an illumination module and therefore makes it slightly larger in size and weight compared to the stereo camera.
- **Stereo camera:** The stereo camera solution is the most compact in size and weight compared to the other two 3D sensors.

## Real-time

In order to build a 3D map in real-time the 3D visual sensor should be able to capture a portion of the scene at high speed, i.e. a high output or frame rate is desired to catch up with the rapid movement of the target platform.

- **Laser range finder:** The laser range finder solution requires a relatively slow tilting mechanism and therefore it is not suitable for real-time application where a portion of 3D volume is needed to be captured at high speed.
- **PMD camera:** The PMD camera is capable of producing 3D images of the scene at high speed.
- **Stereo camera:** The stereo camera is capable of producing 3D images of the scene at high speed.

## Affordability

This reflects the availability and the price of the sensor. This factor could have a big impact if multiple sensors are needed in one design.

- **Laser range finder:** The laser range finder is available from several manufacturers and it can be obtained in the moderate to high price range.
- **PMD camera:** The PMD camera is available from a limited number of manufacturers and the price is relatively high compared to the other two sensors.
- **Stereo camera:** The stereo camera is available from many manufacturers. It can also be easily constructed from scratch using simple off-the-shelf hardware and therefore it is the most affordable system compared to the other two sensors of the three.

## Energy Saving Factor

This reflects the energy usage of the sensor. This factor is important for target system that has a limited power capacity. If the sensor consumes small amount of energy then the operation time of the target system is prolonged.

- **Laser range finder:** The laser range finder consumes moderate to high amount of energy. The energy consumption is rated at 20 Watts, although this number could get even higher when the tilting mechanism is used.

- **PMD camera:** The PMD camera consumes low to moderate amount of energy. The power consumption ranges from 5-20 Watts depending on the power of the illumination modules.

- **Stereo camera:** The stereo camera consumes only small amount of energy since there are no active elements such as illumination modules or moving mechanism used. Only less than 1 Watt is being drawn from the IEEE1394 interface during the operation time.

Table 2.3 summarizes the information of all three visual sensors according to the mentioned aspects. It can be seen that the laser scanner has the longest measurement range and also a good precision. The PMD camera performs as well as the laser scanner and it has a smaller size and weight, which makes it an all-around versatile 3D sensor. The stereo camera has an outstanding advantage, namely the high 2D image resolution. The stereo camera is also a 3D visual sensor with the smallest size and has the lowest power consumption of all three sensors.

The disadvantage of the laser scanner is the moving mirror mechanism and the need of the tilting mechanism as well as its size and weight. The PMD camera is a very good all-around 3D sensor but until now the sensor resolution is still too low for many image processing tasks and it also lacks color information. The PMD is also not yet available in the mass market, which makes it difficult to obtain. The stereo camera uses the triangulation technique, which requires an extra post-processing task. However there are already some products with an embedded FPGA-chip that takes care of depth calculations on the camera in real-time. The stereo camera is relatively cheap and widely available, making it an optimal choice as the 3D visual sensor for the multi-camera hardware.

**Table 2.3: Comparison between different types of 3D sensors.**

| | **Laser range finder** | **PMD camera** | **Stereo camera** |
|---|---|---|---|
| **Coverage area** | Large FoV (max 180°), long range (80 m) | Moderate FoV (max 40°), short range (40 m) | Moderate FoV (max 60°), short range (10 m) |
| **Precision** | High (<5 mm) | Moderate - high (3-10 mm) | Moderate - high (5-20 mm) |
| **Visual detail** | No 2D image | Low resolution 2D grayscale image (204×204 pixels) | High resolution 2D color image (640×480 pixels) |
| **Mobility** | Large and heavy, contains moving parts (1.5 kg) | Small and light (ca. 1 kg) | Small and light (320 g) |
| **Real-time** | Requires tilting to capture 3D scene (<0.5 Hz) | Instant 3D scene capturing (25 Hz) | Instant 3D scene capturing (30 Hz) |
| **Affordability** | Commercially available, expensive | Not widely available, expensive | Widely available, cheap |
| **Energy saving** | Poor | Moderate - poor | Good |

## 2.7 Selection of the 3D Visual Sensor for the Multi-Camera Unit

The main contribution of this thesis work is the multi-camera unit (MCU). As the name implies, the MCU is constructed using multiple 3D visual sensors. The selection of the 3D visual sensor for the MCU hardware is done based on the circumstances of the applications which, in this case, are the real-time motion estimation and the map building tasks. The main requirements for the 3D visual sensor for these tasks are:

- Real-time or fast image capture time in order to deal with the constant movement of the MCU, which is placed on a mobile robot or used as a hand-held device.

- High mobility, which means the MCU size should be small and light so that it can be easily integrated into a small hardware platform or held in an operator's hand.
- High resolution (both texture and geometric scanning resolution) to ensure the quality of the map.
- Affordability, since multiple sensors are needed for the design of the MCU.
- Good measurement precision to ensure the 3D motion estimation result.
- Moderate coverage area. The MCU design eliminates the need for a large field of view camera by combining information from multiple cameras which, in several cases, works better compared to the single camera solution equipped with a large field of view lens.
- Low power consumption, which prolong the operation time of the target system with limited amount of power supply.

Figure 2.19 illustrates the properties of all three 3D visual sensors according to their relative performance. It can be seen that the stereo camera is the best candidate for the MCU hardware since it covers most of the requirements with high scores and therefore it is selected as the 3D visual sensor for the current implementation of the MCU.

The second-most suitable sensor is the PMD camera. However the low image resolution, the lack of color information and the high price prevent it from being chosen for the MCU hardware at the moment. Nevertheless, once these drawbacks are eliminated, the PMD camera is likely to replace the stereo camera as an equal or better 3D visual sensor for the MCU system.

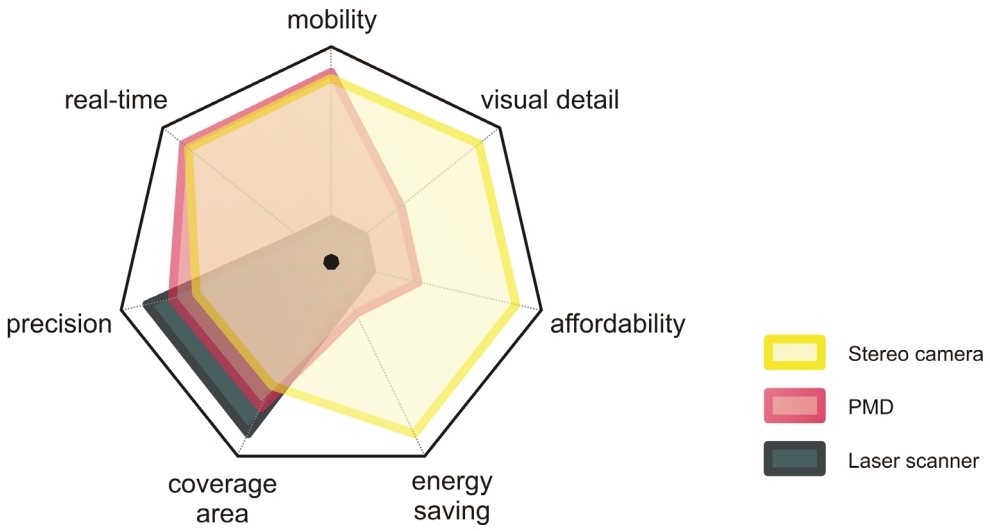


**Figure 2.19: Comparison between different types of 3D sensors for 3D motion estimation and map building task.**

## 2.8 Conclusion

This chapter gives a brief description of three different 3D visual sensors. Two kinds of depth recovery techniques are explained including the time of flight operation used in the laser range finder and the PMD camera as well as the triangulation used in the stereo camera. The advantages and disadvantages of each 3D sensor are explained and some example applications from each 3D sensor are shown. The comparison between three kinds of sensor are given and the results show that the laser range finder is the biggest in size and weight and is not suitable for the real-time 3D map building task or for a small hardware platform. The PMD camera is a robust and compact piece of hardware but it still cannot produce high resolution images for some image processing tasks and the color information is also missing. The stereo camera is well-balanced according to the considered abilities and it fulfils most of the requirements for the 3D motion estimation and map building tasks. It is therefore selected as the 3D visual sensor for the multi-camera unit.

# Chapter 3

# The Multi-Camera Unit

***Introduction — The Multi-Camera Unit — Hardware Description — Software Description — Elimination of Motion Ambiguity using Multi-Camera Systems — Multi-Camera Hardware Simulation — Multi-Camera Hardware Calibration — Conclusion***

## 3.1 Introduction

The multi-camera unit (MCU) is the main contribution of this thesis work. It introduces a unique arrangement of three stereo cameras to overcome the motion ambiguity problem which exists in the single camera system. This results in a new visual odometry device that is highly sensitive to 3D movement. At the same time it also acts as a 3D sensor that produce high density 3D information at a high acquisition rate. This chapter explains the advantage of the multi-camera configuration over the single camera configuration as well as its hardware and software components. A software simulation of the multi-camera system is also implemented to investigate its performance. Finally, a calibration method for the MCU hardware is explained at the end of the chapter.

## 3.2 The Multi-Camera Unit

The multi-camera unit (MCU) was first introduced in Netramai & Roth (2007). The MCU is designed to improve the sensitivity of the real-time 3D motion detection as well as to increase the data acquisition rate for the 3D map building task. The MCU uses three stereo cameras whose optical axes point perpendicular to each other in the direction of the x-, y- and z-axes. By doing so, the accurate 3D motion of the MCU can be efficiently detected since the motion ambiguity in one camera is always compensated by the other two cameras. Figure 3.1 shows the current hardware implementation of the MCU.

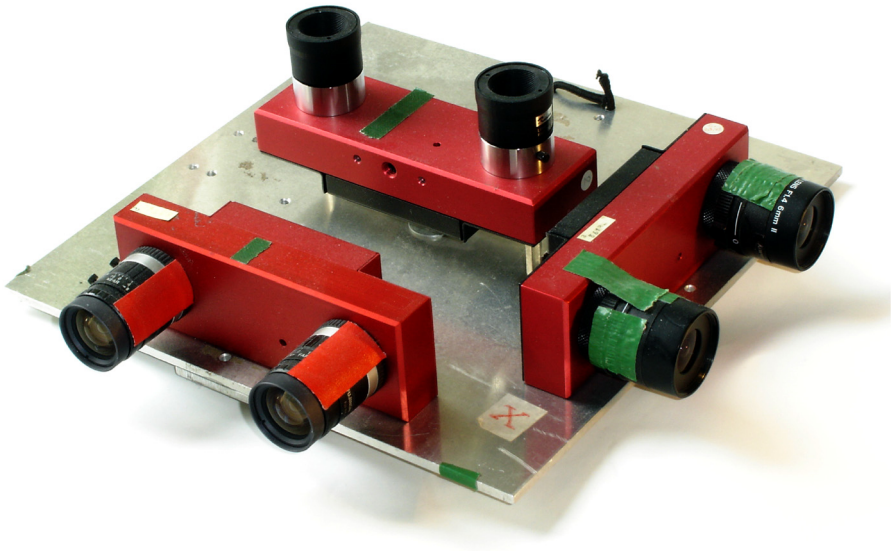

**Figure 3.1: The multi-camera unit.**

### MCU System Overview

The overview of the MCU system for the 3D map building task is shown in Figure 3.2. Three stereo cameras are used to acquire a set of 2D and 3D images from the environment which are used for the motion estimation and map building process. An example of the image set acquired from the MCU is shown in Figure 3.3. The main processes

of the 3D map building task include feature point extraction, motion estimation and real-time map building. The result of these operations is a 3D photorealistic map of the observed environment as well as the detailed 3D trajectory of the MCU during the map building process.

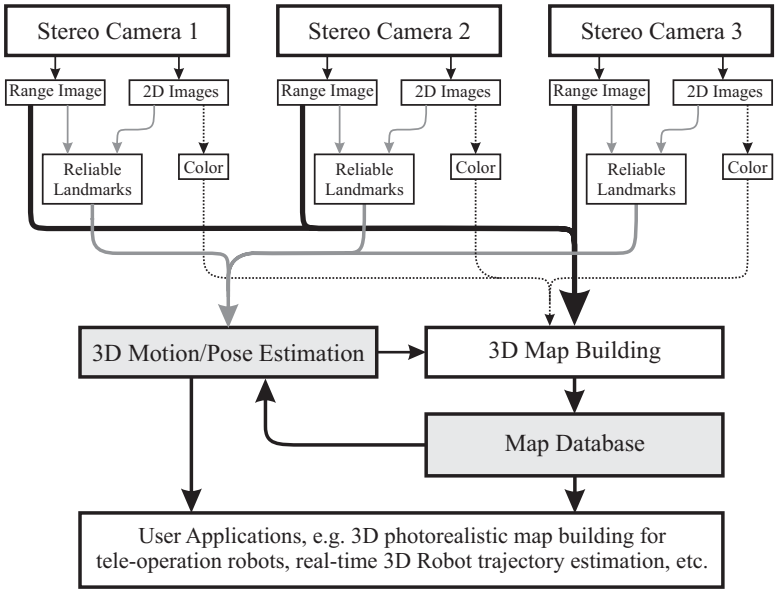


**Figure 3.2: An overview of the MCU system for 3D map building tasks.**

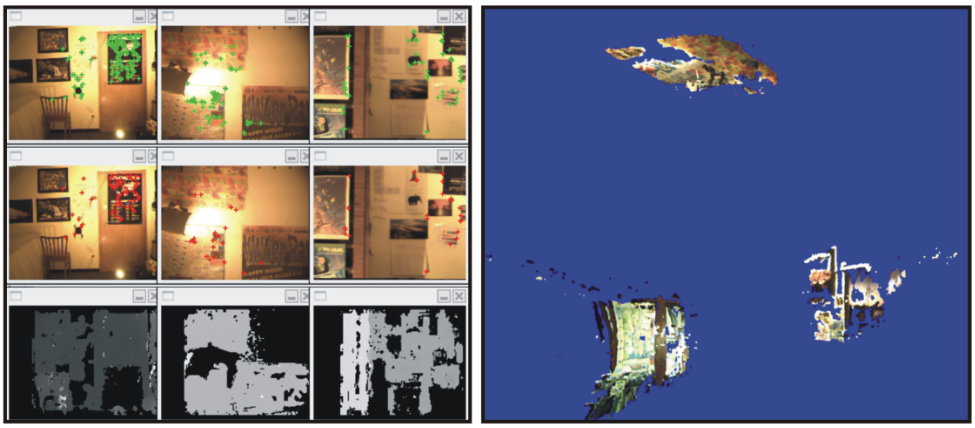

**Figure 3.3: Example images acquired from the MCU which includes (left) 2D images and (right) clouds of 3D points.**

The detailed descriptions of the 3D motion estimation process and map building process are given in Chapter 4 and Chapter 5 respectively.

## 3.3 Hardware Description

The MCU consists of three pairs of stereo cameras. The cameras are mounted on a rigid aluminum platform and are connected to the host computer via the IEEE1394 interface. The mechanical drawing of the MCU is given in Appendix A.2. The following section contains the description of the stereo cameras used in the MCU system.

### 3.3.1 The Stereo Camera

The current MCU hardware consists of three pairs of STH-MDCS stereo cameras from Videre Design (Figure 3.4). Videre Design supplies a ready-to-use solution which consists of a stereo camera and the compatible software library (Small Vision System – SVS library) that efficiently computes the depth information using the area-based correlation algorithm.

**Figure 3.4: The STH-MDCS stereo camera.**

The STH-MDCS stereo camera consists of two identical CMOS sensors which are separated by a 90 mm horizontal baseline. Both cameras are equipped with the identical lenses (e.g. identical construction, same focal length, etc.). The stereo camera produces two video outputs, one from the left camera and the other from the right camera. These video signals are interlaced by the hardware circuit on the camera to produce a single video stream which is transmitted to the host computer via the IEEE1394 interface. The adjusting of the camera parameters is also done over the IEEE1394 interface, where parameters such as exposure, gain and capture size can be manipulated. The technical detail of the STH-MDCS camera is listed in Table 3.1.

**Table 3.1: Technical specification of the STH-MDCS stereo camera.**

| | |
|---|---|
| Image sensor | 1/2" format color CMOS (Micron MT9M001) |
| | $1280 \times 960$ active pixels, progressive scan |
| Image formats (pixels) | $320 \times 240$, $640 \times 480$, $1280 \times 960$ (max) |
| Frame rates | 3.75, 7.5, 15, 30 Hz |
| Gain | 0 - 18 dB |
| S/N | $> 45$ dB, no gain |
| Power | $< 1$ W |
| Synchronization | Internal: pixel-locked |
| | External: 60 $\mu$s |
| Lens | 6 mm, F1.2, C-mount (Fujinon) |
| | 6 mm, F1.4, C-mount (Kowa) |
| | 8 mm, F1.2, C-mount (Rainbow) |
| Size (excluding lenses) | $44 \times 132 \times 33$ mm |
| | (see Appendix A.2 for more detail) |
| Weight | ca. 320 g including lens |
| Stereo baseline | 90 mm |
| SVS software | Running on Linux kernel 2.4, 2.6 |

In addition, the error characteristic of the stereo camera was tested and the results are available in Appendix B.

### 3.3.2 The Host Computer

The host computer runs the software pieces which are necessary for the operation of the MCU and related tasks. The data from the MCU is transferred to the host computer via three IEEE1394 PCI interface cards where one card is directly connected to one stereo camera in order to obtain the maximum data transfer rate. A powerful graphic card is installed for 3D visualization tasks. The specification of the host computer is listed in Table 3.2.

**Table 3.2: Technical specification of the host computer.**

| | |
|---|---|
| CPU | Intel Core 2 Quad processor Q6600 running at 3.0 GHz (overclocked) |
| System board | Gigabyte EP35-DS3R |
| System memory | 4 GB DDR2-800 SDRAM |
| Graphic card | Nvidia GeForce 9800GT with 1 GB video RAM |
| Frame grabber cards | Three Texas Instruments TSB43AB23 IEEE-1394a PCI cards |
| Operating system | Linux 32-bit with kernel version 2.6.27 |

## 3.4 Software Description

This section describes the basic software components of the MCU system including the software library that controls the stereo cameras, the auxiliary functions that provide the necessary functionalities to the MCU system and the graphic user interface that gives an easy way to control and adjust the parameters of the MCU system during run-time.

### 3.4.1 SVS Software Library

The SVS library is a pre-compiled software library that communicates between the stereo camera and the host computer. It controls the parameters and operation of the stereo camera such as image resolution, frame speed, exposure setup, image grabbing, etc. The library also takes care of the stereo depth calculation using highly-optimized machine code for real-time performance. The SVS library is supplied by Videre Design, which is the producer of the STH-MDCS stereo camera. The main functions of the SVS software library are:

- Control of the stereo camera parameters including initialization routine for the stereo camera. The initialization includes setting the image rectification, 3D image transformation matrix and other camera parameters such as frame speed, frame size and relevant stereo calculation parameters.

- Grabbing the images from the stereo camera and storing them in the image buffer on the host computer.
- Calculating the stereo image on the host computer using the images from the left and right cameras.

More information about the SVS software library is available in the software manual [Konolige & Beymer (2007)].

### 3.4.2 Auxiliary Functions

In order to efficiently manage all three stereo cameras at the same time, the auxiliary functions are written by the author to wrap up the set of functions to operate multiple stereo cameras in one single command. These functions include the synchronization function, the time stamp function and the multi-threaded wrapper function.

#### Synchronization Function

It is important that the images from all three stereo cameras are captured at the same time in order to use them for the time-discrete motion estimation process. The synchronization of the stereo cameras is done at software level. The procedure includes issuing the grab command for all three cameras at the exact same time using the timer-interrupt function. The function first check whether all stereo cameras are ready to grab a new image, and if they are then the grab command is issued to three image grabbing threads with millisecond precision. Each thread then takes care of the image grabbing from each camera separately.

#### Timestamp Function

The timestamp function gives a time stamp in millisecond resolution to the images captured by the MCU. This time information is useful for the calculation of motion parameters such as speed and acceleration.

#### Multi-Threaded Wrapper Function

The wrapper function contains the time consuming tasks of the stereo camera including image grabbing, stereo image calculation as well as the higher lever tasks such as corner detection and noise filtering. These tasks are the main bottlenecks and they can lead to a huge performance degradation if processed in a serial manner. The separate

threads are then distributed over different cores of the multi-core CPU in order to take advantage of the multi-threaded operation.

### 3.4.3 MCU System Graphic User Interface

An easy to use graphical user interface (GUI) is created by the author to be able to control and adjust the parameters of the MCU system during run-time. The GUI contains some useful operations such as start and stop of the MCU system and adjusting the threshold value of various algorithms using the slider bars. The live image streams from all cameras are displayed as well as the 3D visualization of the 3D map being created. Figure 3.5 shows the MCU system GUI during a 3D map building task.

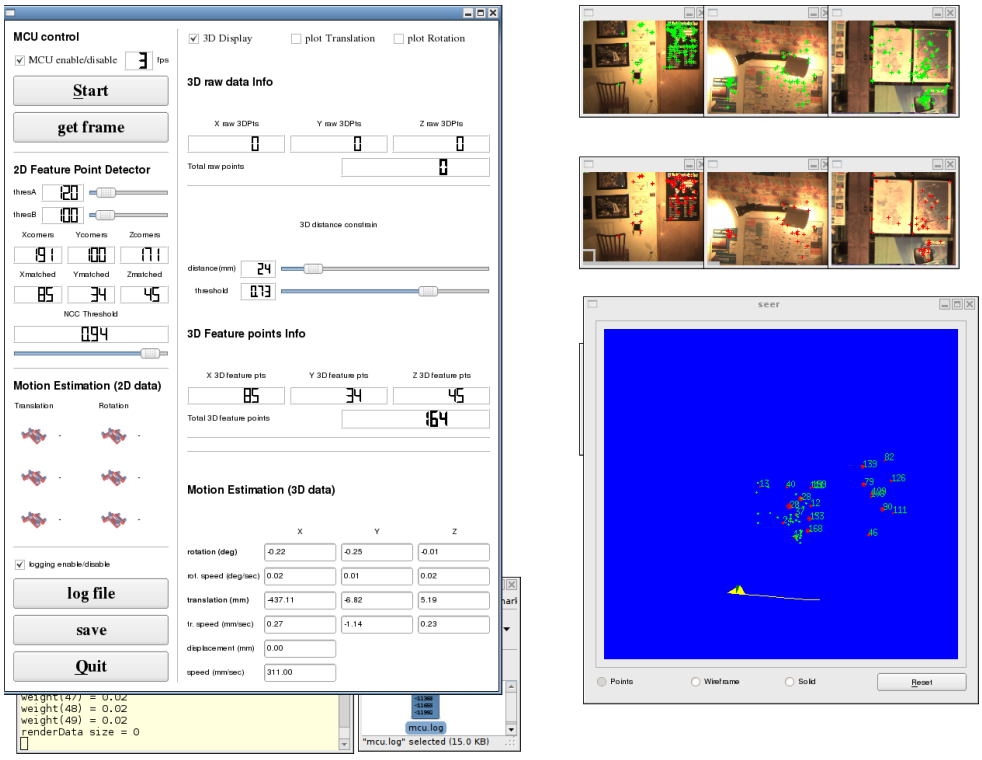


**Figure 3.5: The MCU system graphic user interface. On the left hand side is the main control panel where important parameters can be observed and adjusted at run-time. On the right hand side is the live display of the video streams and the 3D visualization of the 3D map.**

## 3.5 Elimination of Motion Ambiguity using Multi-Camera Systems

One of the main benefits of using a multi-camera system is the elimination of the motion ambiguity. The motion ambiguity is usually found in the single camera system and occurs due to the limited field of view of the optical lens as well as the inaccurate depth estimation according to the error of the stereo image calculation. Two kinds of motion ambiguities are explained in this section: *rotation ambiguity* and *translation ambiguity*. The solutions for these ambiguities using multi-camera systems are also given.

### 3.5.1 Rotational Ambiguity

The rotational ambiguity occurs when a small rotation about the axis which is perpendicular to the optical axis takes place. In this case, the motion field obtained from a small rotation is similar to the motion field obtained from a small translation perpendicular to the optical axis, which leads to the wrong motion estimation result. This situation is illustrated in Figure 3.6. The square object represents the camera or the observer and the arrows-filled area presents the image and the motion field of the feature points within the image. When the observer moves to the left the position of the feature points shift to the right parallel to the observer's movement direction. However, when a small *rotation* around the axis that is perpendicular to the optical axis of the observer is introduced, the similar motion field is obtained. Therefore such small rotation can be confused as translation from the observer's point of view.

To solve this problem, a larger field of view optical lens can be used [Davison & Cid (2004)]. However, using an optical lens with a large field of view can reduce the quality of the stereo calculation if the resolution of the camera sensor is not high enough to accommodate the increased field of view.

The multi-camera system solves this problem by introducing a second camera whose optical axis points in the direction perpendicular to the first camera's optical axis. For example, in Figure 3.7 another camera which points to the right direction of the first camera is introduced which provides one more image to the system. Although the optical axes of both cameras are pointing in the perpendicular direction to the rotation of the observer, it is now easier to distinguish a rotation from a translation. That is, if the motion field on both cameras have similar magnitudes and directions then the detected motion can be confirmed as a rotation.

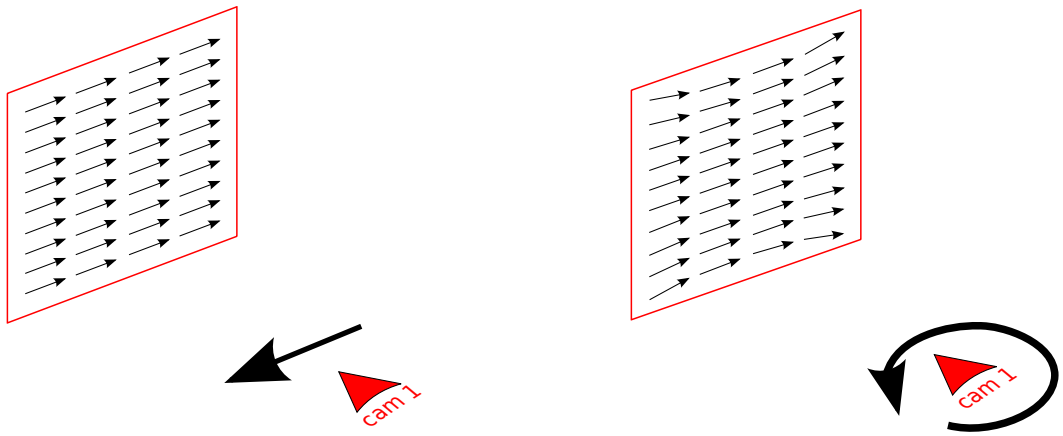


**Figure 3.6: Rotational ambiguity between a small translation (left) and small rotation around an axis perpendicular to the optical axis (right). The motion fields obtained from both cases are similar.**

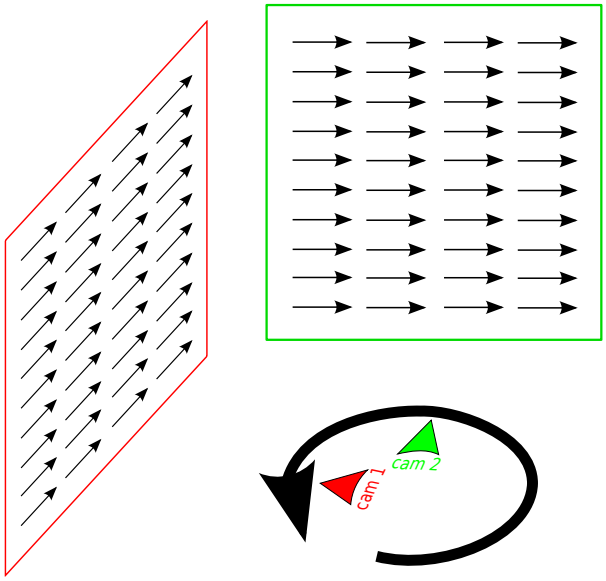


**Figure 3.7: Elimination of the rotational ambiguity using two cameras: $1^{st}$ solution.**

Another solution is to point a second camera along the axis of rotation as shown in Figure 3.8. In this case the rotation can be directly detected by the second camera and hence by using the motion field from both cameras the rotation can be confidently confirmed.

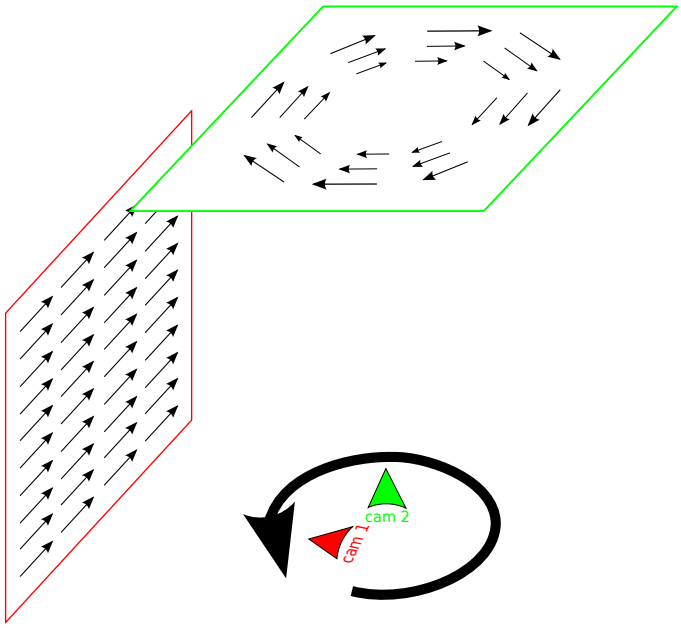


**Figure 3.8: Elimination of the rotational ambiguity using two cameras: $2^{nd}$ solution.**

### 3.5.2 Translational Ambiguity

The translation ambiguity occurs when a small translation along the optical axis takes place. According to the pinhole camera model, the displacement of an object can be detected as shown in Figure 3.9. The object's displacement along a direction that is perpendicular to the optical axis is denoted by $m$ and the displacement along a direction parallel to the optical axis is denoted by $n$. It can be seen that the displacement needed for the object to be seen by the next sensor's element is larger when the object is displaced along the optical axis. This implies that the resolution of the detected translation along the optical axis is coarser than along the perpendicular one. This is also the main cause of stereoscopic depth estimation error, since the uncertainty of the depth calculation is higher when the location is far away from the camera. Such depth

error may cause a sensation of a small translation along the optical axis without the actual movement of the camera. Therefore by using only one camera whose optical axis points to the direction of the translation, the estimation of the translation could suffer from the translation ambiguity.

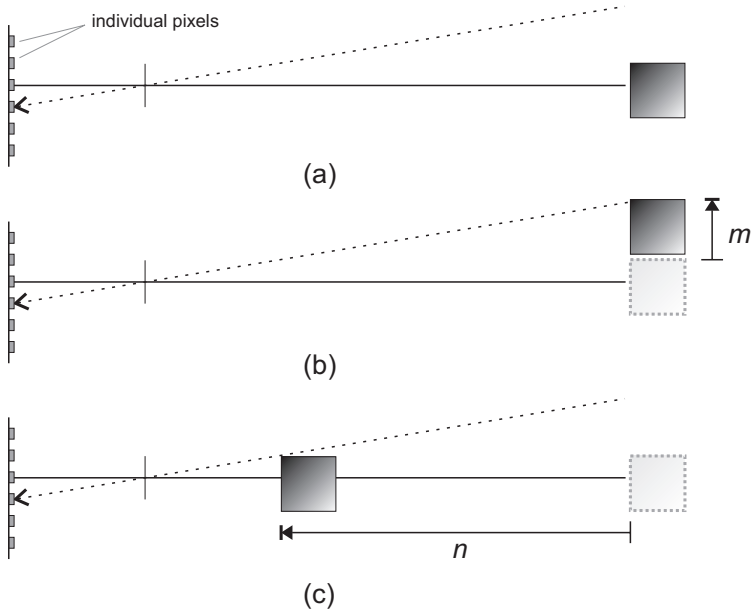


**Figure 3.9: Detection of an object's displacement according to the pinhole camera model. It can be seen that the resolution of the detected translation along the optical axis (c) is coarser than along the perpendicular one (b).**

By using multiple cameras the information from the cameras that are looking in the direction perpendicular to the translation can be used to eliminate the translational ambiguity, since the resolution of the motion detection of the other two cameras are better than the one in trouble. This effect can be seen in Figure 3.10.

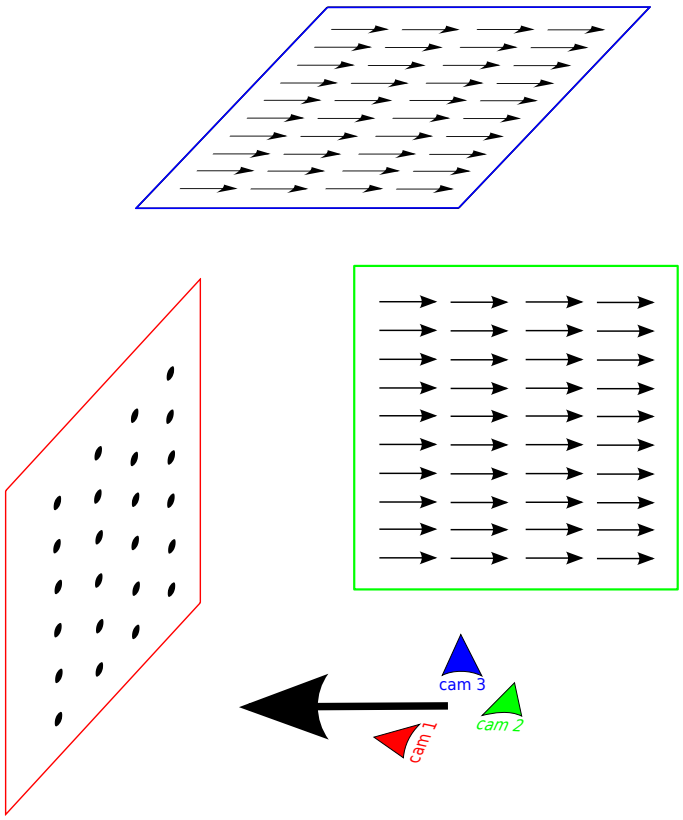


**Figure 3.10: Elimination of translational ambiguity using three cameras. While the camera that points in the movement direction detect hardly any movement, the other two camera can detect the movement just fine and therefore the correct translation estimation can be obtained.**

## 3.6 Multi-Camera Hardware Simulation

A simulation of the multi-camera system is implemented using MATLAB. The goal of this implementation is to evaluate the advantage of the multi-camera system over the single camera system. Two different models were implemented, namely a one camera system and a three camera system. Within each system, the cameras are placed at the origin and are pointed at the direction of the x-, y- and z-axes. For the one camera system only one camera, i.e. the y- or z-axis camera, is considered and for the three camera system all cameras are considered. A set of four data points placed at a distance of two meters is presented to each camera. The complete setup of this simulation is illustrated in Figure 3.11.

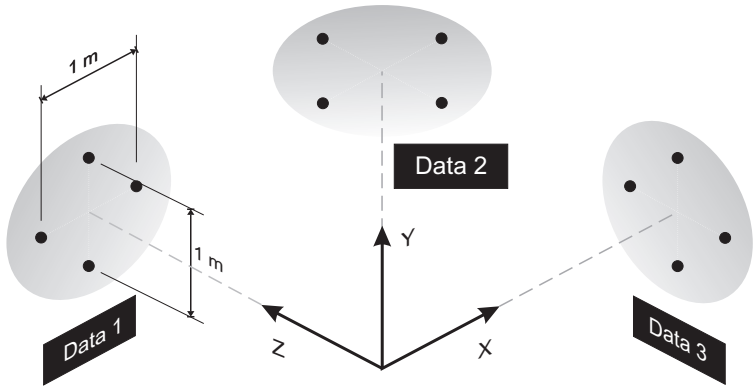


**Figure 3.11: The setup of the multi-camera system simulation.**

The Gaussian noise is added to the data point in order to imitate errors characteristic of the stereo camera. Using the right hand rule the maximum magnitude of the noise $w$ is given by:

$$w = \begin{cases} \pm 10mm & \text{along X and Y axis} \\ \pm 20mm & \text{along Z axis} \end{cases}$$

This value is determined from the real error characteristic of the stereo camera used for this work where the measurement error is highest along the camera optical axis. More detail about the stereo camera error is given in Appendix B.

For this multi-camera system simulation, the motion estimation is done using least square minimization of the Euclidean distance of the marker locations after a pre-defined motion. The same motion estimation algorithm is used for both single camera and multiple camera simulation hardware. The tests include the following pre-defined motions:

- Small translation along optical axis (5 and 10 mm)
- Large translation along optical axis (50 and 100 mm)
- Rotation around optical axis (5, 10 and 50 degrees)
- Rotation perpendicular to the optical axis (5, 10 and 50 degrees)

In the first test a set of small translation along the camera optical axis are introduced to the simulated multi-camera hardware and the result is shown in Figure 3.12. It

can be seen that the estimated translation results using only one camera are heavily corrupted. These corruptions take place due to the stereo calculation error, so the translation along the optical axis is hard to observe correctly. By using three cameras this error is compensated and the translation estimation result is greatly improved. Figure 3.13 confirms the same effect when large translations along the optical axis are used.

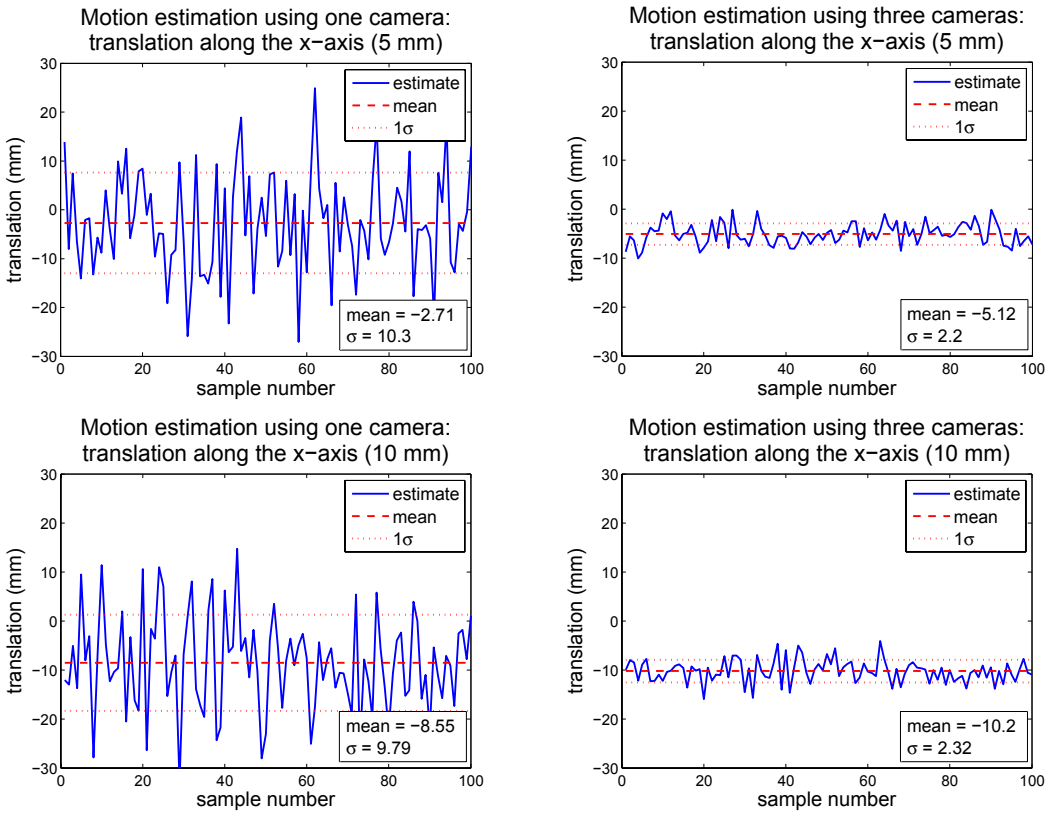


**Figure 3.12: Translation estimation results of the small movements along the x-axis (5 mm and 10 mm) using one camera (left) and three cameras (right).**

Two scenarios for rotation estimation are introduced. In the first case the rotations along the optical axis are used and in the second case the rotations perpendicular to the optical axis are used. Figure 3.14 shows the simulation result of the first case where the single camera system can correctly estimate the rotation with good consistency, which is comparable to the results obtained using a multiple camera system. However, when a

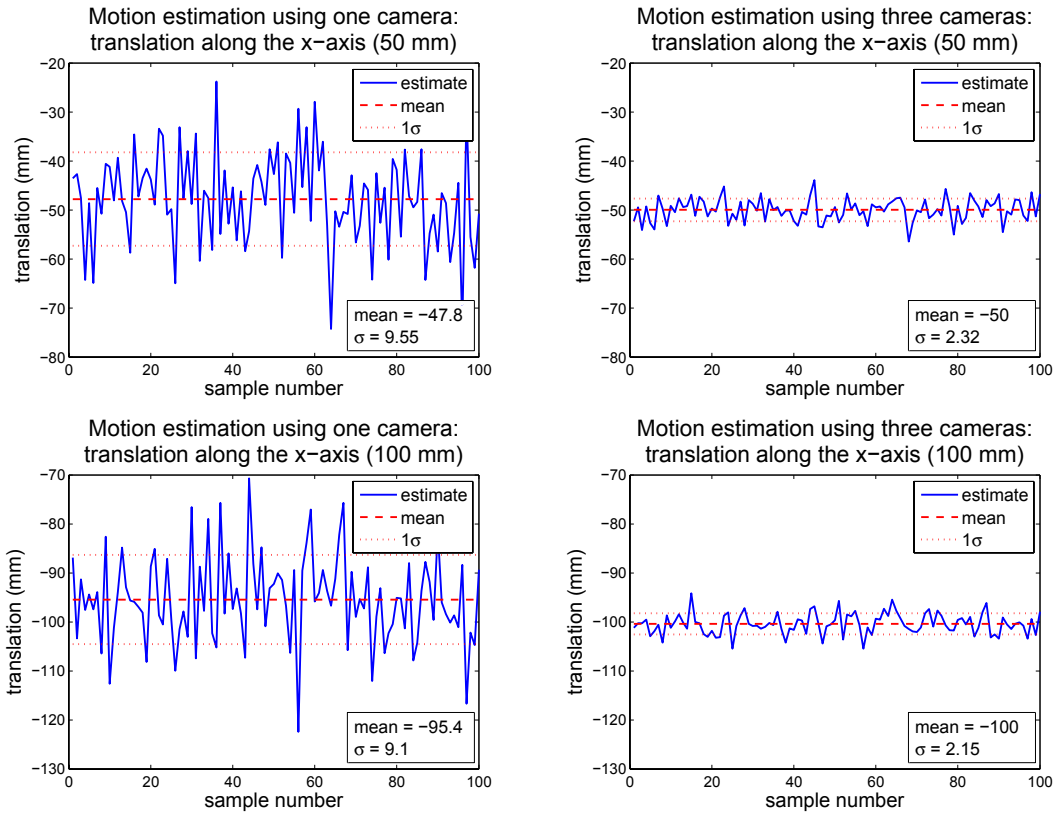


**Figure 3.13: Translation estimation results of the large movements along the x-axis (50 mm and 100 mm) using one camera (left) and three cameras (right).**

rotation perpendicular to the optical axis is introduced the single camera system shows a large estimation error compared to the multiple camera system. This effect can be observed in Figure 3.15. The poor performance is expected since the single camera system suffers the rotational ambiguity.

From the simulation results it can be seen that the multi-camera system gives better motion estimation results over the single camera system in both translation and rotation cases. More information about the motion estimation using the multi-camera system is given in Chapter 4.

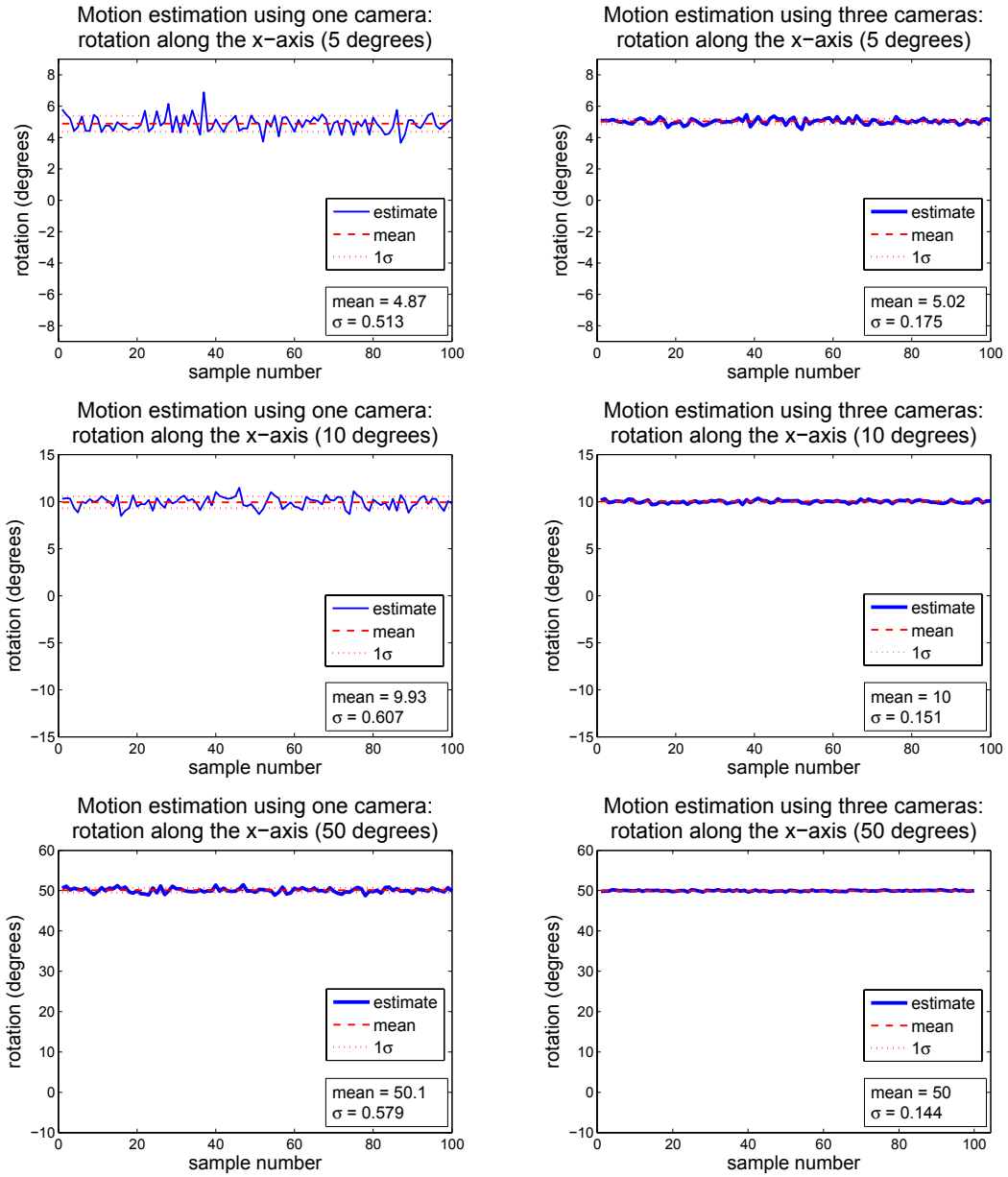


**Figure 3.14: Rotation estimation results of the rotation around the camera optical axis (5 degrees, 10 degrees and 50 degrees) using one camera (left) and three cameras (right).**

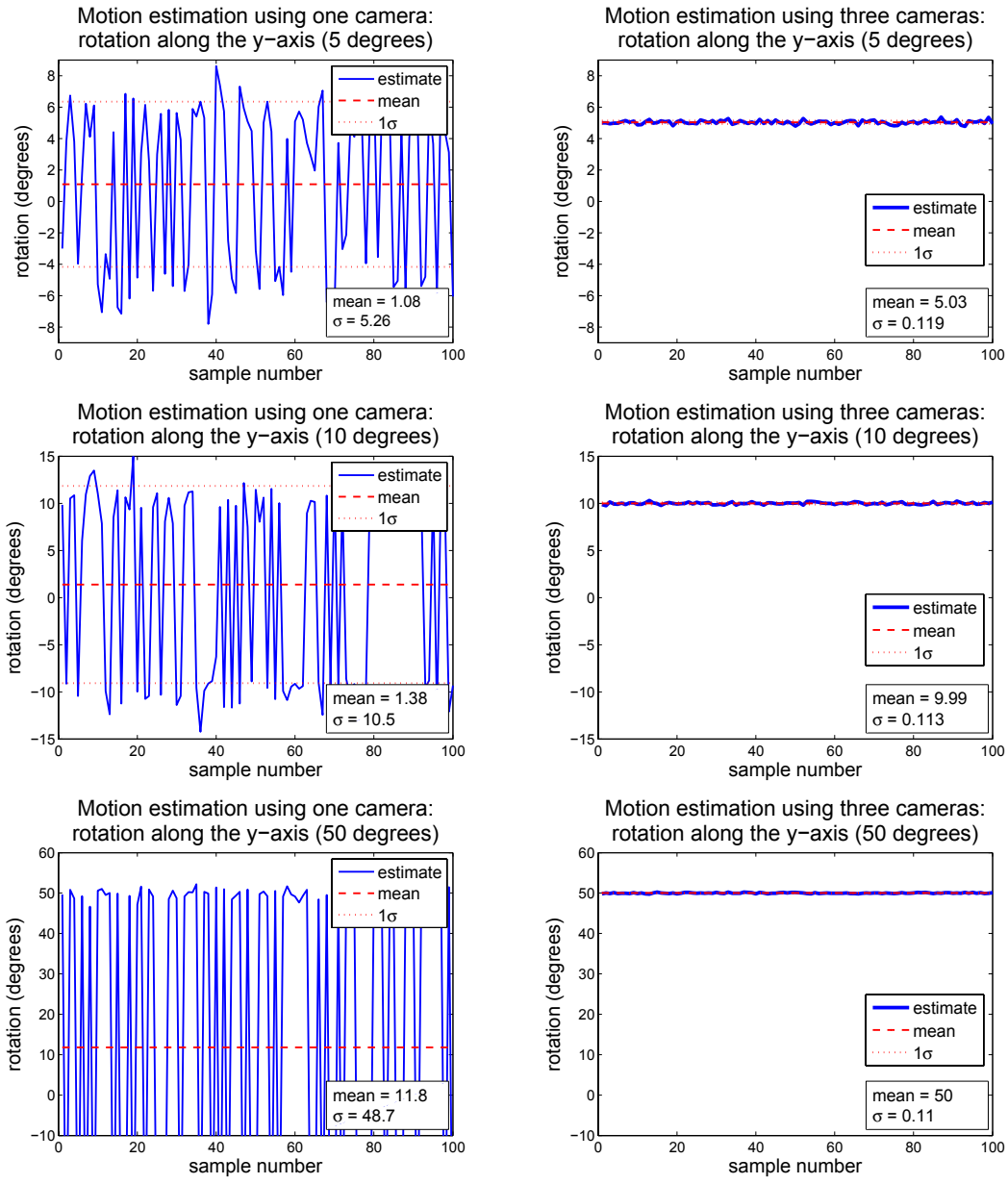


**Figure 3.15: Rotation estimation results of the rotation around the axis perpendicular to the optical axis (5 degrees, 10 degrees and 50 degrees) using one camera (left) and three cameras (right).**

## 3.7 Multi-Camera Hardware Calibration

The MCU hardware must be calibrated before it can be used in any tasks. The MCU calibration consists of two parts. The first part is the calibration of the internal parameters of the stereo camera and the second part is the calibration of the external parameters for the stereo cameras mounted on the MCU hardware platform. The calibration for the internal parameters is done separately on each stereo camera and the goal is to determine the rectification parameters for the left and the right camera so that the epipolar line is correctly aligned between their image planes. The calibration for the external parameters is done in order to determine the relative location between the three cameras so that the data acquired from different cameras can be merged in the same coordinate system. This section explains both calibration process in detail.

### 3.7.1 Calibration of the Internal Parameters

The internal parameters which are needed for the rectification of the left and right images of the stereo camera can be obtained through the calibration process similar to the one described in Tsai (1986). The calibration is done by using a chessboard pattern where the array of feature points are extracted and used to determine the lens distortion and relevant parameters. In this work, the actual calibration is done using the calibration routine included in the SVS library. More details about the calibration process using the SVS library can be found in Konolige & Beymer (2007).

### 3.7.2 Calibration of the External Parameters

The main task of the MCU is to acquire 3D images from the scene using three cameras. In order to do this, 3D images from all cameras have to be transformed into the same coordinate frame and thus the 3D transformation information between the cameras are needed. Figure 3.16 shows the geometric relationship between all three cameras.

To simplify the calibration process, two cameras are to be calibrated at a time. In other words, one camera is chosen as a reference camera and the 3D transformation between the reference camera and the other camera is determined using a calibration target. By using this approach, two calibrations are needed to find all the transformation information between the reference camera and the other two cameras.

From Figure 3.16, let z-axis camera be the reference camera so that the reference axes of the MCU coincide with those of the z-axis camera defined by $Z_x$, $Z_y$ and $Z_z$. The goal of the calibration process is to determine the transformations ${}^{x}M_{z}$ and

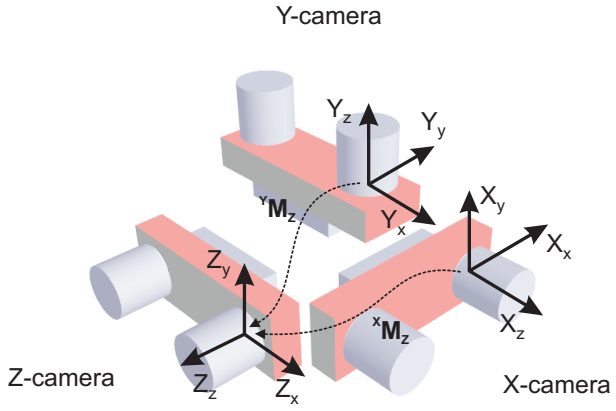


**Figure 3.16: Relation between three cameras on the MCU.**

${}^{y}M_{z}$, which refer to the transformation of the x-axis camera's coordinate to the z-axis camera's coordinate and the transformation of the y-axis camera's coordinate to the z-axis camera's coordinate respectively. The calibration target used is shown in Figure 3.17. The target consists of several square markers (1.5 × 1.5 cm) placed on a flat surface which form the shape of two cross patches located 140 cm apart from each other. Since the cameras are pointing 90 degrees away from each other they do not share any common views and therefore the large distance of 140 cm between the two cross patches is used to make sure that both patches can be seen by the cameras at the same time. The position of the cameras are then determined related to the calibration target and finally the relative position and orientation between the two cameras can be determined. Figure 3.18 shows the geometric description of this calibration scheme. The location and pose of camera 1 is related to the calibration patch A by a rigid transformation $M_A$ and location and pose of camera 2 is related to the calibration patch B by a rigid transformation $M_B$. Once $M_A$ and $M_B$ are available the resultant transformation between camera 1 and camera 2 (${}^{2}M_{1}$) can be determined.

Figure 3.19 shows the MCU hardware and the calibration target during the calibration process. In this case the y-axis camera is to be related to the z-axis camera. The markers from each cross patch are detected by both stereo cameras. Figure 3.20 shows the 3D plot of the markers' location in the same coordinate frame before and after using the correct transformation information respectively.

Once the calibration process is done, the transformation information can be used to merge the 3D images acquired from all three stereo cameras into the same reference coordinate for use in further processes.

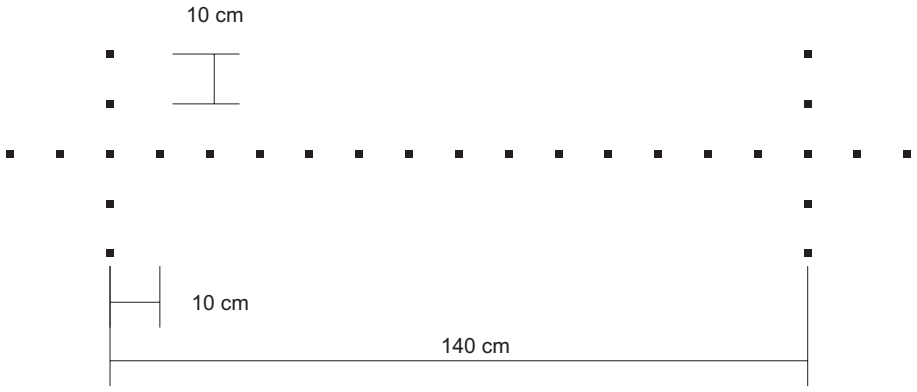


**Figure 3.17: The calibration target specially designed for the MCU.**

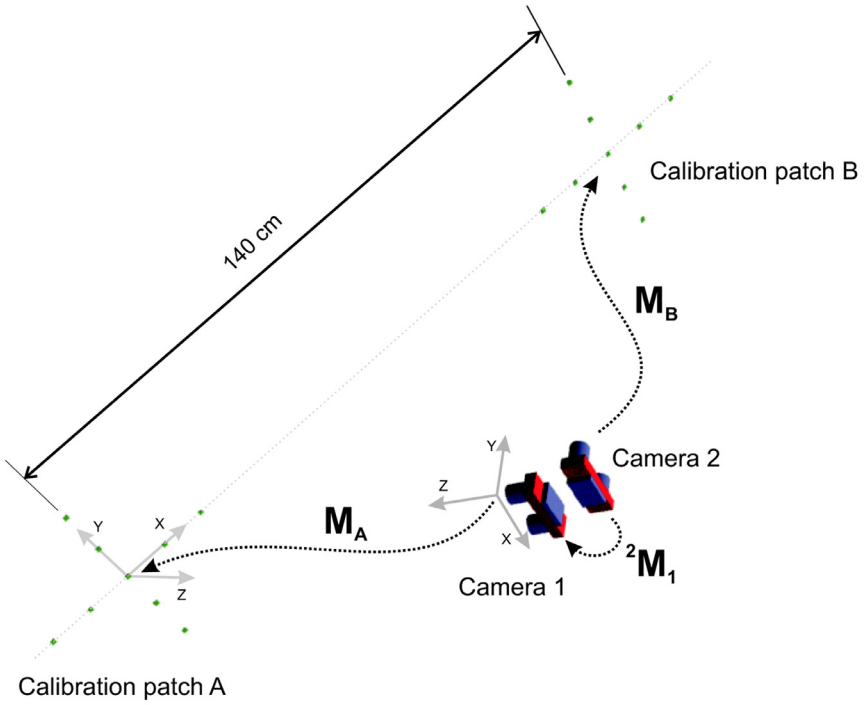


**Figure 3.18: Calibration for the transformation parameters between two cameras.**

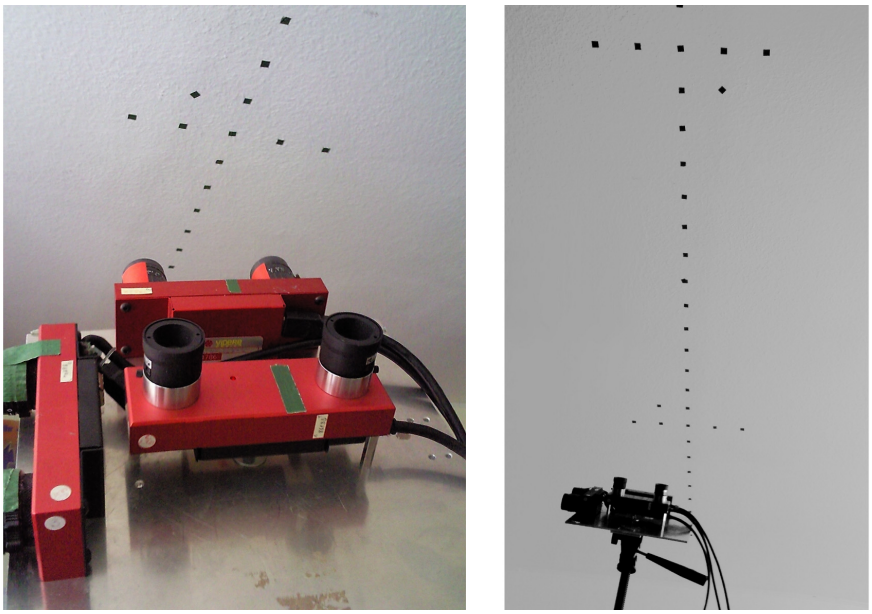

**Figure 3.19: Calibration of the Y- and Z-camera using a calibration target.**

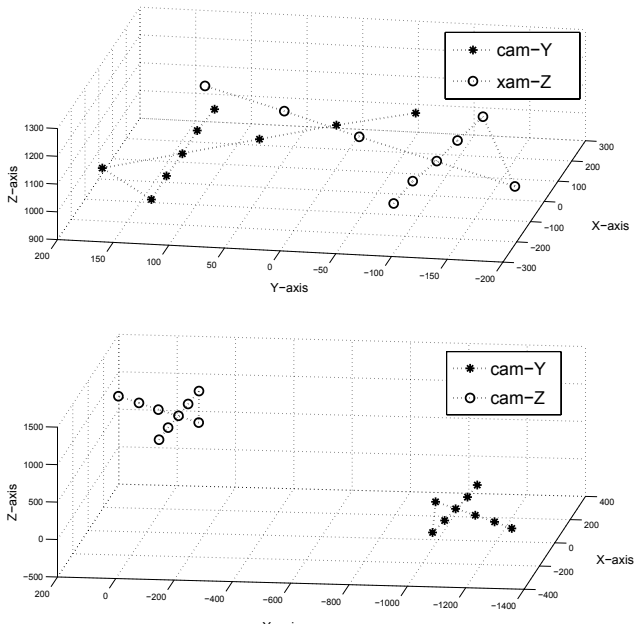


**Figure 3.20: The 3D plots of the cross patches as seen by the y-axis and z-axis camera (a) before and (b) after applying the correct transformation information obtained during the calibration process.**

## 3.8 Conclusion

This chapter explained the state of the art of the MCU system and the detailed description of the hardware and software components of the MCU system. The main advantage of the multi-camera arrangement used in the MCU system is the elimination of the motion ambiguities which results in an increased 3D motion estimation accuracy compared to the single camera system. Simulation results confirming the improved accuracy are also presented. A calibration method for the MCU hardware system using a specially designed calibration procedure is also explained.

# Chapter 4

# Real-time 3D Motion Estimation using MCU

***Introduction — Feature Detection — Feature Matching — 3D Motion Detection using 2D features — Six Degree of Freedom Motion Estimation — Conclusion***

## 4.1 Introduction

Motion estimation is a process that determines the 3D rigid motion information of the observer related to a certain reference frame, e.g. the environment, static objects, etc. The 3D motion can be broken down into two components – rotation and translation. By analyzing sequences of images, the motion between the image frames can be recovered.

This chapter explains the entire 3D motion estimation process realized by the MCU. It consists of five major steps as listed below:

1. Feature detection
2. Feature matching

3. Outlier detection
4. Motion detection using 2D feature points
5. Six degree of freedom motion estimation using 3D feature points

The relationship between these processes is illustrated in Figure 4.1.

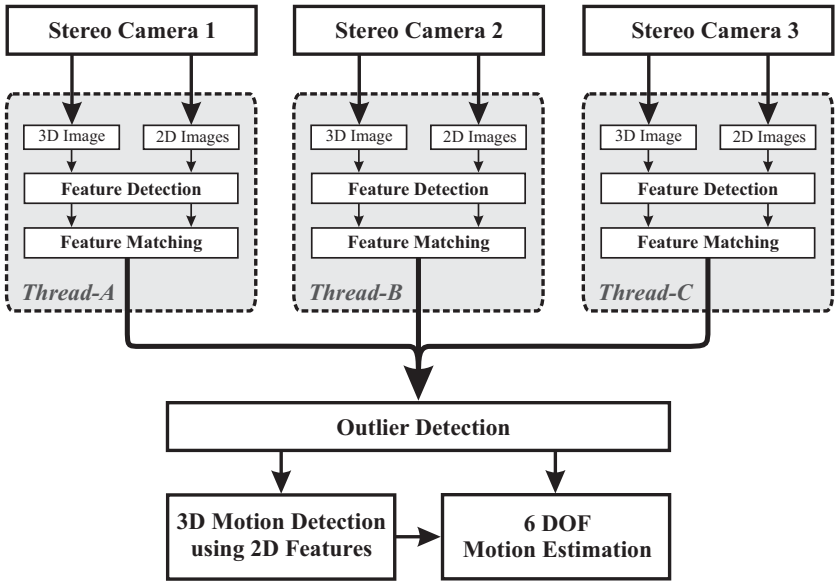


**Figure 4.1: An overview of the 3D motion estimation process.**

## A Note about Multi-Threaded Approach

In this work, the *multi-threaded* technique is used in order to utilize the computing power of the multi-core CPU available on the host computer. The multi-threaded approach makes use of the multi-core CPU by distributing the calculation of the time consuming functions among the processor cores. This speeds up the whole process train since the processing of the image frame from each camera is a very time-consuming task. The processes included in each thread are images acquisition, feature detection and feature matching processes. Each thread takes care of the information from one camera and therefore three threads are needed for the MCU system, as shown in Figure 4.1.

The following sections describe the entire 3D motion estimation process using MCU in detail.

## 4.2 Feature Detection

Feature detection is a process that detects unique or specific features that are embedded within an image such as point, line, color blob, etc. Feature detection has a vital role since it is frequently used to generate initial input that serves through the pipeline of the entire process train. In this work, the corner feature is chosen as the feature point for the MCU system. The input of the feature detection process is the 2D image and the correspondent 3D information or the disparity value. The output is a set of corner features with valid x-, y- and z-location. The main components of the feature detection include the pre-processing of the input images and the corner detection process itself. Figure 4.2 shows the entire feature detection process realized for the MCU system.

### 4.2.1 Pre-processing of the Input Image

The goal of the pre-processing of the input image is to ignore the pixels that are not suitable as feature points so that the number of pixels to be checked for cornerness can be reduced. Moreover, not every detected feature point contains the required information which is needed for the 3D motion estimation and map building task and thus they should also be ignored. Two detection methods are used to pre-process the input image in order to find these unwanted pixels or outliers. The first method quickly checks for the cornerness property of the pixel location using an intensity similarity check and the second method checks for the valid disparity value, since the disparity value is needed for the calculation of the x-, y- and z- location of the feature point. These two methods are explained below.

#### Intensity Similarity Check

This process checks for the basic corner property of the target pixel by comparing its pixel intensity against the neighbouring pixels. A corner is usually found at a location where the pixel intensity changes abruptly compared to the neighbouring pixels. Therefore there is a low chance that the target pixel represents a corner location if the intensity of the target pixel is similar to that of the neighbouring pixels. The neighbouring pixels to be checked for local intensity are shown in Figure 4.3. The pixels under the $7 \times 7$ pixels cross-shape are checked for the intensity similarity. If their intensity

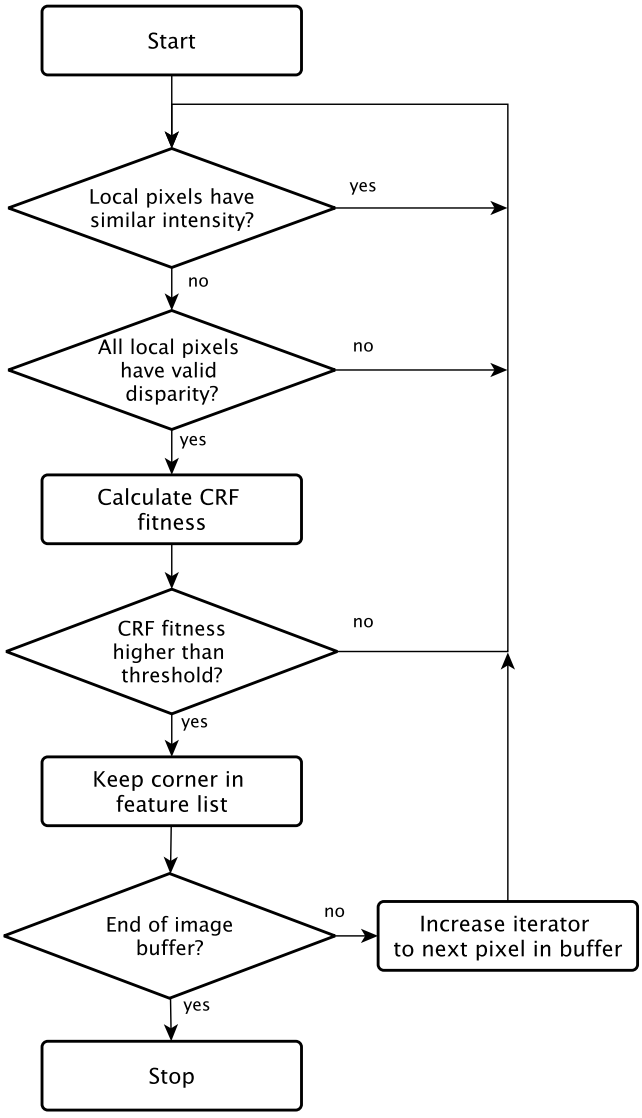


**Figure 4.2: The feature detection process.**

difference is below a threshold value then the current pixel location is ignored from the feature detection process.

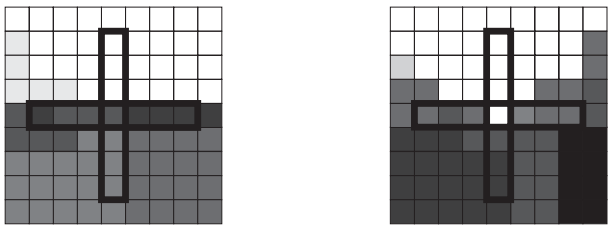

**Figure 4.3: Example of the local intensity check. On the left hand side the pixels under the seven-pixel horizontal line have a similar intensity so there is a high likelihood of a low CRF response. On the right hand side the local pixels along both lines are different (the center pixel and the other three pixels above) and therefore there is a high likelihood that the pixel location represents a corner location and thus would not be rejected.**

### Disparity Value Check

Not every pixel that passes the intensity similarity check has a valid disparity value. The disparity value is computed by the SVS library and it indicates the availability of the depth information of the pixel. Without this value the x-, y- and z-location of the pixel cannot be determined. Therefore it is logical to ignore the pixel that has no valid disparity value. Figure 4.4 shows an example of a disparity image obtained from the SVS library. The black area in the disparity image indicates no disparity value and the grey area indicates that the disparity values are available. It can be seen that a large portion of the image does not contain valid disparity values and therefore it is not necessary to apply the feature detection process to this area.

By applying the intensity similarity check and the disparity value check methods, the number of pixels to be checked for cornering is reduced and the overall speed of the feature detection process is shortened by a factor of five to ten, depending on the complexity of the environment.

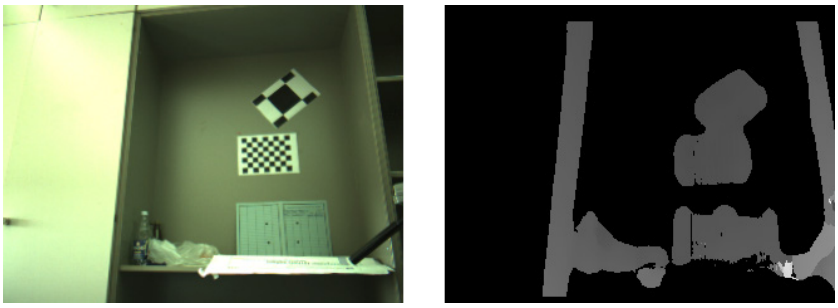

**Figure 4.4: A test scene (left) and its disparity image (right).**

### 4.2.2 Corner Detection using a Corner Response Function

The corner-type feature is a well studied feature within the machine vision community. The term corner does not directly refer to the true physical corners. Instead, corner in this case refers to the interest point or a certain place within an image where the change of intensity in certain directions occur. An example of the corner detector output is presented in Figure 4.5.

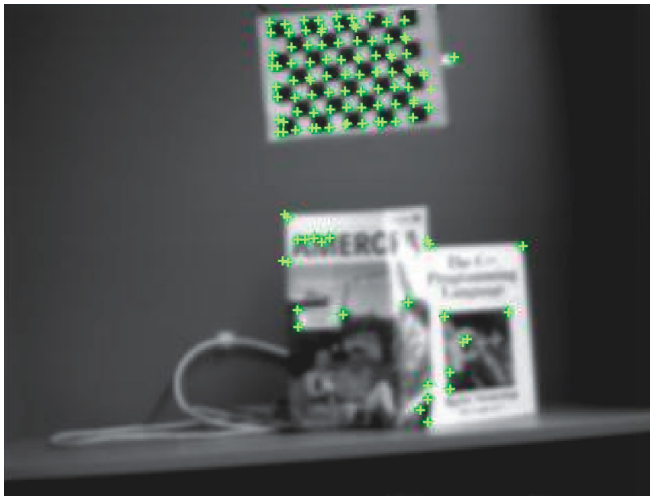

**Figure 4.5: An example output of the corner detector. The green crosses indicate the location of the corners found within the image.**

In this work, corners are used as reference points for the motion estimation process. The main reasons are:

- Corner is a feature that can be found from within almost any environment (indoors, outdoors).
- Corner is a strong, accurate and consistent feature since the exact 2D location of an individual corner can be obtained up to sub-pixel accuracy.
- Corner detection is fast and robust. Using corners instead of every pixel within the image reduces the magnitude of data needed to process which is a big benefit for the real-time system.

The current implementation of the MCU system uses the Corner Response Function (CRF) [Trajković & Hedley (1998)] as a corner detector due to its speed, simplicity and robustness. The CRF algorithm does not consider only the intensity gradient within the horizontal and vertical direction but also takes into account the intensity gradient in other directions according to the masks being used. This makes it robust against the detection of the same feature point against a slight rotation of the camera.

Consider Figure 4.6. Let $C$ be the location of the pixel to be tested for corner response. The corner response function is defined by:

$$R_{CRF} = min((i_P - i_C)^2 + (i_{P'} - i_C)^2) \tag{4.1}$$

where

| | |
|---|---|
| $i_P$ | image intensity at point $P$ |
| $i_{P'}$ | image intensity at point $P'$ |
| $i_C$ | image intensity at center point $C$ |

The point pair $P$ and $P'$ is replaced by other point pairs that are opposite to each other, e.g. $A$ and $A'$ or $B$ and $B'$. These point pairs are defined according to the mask definition. The pixel at location $C$ is considered a corner if the smallest response derived from any of the point pairs is larger than a certain threshold. In other words, a small CRF response indicates the small intensity gradient change and a large CRF response indicates a large intensity gradient change, which is a typical property of a corner. This effect is illustrated in Figure 4.7.

Another advantage of the CRF algorithm is the use of multiple masks. A mask size of $3 \times 3$ pixels would yield a total number of four pixel pairs while a mask size of $7 \times 7$ would yield a number of eight pixel pairs. By having more pixel pairs the cornerness

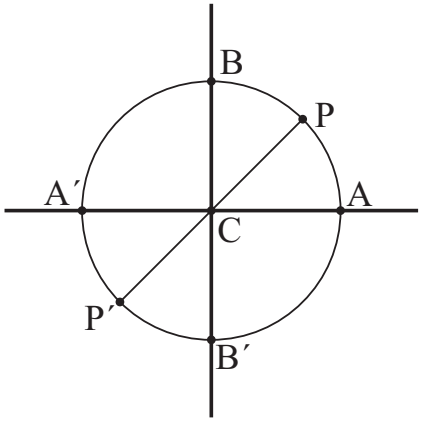


**Figure 4.6: Illustration of the neighbouring pixels required for the CRF calculation.**

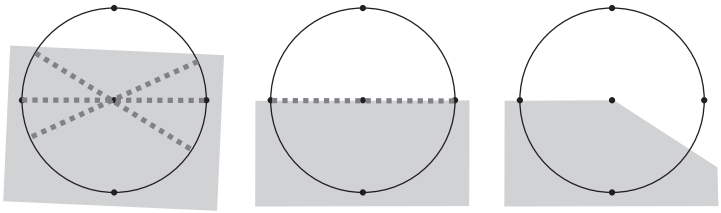

**Figure 4.7: Some different scenarios for the corner detection. On the left most figure there are many pixel pairs, indicated by dotted lines, that would generate small CRF responses since the intensity differences at both ends of such lines are small. In the middle figure there is one pixel pair that produces a small CRF response. In the rightmost figure there are no pixel pairs that would produce a small CRF response and thus a corner is found.**

is being checked using finer angle steps. The use of larger masks also confirms the cornerness in the larger area, which makes it more robust against image noise. Note that all masks can be deployed iteratively at the same location in order to securely confirm the cornerness property of the tested pixel. The CRF masks with different sizes are shown in Figure 4.8.

### Improvement for the Corner Detection Process

The 2D images from the MCU usually contain noise and it is helpful to apply a blurring filter to the image before the detection of the corners. A Gaussian convolution mask is used to smoothen the image. The $3 \times 3$ convolution mask is defined by:

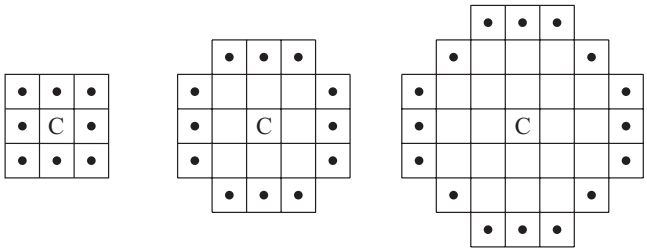


**Figure 4.8: CRF masks with different sizes ($3 \times 3$, $5 \times 5$ and $7 \times 7$ pixel).**

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{4.2}$$

By smoothening the image, the noisy pixels are no longer identified as corners and only *strong* feature points are taken into account in the next process. Figure 4.9 shows the difference of the image before and after applying a Gaussian convolution mask. The left image shows a sample scene with sharp edges from black objects where rough edges can be seen. These rough edges are considered as noise since the real physical shape of the object does not contain such spiky shapes.

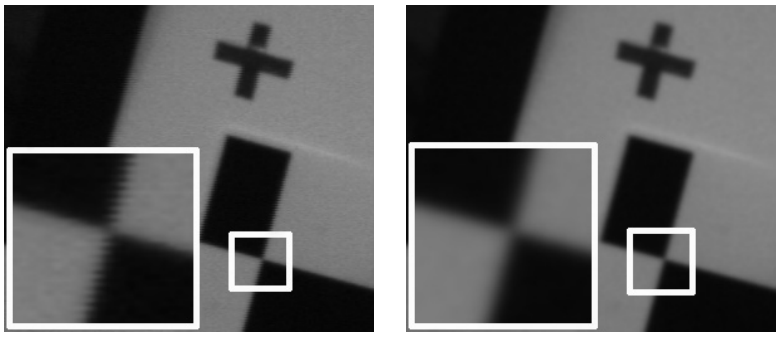

**Figure 4.9: Comparison between images with and without the Gaussian blur.**

Figure 4.10 shows the corner detection result with and without the blurring filter. The left image produces far too many corners due to the sharp and dusty noise of the image while the right image shows only significant features. Note that the Gaussian

mask is not applied on the image buffer itself but instead on a separate buffer in order to keep the original intensity value of the pixel for later use, e.g. for back-projecting on the 3D model or for viewing in the real-time user interface.

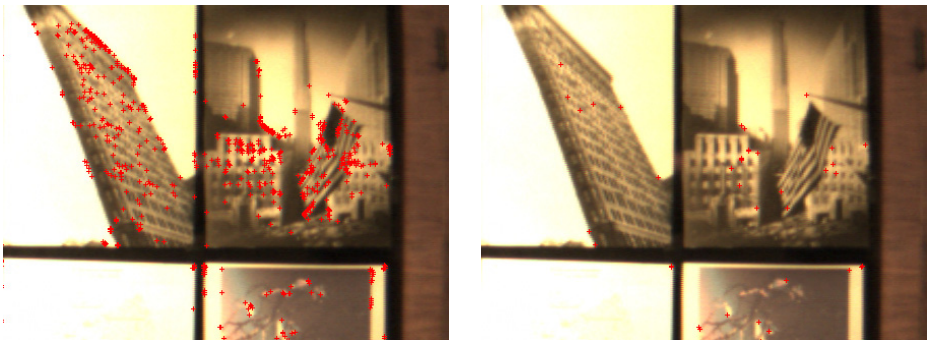

**Figure 4.10: The corner detection results with (right) and without (left) the Gaussian blur.**

## 4.3 Feature Matching

Feature Matching is the process that matches the same location on the physical object, e.g. feature point, that is being seen on two successive image frames. This problem is also known as the correspondence problem. The goal is to match a small patch of the previously captured image (at time $t-1$) to a certain area of similar size in the current image (at time $t$). The matching is successful when a patch with similar intensity and pattern can be found in the target image. The input of the feature matching are two lists of corner points found in the current image and the previous image. These corner points are derived by the corner detection routine described earlier. The output is a subset of the feature points which appear on both images and that correspond to each other. Figure 4.11 shows an example of the same feature points being matched after a camera movement.

The entire feature matching process realized for the MCU system includes:

- 2D displacement boundary checking
- Feature matching using NCC coefficient

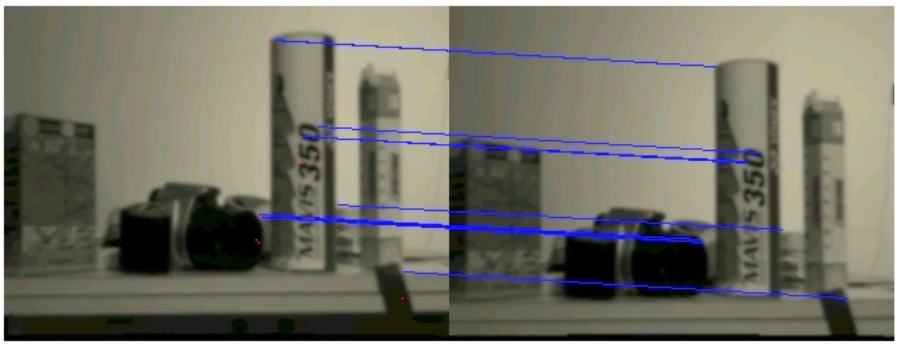

**Figure 4.11: An example of the feature matching after a camera movement. On the left hand side is the image acquired at time $t-1$ and on the right hand side is the image acquired at time $t$.**

The interaction between these processes is shown in Figure 4.12. More detailed explanations for these processes are given in this section.

### 4.3.1 Checking the Feature Point Displacement

The simplest way to match the feature set after a camera movement is to compare a selected feature in the first image against every feature available in the second image. While this method can give a correctly matching result, the time required to check every point pairs is large. By taking into account the assumption of the system behavior, i.e. fast acquisition rate and slow movement of the MCU, the matching process can be narrowed down to a certain area on the target image, i.e. the change of feature point location should be small since the MCU movement within a time step is assumed to be small.

In the current implementation, a 2D displacement threshold is used to limit the search boundary. Consider Figure 4.13; a feature in image frame 1, marked by a white cross, is to be matched against the feature set on image frame 2 after a camera movement. By defining a search area on image frame 2 within a certain radius the number of feature points to be tested for matching is lessen and therefore the time required to find the corresponding match is reduced.

The condition whether the feature points in image frame 2 should be checked or not is defined by:

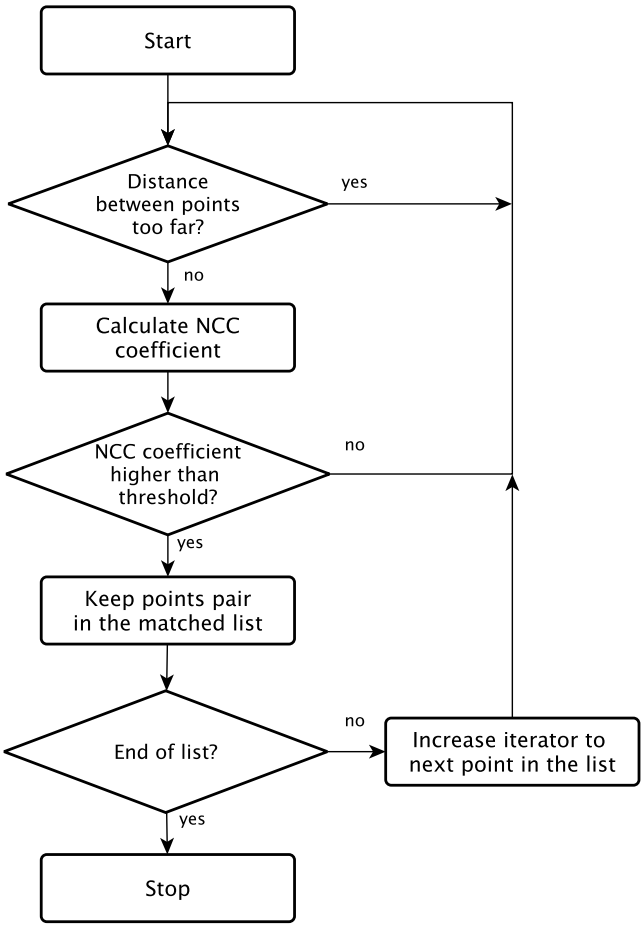


**Figure 4.12: The feature matching process.**

$$\text{Feature point displacement checking} \begin{cases} \text{returns true} & \text{if displacement} < r \\ \text{returns false} & \text{if displacement} > r \end{cases}$$

The radius $r$ is defined by:

$$r = \sqrt{(x - x')^2 + (y - y')^2} \tag{4.3}$$

Where $(x, y)$ and $(x', y')$ are the 2D coordinates of the feature point in image frame 1 and image frame 2 respectively. The calculation is done in pixel units.

The set of narrowed down feature points is then tested for the corresponding match using the normalized cross correlation as described in the following section.

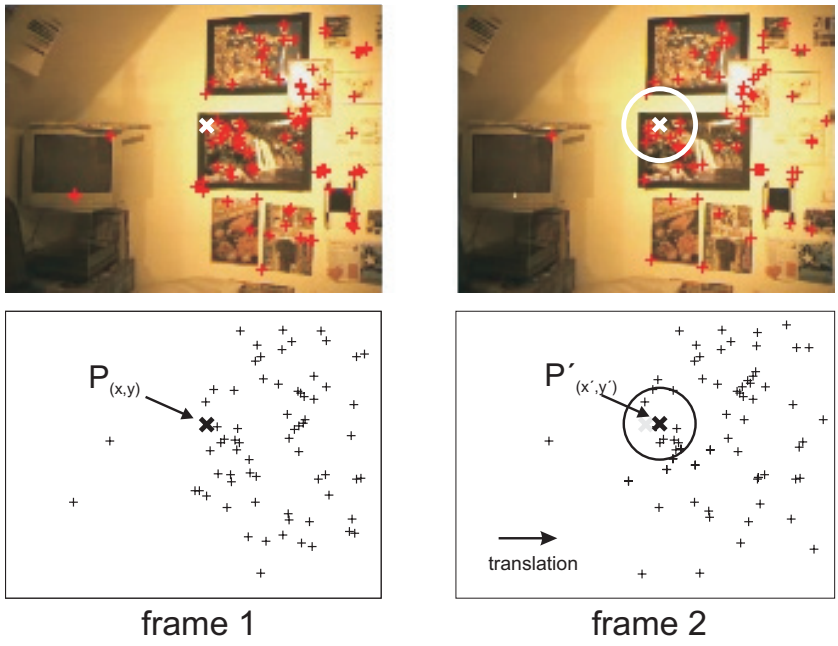


**Figure 4.13: The displacement boundary checking of the 2D feature points. In frame 1, a feature (marked by a white cross) is to be matched to the candidate features on frame 2. By assigning a boundary (marked by a circle with defined radius around the target feature), the search for the matched feature is narrowed down to only the features within the circle radius $r$.**

### 4.3.2 Feature Matching using Normalized Cross Correlation

In this work, the Normalized Cross Correlation (NCC) technique described in Lewis (1995), Lewis (2005) and Belle (2007) is used for the feature matching process. NCC is an intensity based, cross-correlation that normalized the input images according to the pixel information of the image patch. The normalization helps to cope with the changing light conditions and thus the same feature can be detected again after several time steps or after a change of camera location. Figure 4.11 shows a result of the feature matching on two successive images from the left camera of the stereo camera.

For the feature point $P_{(x,y)}$ and $P'_{(x',y')}$, the normalized cross correlation coefficient is defined by:

$$\frac{\sum_{x,y}[P_{(x,y)} - \bar{P}][P'_{(x',y')} - \bar{P}']}{\left\{\sum_{x,y}[P_{(x,y)} - \bar{P}]^2 \sum_{x,y}[P'_{(x',y')} - \bar{P}']^2\right\}^{0.5}} \tag{4.4}$$

where

$\bar{P}$ mean of the intensity in the region under point $P$
$\bar{P}'$ mean of the intensity in the region under point $P'$

The current NCC implementation has an effective comparison area of $9 \times 9$ pixels. The empirical value for the NCC coefficient (NCC=0.9) is found during the development of the MCU system and is used for the selection of the matched feature, that is:

$$\text{feature points matching} \begin{cases} \text{returns true} & \text{if NCC coefficient} \geq 0.9 \\ \text{returns false} & \text{if NCC coefficient} < 0.9 \end{cases}$$

### 4.3.3 Outlier Detection

Although the feature points are successfully matched using the NCC coefficient, it is possible that some outliers are still existing. Further steps can be used to detect and reject these outliers.

#### Outlier Detection using Hue Component Matching

The color match constraint makes use of the hue component to reject the outliers. It compares the matched feature points derived from the feature matching process to determine whether the hue values are similar or not. If the hue values of the matched feature point are not similar then they are considered outliers. The color information of the feature point can be retrieved from the 2D color image from the stereo camera. The stereo camera gives out the color information in RGB format. However the use of an RGB color model is not suitable for the matching scheme since it is easily manipulated by the brightness of the scene. In order to get around this problem, the HSV color model is used. The HSV color model is better for color matching since it separates the color information into a unique channel and therefore the outlier rejection can be done robustly against the change of light conditions in the scene.

The hue value $h$ is usually represented in degrees and it runs from 0° to 360°. Given the RGB value from the stereo camera, the hue value is calculated by:

$$h = \begin{cases} 60 \times (4 + \frac{(r-g)}{max-min}) & \text{if max} = \text{b} \\ 60 \times (2 + \frac{(b-r)}{max-min}) & \text{if max} = \text{g} \\ 60 \times (\frac{(g-b)}{max-min}) & \text{if max} = \text{r} \end{cases}$$

where $max = max(r, g, b)$ and $min = min(r, g, b)$. The $r$, $g$ and $b$ values are the red, green and blue components of the RGB model, which range from zero to one.

Figure 4.14 shows a common color representation of the hue value. For example, red is located at 0° and 360°, blue is located at 240° and green is located at 120°.

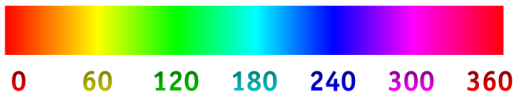


**Figure 4.14: A color scale according to the hue component.**

### Outlier Detection using the Detected MCU Motion

This process rejects the feature points whose motion vectors are not in agreement with the detected MCU motion. More explanation about this outlier detection is given later in Section 4.4.

## 4.4 3D Motion Detection using 2D Features

By using just the 2D images from the MCU, a very accurate 3D motion of the MCU such as forward/backward movement as well as rotations can be detected. As described in Section 3.5, the unique design of the MCU makes it possible to correctly detect even a small rotation and translation with low motion ambiguity error. The 3D motion detection is done based on the calculation of the 2D features' location and thus it is simple and fast, yet accurate and robust. This motion information is used to reject the feature points that are not in agreement with the current movement of the MCU and it can also be used to initiate the final 3D motion estimation using the ICP algorithm later on.

Consider Figure 4.15. The feature points $P$ and $P'$ represent the matched feature point derived from the feature matching process. The motion detection starts by calculating the 2D displacement of the feature point between two successive frames. The 2D displacements are calculated separately in x- and y-axes by the following equations:

$$x_{displacement} = x + u - x = u \tag{4.5}$$

$$y_{displacement} = y + v - y = v \tag{4.6}$$

where

$x$ previous location of the feature point in x-axis (in pixels)
$y$ previous location of the feature point in y-axis (in pixels)
$u$ displacement of the feature point in x-axis (in pixels)
$v$ displacement of the feature point in y-axis (in pixels)

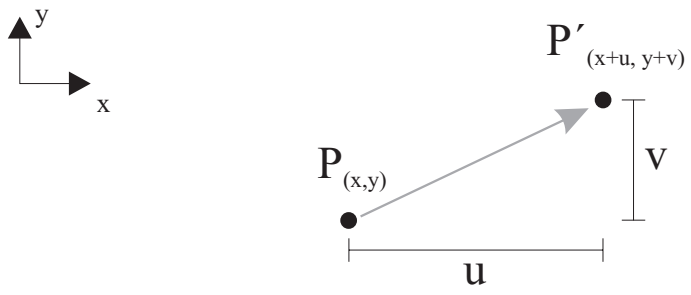


**Figure 4.15: 2D displacement of a feature point after a movement of the camera.**

The displacement information of the feature point is then used to determine the 2D motion of the camera in terms of translation in the x- and y-axes. For example, in Figure 4.15 the feature point $P$ is displaced to the new location $P'$. This displacement is obtained by the movement of the camera toward the left direction along the x-axis and down direction along the y-axis. That is, while the new location of the feature point appears to be at the north-east of the previous location, the camera is in fact moved in the south-west direction. By calculating the 2D displacement of all the feature points available within the camera image, the 2D movement of the camera related to the rigid scene is obtained as follows:

$$\text{2D motion in x-axis} = \begin{cases} \text{right (positive direction)} & \text{if u} < 0 \\ \text{left (negative direction)} & \text{if u} > 0 \end{cases}$$

$$\text{2D motion in y-axis} = \begin{cases} \text{up (positive direction)} & \text{if v} < 0 \\ \text{down (negative direction)} & \text{if v} > 0 \end{cases}$$

These motion directions are illustrated in Figure 4.16.

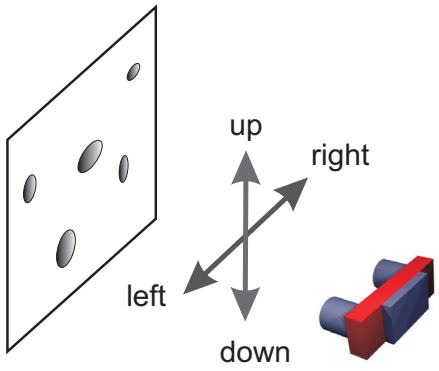


**Figure 4.16: Directions of the 2D motion of a stereo camera.**

In the case of single camera, this information is prone to motion ambiguity. However, in multi-camera systems the information from all three cameras is combined and the ambiguity error is compensated[1]. The combination of the 2D motion information from all cameras is illustrated in Figure 4.17, where the movement directions of the MCU are referred to the coordinate frame of the x-camera. For example, if the x-axis camera detects an up-movement and the z-axis camera also detects an up-movement while the y-axis camera reports no movement then it can be concluded that the MCU is undergoing an upwards movement. Although simple, this method has proved to be very efficient and is good enough to be used for the outlier detection as well as to generate initial values for the 3D motion estimation algorithm. Table 4.1 lists some of the possible combinations and the estimated motions.

[1]See Section 3.5 for more details about the elimination of motion ambiguity using multi-camera system.

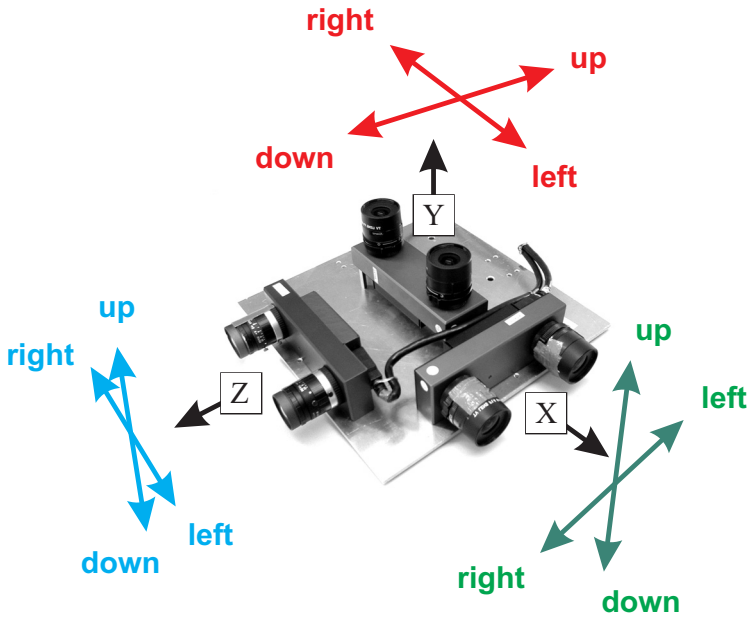


**Figure 4.17: 3D motion detection of the MCU using 2D features. Here the 2D motions from all three cameras are combined to produce the 3D motion of the MCU. This motion is then used in the later processes, i.e. for the outlier detection and final 3D motion estimation.**

## Outlier Detection using the Detected MCU Motion

Now that the 3D motion of the MCU is detected, further outliers among the detected feature points can be screened out. This is done by looking at the feature points that have different motion vectors compared to the detected 3D motion of the MCU. For example, if the detected MCU motion is the combination of up and right directions, there might be some feature points that have different motion vectors which are not in agreement. Such feature points are considered as outliers. Figure 4.18 explains this scenario in brief.

**Table 4.1: 3D motion detection using 2D motion information.**

| X-cam | Z-cam | Y-cam | motion |
|---|---|---|---|
| up | up | - | up |
| down | down | - | down |
| left | - | up | left |
| right | - | down | right |
| - | left | left | forward |
| - | right | right | backward |
| - | down | down | rotate cw around x-axis |
| - | up | up | rotate ccw around x-axis |
| up | - | right | rotate cw around y-axis |
| down | - | left | rotate ccw around y-axis |
| left | left | - | rotate cw around z-axis |
| right | right | - | rotate ccw around z-axis |
| up, right | up, right | - | up and rotate ccw around z-axis |

*note: cw = clockwise, ccw = counterclockwise*

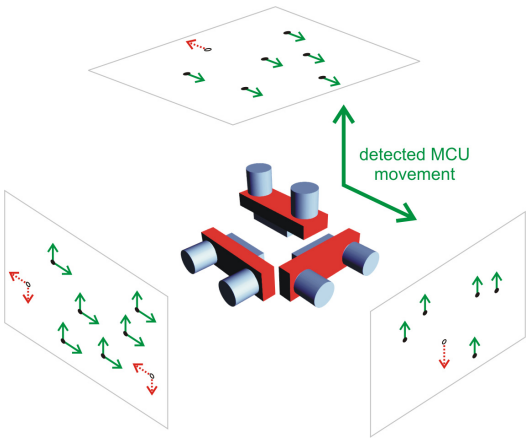


**Figure 4.18: The detected 3D motion can be used to eliminate the outliers. In this example, the 2D features with red motion vectors are considered outliers since they have different motion vectors compared to the detected MCU motion.**

## 4.5 Six Degree of Freedom Motion Estimation

Now that the set of matched feature points without outliers is available, the six degree of freedom (DoF) motion can be estimated. The six DoF motion is expressed in terms of three rotations and three translations along the x-, y- and z-axis. To begin with, the initial estimation of the motion parameters are determined using the RANSAC algorithm and afterward the final 3D motion is estimated using the ICP algorithm. These two processes are explained in detail in the following sections.

### 4.5.1 Initialization of Motion Parameters using the RANSAC Algorithm

The Random Sample Consensus (RANSAC) algorithm is used for the initialization of the motion parameters. The RANSAC algorithm is a tool to estimate the parameters of a system model as well as for generating a set of inliers apart from the noisy data set. More detail about RANSAC can be found in the work of Fischler & Bolles (1981).

The input of the RANSAC algorithm is the set of matched feature points derived from the previous process and the output are the consensus set of feature points and an estimation of the 3D motion. Figure 4.19 shows the author's implementation of the RANSAC algorithm used in this work.

Once the RANSAC algorithm is executed, the motion initial parameters including the rotation and translation components as well as the average 3D distance error between the two set of matched feature points are obtained. This information is then used for the final 3D motion estimation using the ICP algorithm, which is explained in the next section.

### 4.5.2 Estimation of Motion Parameters using the ICP Algorithm

In this work, the Iterative Closest Point (ICP) is used to find the rotation and translation of the MCU at each timestep. The ICP algorithm determines the optimal 3D transformation of the MCU that fits the two sets of feature points acquired before and after the camera movement by iteratively minimizing the Euclidean distance error between the feature points. More information about the ICP algorithm can be found in the work from Besl & Mckay (1992) and Rusinkiewicz & Levoy (2001).

The input of the ICP algorithm are two sets of feature points acquired before and after the camera movement and the information derived from the RANSAC algorithm

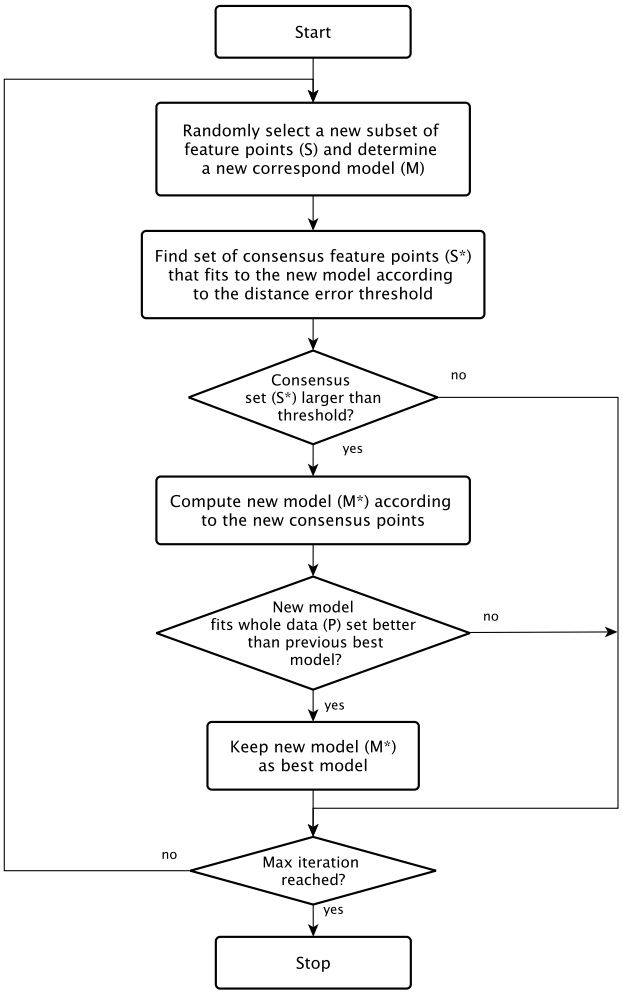


**Figure 4.19: The RANSAC algorithm.**

including initial 3D motion information and the average 3D distance error. The output of the ICP algorithm is the 3D rotation and translation that optimally describes the motion of the MCU. The ICP algorithm consists of the following steps:

- Set the error threshold $D_{max}$ using the average error distance derived from the RANSAC algorithm (first time only).
- Compute the optimal motion between the matched feature points using SVD.
- Apply motion to the feature points set.

- Update the threshold $D_{max}$ and repeat the whole process until the maximum number of iterations is reached or the 3D distance error is lower than a threshold value.

The author's implementation of the ICP algorithm is illustrated in Figure 4.20.

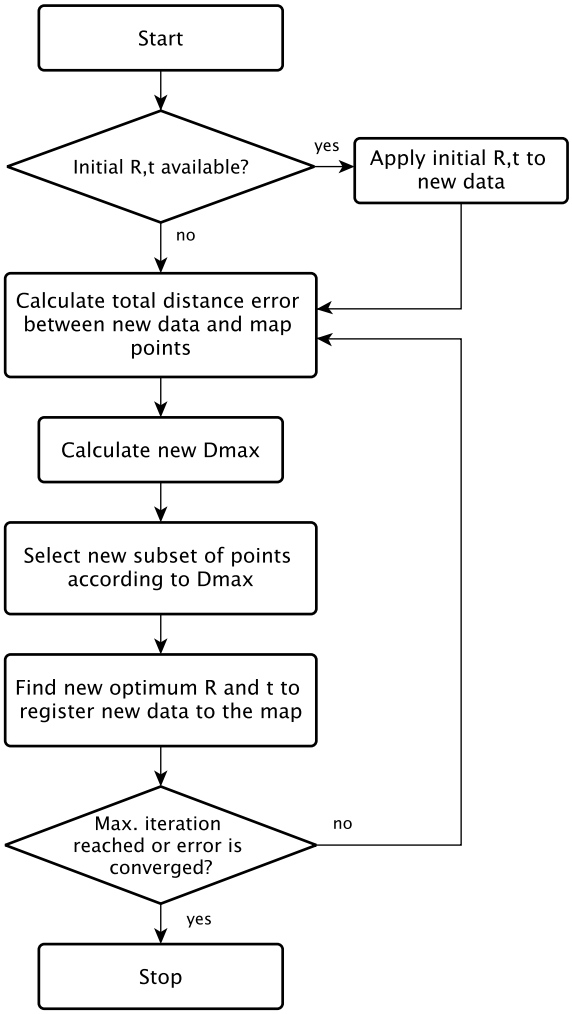


**Figure 4.20: The ICP algorithm.**

### The $D_{max}$ Value

The $D_{max}$ is the threshold value that is used for the selection of the feature points during each ICP iteration. If $D_{max}$ is bigger than necessary then the ICP algorithm may not converge because the bad point pairs are always included during the iterations. On the other hand, if the $D_{max}$ value is too small, the number of iterations will increase since the good point pairs are not being included, which results in a long converge time. This problem was addressed by Zhang [Zhang (1994)] and the solution is to recalculate the $D_{max}$ value at each iteration according to the statistical analysis.

Let $p = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$ and $q = \{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_n\}$ be the set of feature points acquired before and after the camera movement and $d_i$ be the 3D Euclidean distance error between the $i^{th}$ correspond feature points in $p$ and $q$. The mean distance $d_{mean}$ and the deviation of the distance $\sigma$ are calculated by:

$$d_{mean} = \frac{1}{N}\sum_{i=1}^{N} d_i \tag{4.7}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(d_i - d_{mean})^2} \tag{4.8}$$

Where $N$ is the number of the corresponding feature point. For each iteration, the value of $D_{max}$ is selected according to the value of $d_{mean}$ as follows:

$$D_{max} = \begin{cases} d_{mean} + 3\sigma & \text{if } d_{mean} < D_{max} \quad \text{(the registration is good)} \\ d_{mean} + 2\sigma & \text{if } d_{mean} < 3D_{max} \quad \text{(the registration is moderate)} \\ d_{mean} + \sigma & \text{if } d_{mean} < 6D_{max} \quad \text{(the registration is poor)} \end{cases}$$

The new $D_{max}$ is then used to select the corresponding feature points in $p$ and $q$ at each iteration of the ICP algorithm. If the distance between the corresponding feature points in $p$ and $q$ is larger than $D_{max}$ then the feature point is ignored and only the remaining feature points are used for the estimation of the motion information.

### Finding Optimal Rotation and Translation using SVD

The process to determine optimum rotation and translation between two feature point sets, say $p$ and $q$, using the Singular Value Decomposition (SVD) consists of two steps. First the translation component is detached by using the centroid information of both data set:

$$\mathbf{p}'_i = \mathbf{p}_i - \mathbf{p}_{centroid} \tag{4.9}$$
$$\mathbf{q}'_i = \mathbf{q}_i - \mathbf{q}_{centroid} \tag{4.10}$$

Where $i = \{1, 2, 3, ..., N\}$ and $N$ is the number of feature points in the point set. The covariance matrix $\mathbf{H}$ is computed by:

$$\mathbf{H} = \sum_{i=1}^{N} \mathbf{q}'_i \mathbf{p}'^T_i \tag{4.11}$$

The next step is to compute the SVD of $\mathbf{H}$:

$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{4.12}$$

As a result, the optimal 3D rotation $\mathbf{R}$ is defined by:

$$\mathbf{R} = \mathbf{V}\mathbf{U}^T \tag{4.13}$$

and the 3D translation vector $\mathbf{t}$ is defined by:

$$\mathbf{t} = \mathbf{p}_{centroid} - \mathbf{R}\mathbf{q}_{centroid} \tag{4.14}$$

### Euclidean Distance Error Calculation

The goal of the ICP algorithm is to find the optimal rotation and translation information that minimize the Euclidean distance between the matched features within two sets of feature points after a camera movement, i.e. $p$ and $q$:

$$\sum_{i=1}^{N} \|\mathbf{p}_i - \mathbf{q}_i(\mathbf{R}, \mathbf{t})\| \tag{4.15}$$

The Euclidean distance between $\mathbf{p}_i$ and $\mathbf{q}_i$ is defined by:

$$\|\mathbf{p}_i - \mathbf{q}_i\| = \sqrt{(x_{p_i} - x_{q_i})^2 + (y_{p_i} - y_{q_i})^2 + (z_{p_i} - z_{q_i})^2} \tag{4.16}$$

As shown in Figure 4.20, the Euclidean distance is used to determine the termination of the ICP algorithm. If the change of the Euclidean distance in the current iteration compared to the previous iteration is smaller than a certain threshold value then the algorithm can be terminated before the maximum number of iterations is reached.

## 4.6 Conclusion

This chapter explained how 3D motion estimation using MCU works. First the feature points are detected using the CRF corner detector. Then the feature points between two successive frames are related to each other by feature matching process using the NCC coefficient. Once the correct correspondences are found the 2D motion from all three cameras are determined. This 2D motion information is combined to produce the initial 3D motion of the MCU. Finally, the ICP algorithm is used to determine the final 3D motion estimation of the MCU.

# Chapter 5

# Real-time 3D Map Building using the MCU

***Introduction — Real-time 3D Map Building using a Probabilistic Approach — FastSLAM — MCU System Equations — FastSLAM Implementation — Simulation of the FastSLAM System — Real-time 3D Photorealistic Map Building using 3D Images — Conclusion***

## 5.1 Introduction

This chapter presents a method for a real-time 3D map building using the MCU system. The main goal is to maintain a consistent map of the environment as well as the location of the MCU and its travelled trajectory. In order to reduce the computational complexity and to achieve the real-time performance, the use of full resolution 3D images is avoided. Instead, a set of feature points obtained during the motion estimation process is used. A probabilistic approach is then used to maintain the consistency of the feature points and the MCU location in real-time. The result is a consistent feature point map that represents the geometric structure of the environment and the estimated location of the MCU. By knowing the MCU location, every newly acquired 3D image

can be merged to create a 3D photorealistic map of the environment. The inputs of the map building process are the motion estimation information, the feature points and the 3D images from the stereo cameras. The map building process is illustrated in Figure 5.1.

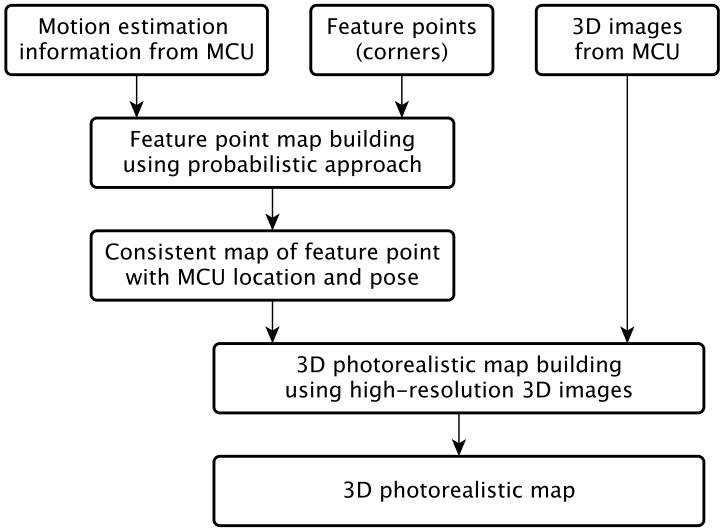


**Figure 5.1: Overview of the map building process.**

## 5.2 Real-time 3D Map Building using a Probabilistic Approach

Generally, two main components are needed for the creation of a complete 3D map. The first one is the set of feature points that are extracted from the environment and the second one is the rigid transformation information that relates the two sets of feature points from different time steps to the same reference coordinate. If both components are available, a 3D map can be constructed without difficulty. However, there are problems when dealing with map building in real world situations. These problems are:

- The measurement of the feature point location is corrupted with noise.
- The motion information contains uncertainty.

Failing to cope with these uncertainties usually results in the instability of the map building algorithm and a corrupted map.

In order to deal with these uncertainties the probabilistic approach for map building is needed. There is a framework that deals with the mapping and motion estimation uncertainties, namely the Simultaneous Localization And Mapping (SLAM). One of the well known SLAM implementations is based on the Extended Kalman Filter (EKF). However the design of the EKF itself poses a problem for the map building task, namely the computational complexity of the system matrices. A straightforward implementation of the EKF-based SLAM requires time quadratics related to the number of features in the map, which is the result of the massive system matrix multiplications [Thrun (2002)]. The EKF approach also copes poorly with the correspondence problem in which the wrong association of the feature points measurement and the map data can result in the instability of the algorithm.

The FastSLAM[1] algorithm is another probabilistic approach that is realized based on the particle filter technique [Montemerlo *et al.* (2002), Montemerlo (2003), Montemerlo *et al.* (2003), Bailey *et al.* (2006), Sim *et al.* (2005)]. It copes with the correspondence problem by introducing multiple hypotheses on the observer localization and the associated feature point map. FastSLAM contains set of particles that individually keep track of the path history and the feature point map. It uses particle filter to sample the new pose and uses low-dimension EKFs to maintain the location of each individual feature point in the map. This design requires a smaller memory footprint and shorter computational time compared to the EKF-based SLAM approach [Montemerlo *et al.* (2003)]. For these reasons the FastSLAM algorithm is chosen for the map building task in this work.

[1]Currently there are two implementations of the FastSLAM algorithms, which are FastSLAM 1.0 and FastSLAM 2.0. Unless otherwise stated, the term FastSLAM in this work refers to the FastSLAM 2.0 implementation.

## 5.3 FastSLAM

### 5.3.1 Overview

The objective of the FastSLAM algorithm is to integrate the new feature point measurement into a consistent feature point map and at the same time maintain the location of the MCU related to the map coordinate system. The main information within the FastSLAM algorithm are the *path* of the MCU denoted by $s^t$ and the *feature point map* denoted by $\Theta$. Both are being maintained by a set of particles within the FastSLAM algorithm. The path $s^t$ can be written as a set of the MCU pose vectors from the first until the latest time step:

$$s^t = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, ..., \mathbf{s}_t\} \tag{5.1}$$

The feature point map $\Theta$ is a set of $N$ feature point locations and it can be written as:

$$\Theta = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, ..., \boldsymbol{\theta}_N\} \tag{5.2}$$

In the FastSLAM algorithm, the posterior probability of the feature point map given the MCU path is expressed as:

$$p(\Theta, s^t | z^t, u^t, n^t) \tag{5.3}$$

where

| | |
|---|---|
| $z^t$ | set of the feature points measurement $z^t = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_t\}$ |
| $u^t$ | set of the MCU odometry measurement $u^t = \{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_t\}$ |
| $n^t$ | set of the data associate variables $n^t = \{n_1, n_2, ..., n_t\}$ |

The probabilistic model of the feature points measurement is:

$$p(\mathbf{z}_t | \mathbf{s}_t, \Theta, n_t) \tag{5.4}$$

and the probabilistic motion model of the MCU is:

$$p(\mathbf{s}_t | \mathbf{u}_t, \mathbf{s}_{t-1}) \tag{5.5}$$

Both probabilistic models are modeled by non-linear functions $\mathbf{g}$ and $\mathbf{h}$ with independent Gaussian noise $\boldsymbol{\varepsilon}_t$ and $\boldsymbol{\delta}_t$ and the covariance $\boldsymbol{\Sigma}_\varepsilon$ and $\boldsymbol{\Sigma}_\delta$ respectively.

$$p(\mathbf{z}_t | \mathbf{s}_t, \Theta, n_t) = \mathbf{g}(\mathbf{s}_t, \boldsymbol{\theta}_{n,t}) + \boldsymbol{\varepsilon}_t \tag{5.6}$$

$$p(\mathbf{s}_t | \mathbf{u}_t, \mathbf{s}_{t-1}) = \mathbf{h}(\mathbf{s}_{t-1}, \mathbf{u}_t) + \boldsymbol{\delta}_t \tag{5.7}$$

The FastSLAM algorithm solves the feature point location estimation independently from the MCU pose estimation. That is, assuming that the path of the MCU is known, the location of each feature point can then be estimated independently. Therefore the factorization of the posterior distribution in Equation 5.3 becomes:

$$p(\Theta, s^t|z^t, u^t, n^t) = p(s^t|z^t, u^t, n^t) \prod_{n=1}^{N} p(\boldsymbol{\theta}_n|s^t, z^t, u^t, n^t) \tag{5.8}$$

where $p(s^t|z^t, u^t, n^t)$ is the path posterior and $p(\boldsymbol{\theta}_n|s^t, z^t, u^t, n^t)$ is the feature point location estimator.

The FastSLAM algorithm samples the path using a particle filtering technique where a number of $m$ particles are used to maintain $m$ unique MCU paths and $m$ feature point maps. Figure 5.2 illustrates the components of each FastSLAM particle where $\boldsymbol{\mu}_{n,t}^{[m]}$ and $\boldsymbol{\Sigma}_{n,t}^{[m]}$ are the mean and covariance matrix representing the $n^{th}$ feature location conditioned on the path $s^{t,[m]}$ at time $t$.

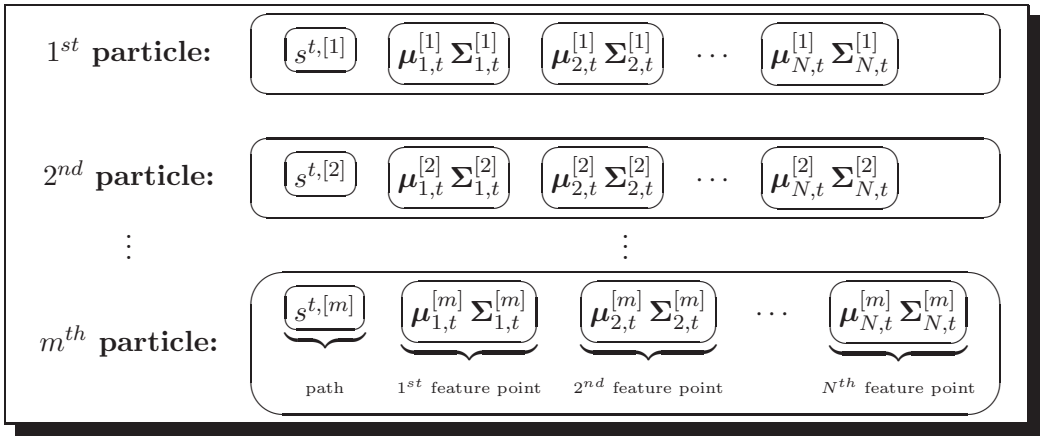


**Figure 5.2: The FastSLAM with $m$ particles.**

## 5.3.2 The FastSLAM Algorithm

At every iteration, the FastSLAM algorithm samples the new MCU pose $\mathbf{s}_t^{[m]}$ based on the set of MCU pose up to the previous iteration ($s^{t-1,[m]}$), the MCU odometry measurement, the measurement of the feature point locations and the data association

variable:

$$\mathbf{s}_t^{[m]} \sim p(\mathbf{s}_t | s^{t-1,[m]}, u^t, z^t, n^t) \tag{5.9}$$

The measurement function $\mathbf{g}(\mathbf{s}_t, \boldsymbol{\theta}_{n,t})$ is approximated by linear functions so that the distribution is Gaussian and a standard EKF solution can be applied. The Taylor expansion of $\mathbf{g}$ is:

$$\mathbf{g}(\mathbf{s}_t, \boldsymbol{\theta}_{n,t}) \approx \hat{\mathbf{z}}_t + \mathbf{G}_\theta(\boldsymbol{\theta}_{n,t} - \boldsymbol{\mu}_{n,t-1}^{[m]}) + \mathbf{G}_s(\mathbf{s}_t - \hat{\mathbf{s}}_t) \tag{5.10}$$

Where $\hat{\mathbf{s}}_t$ and $\hat{\mathbf{z}}_t$ are the predicted pose and predicted measurement and the matrices $\mathbf{G}_\theta$ and $\mathbf{G}_s$ are the Jacobians of $\mathbf{g}$. By using the linear approximation of the measurement function $\mathbf{g}$, the covariance $\boldsymbol{\Sigma}_{s_t}^{[m]}$ and the mean $\boldsymbol{\mu}_{s_t}^{[m]}$ of the MCU pose are then determined:

$$\boldsymbol{\Sigma}_{s_t}^{[m]} = [\mathbf{G}_s^T \mathbf{Z}_t^{-1} \mathbf{G}_s + \boldsymbol{\Sigma}_{\delta,t}^{-1}]^{-1} \tag{5.11}$$

$$\boldsymbol{\mu}_{s_t}^{[m]} = \boldsymbol{\Sigma}_{s_t}^{[m]} \mathbf{G}_s^T \mathbf{Z}_t^{-1} (\mathbf{z}_t - \hat{\mathbf{z}}_t) + \hat{\mathbf{s}}_t^{[m]} \tag{5.12}$$

$$\mathbf{Z}_t = \boldsymbol{\Sigma}_{\varepsilon,t} + \mathbf{G}_\theta \boldsymbol{\Sigma}_{n,t-1}^{[m]} \mathbf{G}_\theta^T \tag{5.13}$$

Next, the FastSLAM algorithm updates the observation of the feature points according to the following posterior:

$$p(\boldsymbol{\theta}_{n,t} | s^{t,[m]}, n^t, z^t) = \eta p(\mathbf{z}_t | \boldsymbol{\theta}_{n,t}, \mathbf{s}_t^{[m]}, n_t) p(\boldsymbol{\theta}_{n,t} | s^{t-1,[m]}, z^{t-1}, n^{t-1}) \tag{5.14}$$

where $\eta$ is the normalizer constant. By using the linearized measurement function $\mathbf{g}$, the mean $\boldsymbol{\mu}_{n,t}^{[m]}$ and the covariance $\boldsymbol{\Sigma}_{n,t}^{[m]}$ of all feature point locations are updated using EKF.

$$\boldsymbol{\mu}_{n,t}^{[m]} = \boldsymbol{\mu}_{n,t-1}^{[m]} + \mathbf{K}_t(\mathbf{z}_t - \hat{\mathbf{z}}_t) \tag{5.15}$$

$$\boldsymbol{\Sigma}_{n,t}^{[m]} = (\mathbf{I} - \mathbf{K}_t \mathbf{G}_{\theta_{n,t}}) \boldsymbol{\Sigma}_{n,t-1}^{[m]} \tag{5.16}$$

The FastSLAM algorithm recalculates the weight of all particles at every iteration. The probability for the $m^{th}$ particle to be sampled is given by its importance weight $w_t^{[m]}$, which is defined by the ratio of the target distribution over the proposal distribution:

$$w_t^{[m]} = \frac{\textit{target distribution}}{\textit{proposal distribution}} \tag{5.17}$$

$$= \frac{p(s^{t,[m]} \mid z^t, u^t, n^t)}{p(s^{t-1,[m]} \mid z^{t-1}, u^{t-1}, n^{t-1}) p(\mathbf{s}_t^{[m]} \mid s^{t-1,[m]}, z^t, u^t, n^t)} \tag{5.18}$$

For the complete information about the FastSLAM algorithm please refer to Montemerlo (2003).

## 5.4 MCU System Equations

The MCU pose can be represented by the pose vector $\mathbf{x}_v$, which consists of the position of the MCU related to the 3D world coordinate $(x, y, z)$, and the orientation $(\alpha, \beta, \gamma)$, which represent the relative angle to the x-, y- and z-axes of the world coordinate using the right hand rule.

$$\mathbf{x}_v = [x_v \quad y_v \quad z_v \quad \alpha_v \quad \beta_v \quad \gamma_v]^T \tag{5.19}$$

The odometry output from the MCU at each time step is denoted as $\mathbf{u}_t$ and it is consists of the translation and rotation parameters:

$$\mathbf{u}_t = [x_u \quad y_u \quad z_u \quad \alpha_u \quad \beta_u \quad \gamma_u]^T \tag{5.20}$$

Each feature point is described by a vector $\mathbf{y}_n$, which simply consists of the $x$, $y$ and $z$ components. The subscribe $n = \{1, 2, ..., N\}$ denotes the number of the feature points at the current time step $t$. This can be written as:

$$\mathbf{y}_n = [x_n \quad y_n \quad z_n]^T \tag{5.21}$$

### The Motion Model

The pose of the MCU ($\mathbf{x}_v$) at each time step $t$ can be derived by:

$$\mathbf{x}_{v_t} = \mathbf{f}_v(\mathbf{x}_{v_{t-1}}, \mathbf{u}_t) \tag{5.22}$$

The transition function $\mathbf{f}_v$ can be written using the previously defined coordinate system as:

$$\mathbf{f}_v(\mathbf{x}_{v_{t-1}}, \mathbf{u}_t) = \begin{bmatrix} x_{v_{t-1}} + R_{11}x_u + R_{12}y_u + R_{13}z_u \\ y_{v_{t-1}} + R_{21}x_u + R_{22}y_u + R_{23}z_u \\ z_{v_{t-1}} + R_{31}x_u + R_{32}y_u + R_{33}z_u \\ \alpha_{v_{t-1}} + \alpha_u \\ \beta_{v_{t-1}} + \beta_u \\ \gamma_{v_{t-1}} + \gamma_u \end{bmatrix} \tag{5.23}$$

where the terms $R_{ij}$ are the corresponding entries of the 3D rotation matrix $\mathbf{R}$:

$$\begin{aligned}\mathbf{R} &= \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} \\ &= \begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{pmatrix}\end{aligned} \tag{5.24}$$

### The Measurement Model

The measurement $\mathbf{z}_n$ of the $n^{th}$ feature point $\mathbf{y}_n$ is defined by:

$$\mathbf{z}_n = \mathbf{h}_n(\mathbf{x}_v, \mathbf{y}_n) \tag{5.25}$$

where $\mathbf{x}_v$ is the pose of the MCU and $\mathbf{y}_n$ is the 3D coordinate of the feature point.

## 5.5 FastSLAM Implementation

The implementation of the FastSLAM algorithm using MCU consists of the following process:

- Update the current MCU pose and the new measurement.
- Data association of known feature points using lookup table.
- Add new feature points to the map.
- Compute weight of the particles.
- Resample the particles.
- Calculate sample proposal.
- Update known feature points using EKFs.

The relationship of these process is shown in Figure 5.3.

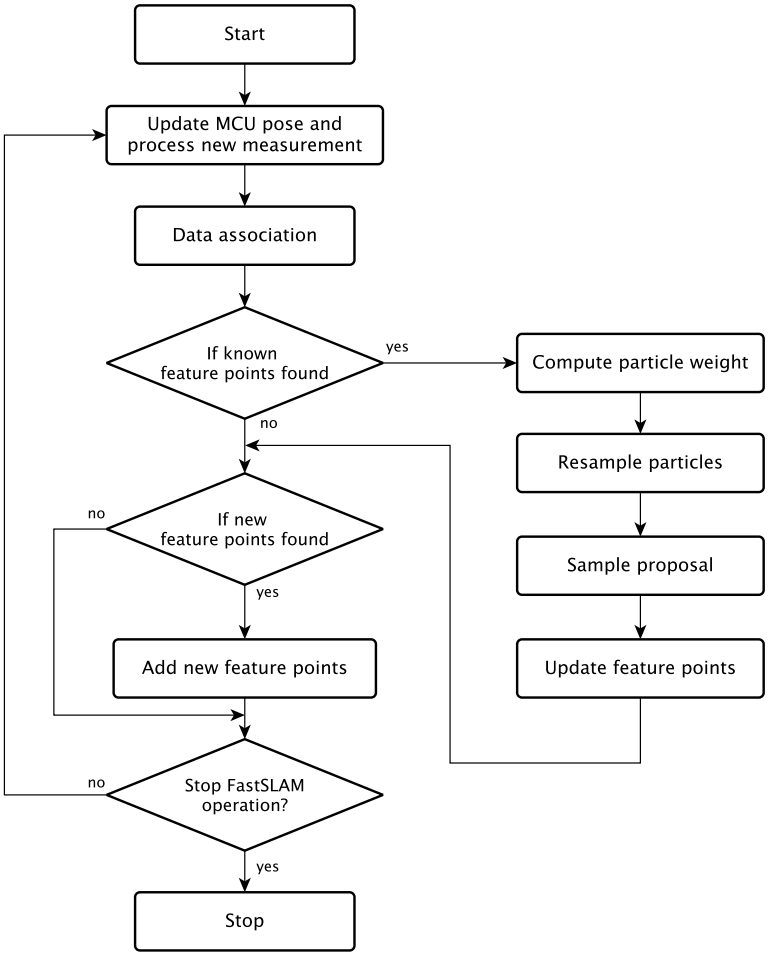


**Figure 5.3: An overview of the FastSLAM algorithm.**

## Update the Current MCU Pose and the New Measurement

The pose of the MCU and the new measurement are updated according to Equation 5.23 and Equation 5.25 respectively.

## Data Association

This process associates the new feature point measurements to the known feature points using a look-up table. If a known feature point is found then it will be updated by the following process. Unknown feature points will be added to the map by the add new feature point process.

### Add New Feature Point

This process adds new feature points to the feature point map. An index number is then assigned for the new feature point and is kept in the look-up table.

### Compute Weight of the Particles

The new weights of the particles are calculated according to Equation 5.18.

### Resample Particles

This process draws a new set of samples from the current particles set with probabilities in proportion to the newly computed weight. Resampling is done in order to avoid the depletion of the particles.

### Sample Proposal

The new sample proposal of the MCU pose is derived according to Equation 5.11 and Equation 5.12

### Update Known Feature Points

This process updates the mean and covariance of the feature points in every particles according to Equation 5.15 and Equation 5.16 respectively.

## 5.6 Simulation of the FastSLAM System

A software simulation of the FastSLAM algorithm is implemented using MATLAB in order to investigate the behavior of the algorithm. The simulation includes the full implementation of the FastSLAM algorithm and the 3D data visualization. The simulation provides a set of 3D feature points and a pre-defined trajectory that the simulated 3D sensor has to follow. Gaussian noise is introduced to the measurement of the feature point location as well as the estimated motion information.

At each iteration the simulated 3D sensor observes the feature points that are seen within its field of view and updates its pose and location according to a simple motion model. The pose and location of the 3D sensor and the location of the feature points are then used as input for the FastSLAM algorithm. The output is the probabilistic map of the feature points as well as the estimated location of the 3D sensor.

The following important parameters are used for the simulation system:

- Number of particles used is 500.
- The motion of the 3D sensor along the defined trajectory consists of pure translation along the path and pure rotation around the turning points.
- The estimated motion information contains uncertainties of 20 mm and 0.5 degrees in cases of translation and rotation respectively.
- The maximum translation and rotation of the 3D sensor per iteration are limited to 10 mm and 3 degrees respectively.
- The feature point measurement contains uncertainly of 20 mm in the x, y and z direction.
- The 3D sensor has to complete the whole trajectory twice.

## Results

The motion information including the location and pose of the 3D sensor are recorded during each iteration and the error between the estimated and actual values are calculated. Figure 5.4 shows the motion estimation errors during two completions around the defined trajectory. It can be seen that the estimated values are very close to the actual values.

In Figure 5.5, the sum of trace of the covariance matrices of the feature points are calculated and plotted against iteration steps in order to observe the consistency of the derived feature point map. From the start to the end of the first completion of the trajectory (from the $1^{st}$ to the $325^{th}$ iteration), new feature points are being added to the map, as can be seen by the increasing sum of trace value which eventually decreased after several iterations. As the 3D sensor begins its second round of the trajectory (after the $325^{th}$ iteration), there are no more new features being added to the map and it can be seen that the sum of trace converges and gets smaller as more iterations pass by. This shows that the consistency of the map is continuing to improved over time.

Figure 5.6 shows the graphic representation obtained from the FastSLAM simulation during the simulation process. In Figure 5.6(a), the estimated location of feature points obtained from each particle are distributed around the actual locations and these estimated locations converge to the actual locations as more iterations pass by (Figure 5.6(b), Figure 5.6(c) and Figure 5.6(d)). In Figure 5.6(e), several new features are

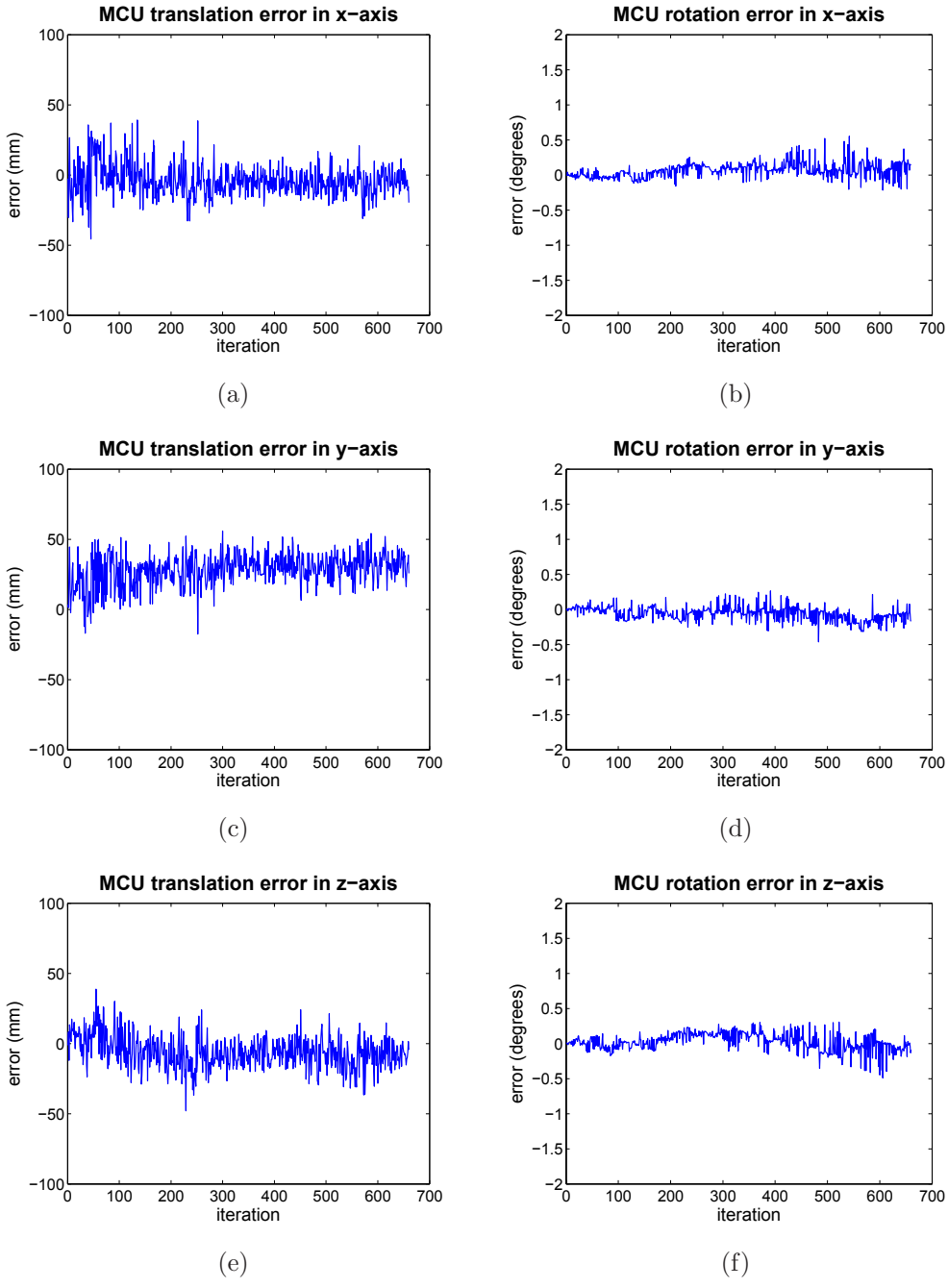


**Figure 5.4: The pose and location error obtained from the FastSLAM simulation.**

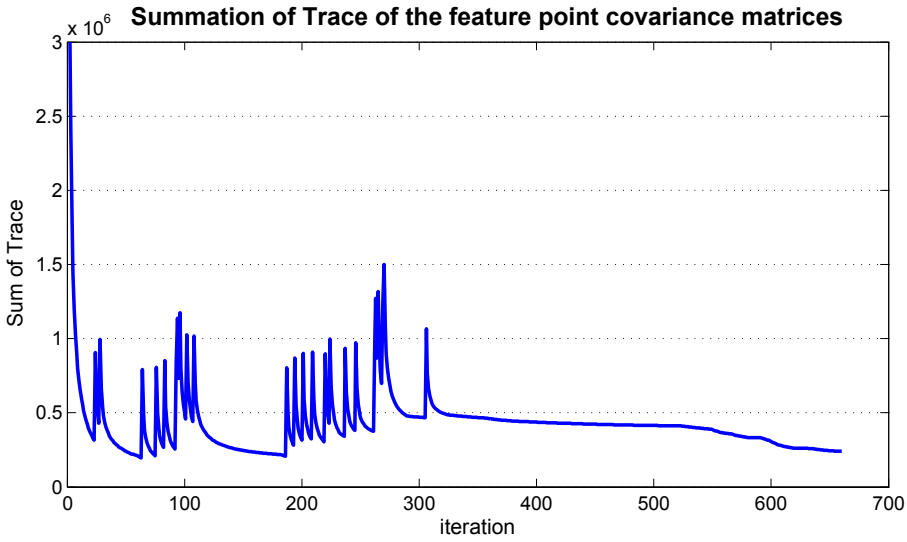


**Figure 5.5: The sum of trace of the feature point covariance matrices.**

added to the map and their locations also converge to the actual locations as shown in Figure 5.6(f).

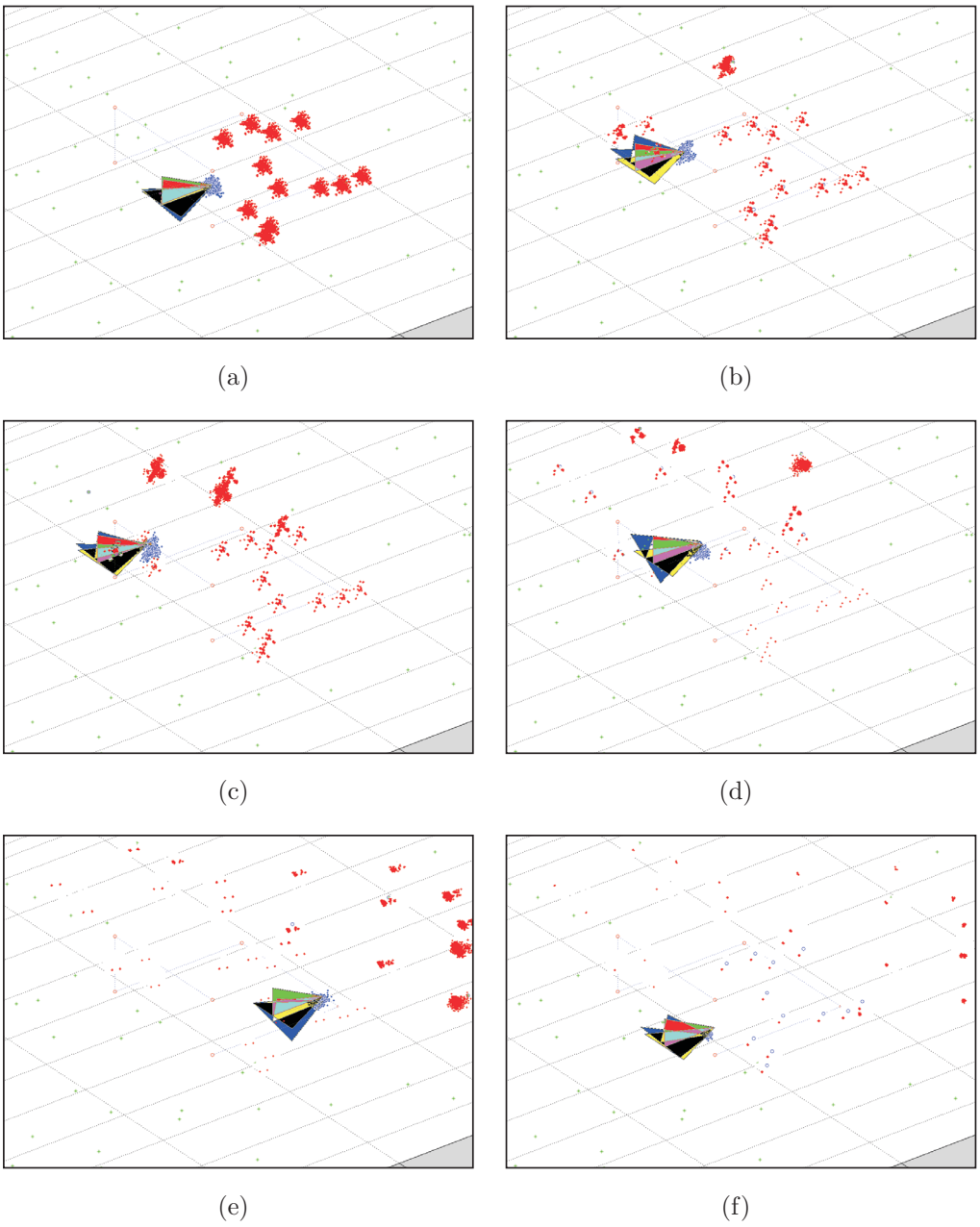


**Figure 5.6: Result of the FastSLAM algorithm simulation implemented in MATLAB using 500 particles. The yellow-green triangle polygon indicates the actual pose of the 3D sensor while the blue triangle polygon indicates the estimated pose. The green markers represent the actual locations of the feature points and the red markers represents the estimated values as derived from every particles. The result shows the convergence of the feature point's location as well as the accurately estimated pose of the 3D sensor compared to the actual one.**

## 5.7 Real-time 3D Photorealistic Map Building using 3D Images

The procedure to obtain the 3D photorealistic map of the environment is to incrementally merge the 3D images acquired from the MCU together according to the location and pose of the MCU. This process is illustrated in Figure 5.7 where new 3D images $(X_i, Y_i, Z_i)$ are being added to the map according to the pose $P_i$ of the MCU.

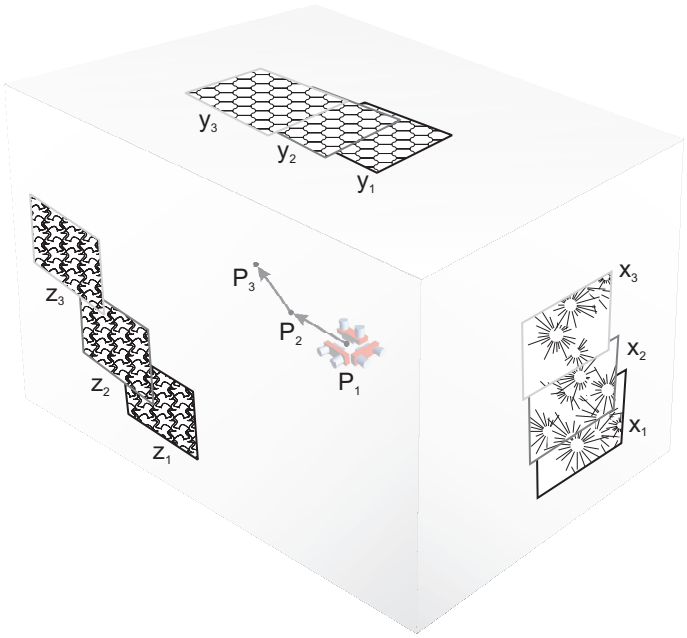


**Figure 5.7: The goal of the map building process is to incrementally build a photorealistic 3D map of the environment by merging new 3D images to the map data.**

The current implementation of the 3D photorealistic map building is simple. First the 3D images or the clouds of 3D points acquired at each time step are stored on the hard drive as map tiles. At the same time the location and pose of the MCU is also recorded. Then the map rendering software periodically fetches these files and adds the 3D points from the map tile files to the map data and renders the 3D photorealistic model in real-time. These processes are shown in Figure 5.8.

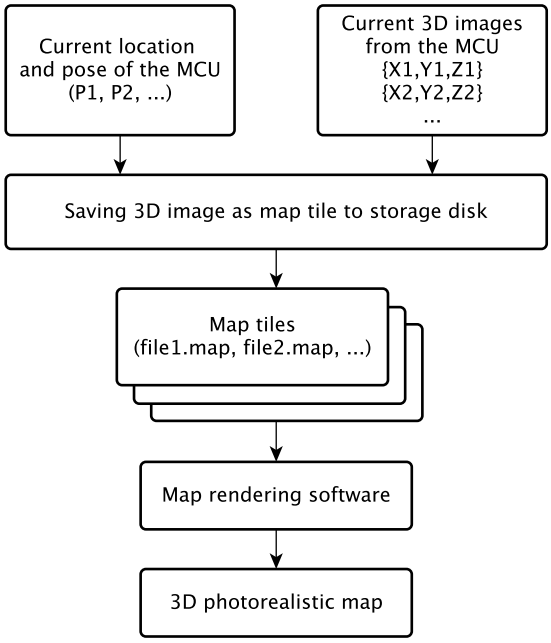


**Figure 5.8: The 3D photorealistic map building process.**

### 5.7.1 MCU Log File

The MCU log file contains the location and pose of the MCU derived at every time step from the FastSLAM algorithm. This information is stored on a hard drive as a text file. Figure 5.9 shows the log file format used in this work.

```
<24490, 0, 0, 0, 0, 0, 0,  0.00287006, 0.0008779,  0.0049752,  0.282576, -0.095407, -0.072662>
<25080, 0, 0, 0, 0, 0, 0, -0.00726685, 0.0090541,  0.0101487,  0.285235, -0.302657,  0.234023>
<25722, 0, 0, 0, 0, 0, 0, -0.00612231, 0.0096857,  0.0106534,  0.282811, -0.325240,  0.211693>
<26324, 0, 0, 0, 0, 0, 0, -0.00334177, 0.0039395,  0.0034720,  0.015731, -0.328844,  0.055948>
<26909, 0, 0, 0, 0, 0, 0, -0.00466342, 0.0029244, -9.6588605, -0.048754, -0.122407,  0.029919>

< a , b,c,d,e,f,g,   h    ,   i   ,   j   ,   k   ,   l   ,   m  >
```

**Figure 5.9: Example of the MCU log file. This example log file contains five entries where various information about the MCU during five time steps are recorded.**

The MCU log file contains the following information:

- (a): time stamp (in millisecond).

- (b - g): movement directions of the MCU (according to Section 4.4).
- (h, i, j): orientation of the MCU (roll, pitch, yaw) in degree.
- (k, l, m): position of the MCU (x, y, z) in millimeter.

The log file can be used to generate a travelled trajectory of the MCU. By rendering every MCU position available in the log file using the map rendering software, the travelled trajectory of the MCU can be easily inspected. Example trajectories can be found in the results of the map building experiments in the next chapter.

### 5.7.2 Map Tiles

The map tiles are text files that contain the clouds of 3D points captured at each time step by the MCU. These tiles can be put together in order to create a photorealistic model of the explored environment. Figure 5.10 shows the file format of the map tile used in this work.

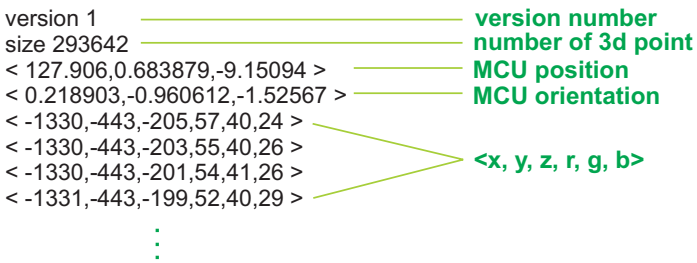


**Figure 5.10: Example of a map tile file. Each file consists of the header information and the entries of the 3D points. Only the first four entries are shown in this example.**

As can be seen, the header of the map tile contains the information about MCU location and orientation at the capture time. This makes it easy to merge several map tiles together afterwards since the location and orientation information are all referred to the same origin.

The header contains the following information:

- **Version number:** this gives the map building system a flexibility to handle possible file format changes in the future.

- **Number of 3D point:** this indicates the total number of 3D points available in the file.
- **MCU position:** the position of the MCU (x, y, z) in millimeter.
- **MCU orientation:** the orientation of the MCU (roll, pitch, yaw) in degree.

The rest of the file is the entries of the 3D points where each line represents one 3D point. The information for each point consists of the x, y and z location as well as the color information which is stored according to the RGB color format.

### 5.7.3 3D Map Rendering Software

The map rendering software is written by the author in order to visualize the 3D map building process in real-time. The 3D points are rendered using the OpenGL graphic library. Figure 5.11 shows the map rendering software used in this work.

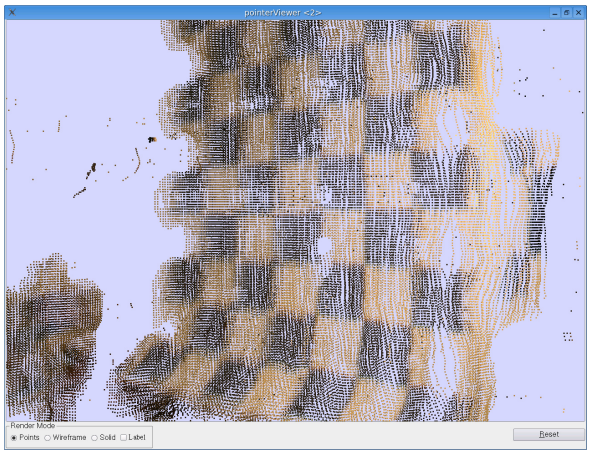


**Figure 5.11: The 3D map rendering software.**

By following the procedures shown in Figure 5.8, the 3D map can be incrementally built. Figure 5.12 shows the progress of the map building process as being visualized by the map rendering software.

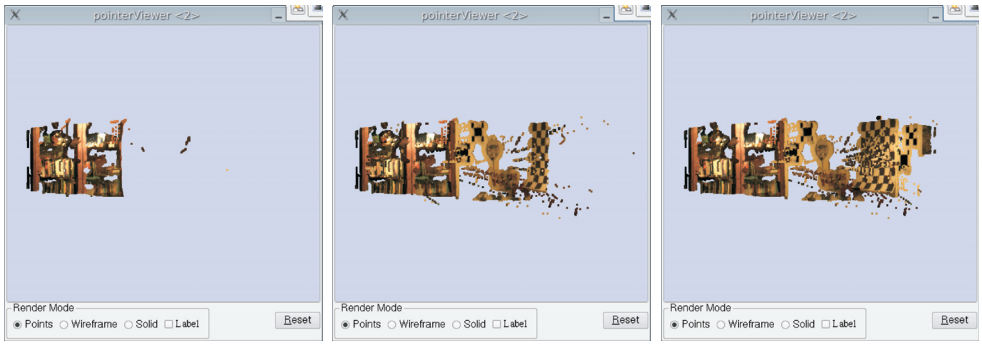


**Figure 5.12: The map building progress. First the map rendering software renders only the first map tile (left). As time pass by, more map tiles are being added (middle), resulting in a 3D photorealistic map that keep expanding over time (right).**

## 5.8 Conclusion

This chapter explained the map building process which is based on the probabilistic analysis approach. The goal is to maintain a consistent map of the environment as well as the location of the MCU and its travelled trajectory. The feature set derived during the motion estimation process is used as a sparse representation of the environment in order to archive the real-time map building performance. The location of the feature points as well as the location and pose of the MCU are maintained consistent with the real world values using the FastSLAM algorithm. A MATLAB simulation of the FastSLAM algorithm shows the convergence of the map as well as a consistent MCU localization. Finally, the 3D photorealistic map of the environment is obtained by merging the 3D images acquired by the MCU during the exploration time.

# Chapter 6

# Experiments and Results

***Introduction — Experiment Setup — 3D Motion Estimation Experiments and Results — 3D Photorealistic Map Building Experiments and Results — Conclusion***

## 6.1 Introduction

This chapter presents the experiment results of the 3D motion estimation and map building tasks using the MCU system. In the motion estimation experiments the MCU is moved along the pre-defined trajectories and the estimated 3D motion is to be determined using the procedures described in Chapter 4. In the map building experiments, the MCU is used to explore an environment where knowledge about the environment is zero at the start and the procedures described in Chapter 5 are used to produce the 3D photorealistic maps of the environment.

## 6.2 Experiment Setup

### 6.2.1 Assumptions

Certain assumptions were made for the 3D motion estimation and map building experiments using the MCU system. The following assumptions are required so that the MCU can operate correctly for both 3D position estimation and map building tasks.

- The test environment should be an indoor environment. This is to make sure that undesired effects such as strong sunlight, reflections, unexpected moving objects, etc. will not interfere with the experiment, since the current implementation of the MCU system is not designed to cope with such strong disturbances.
- The environment to be explored should be static. No moving objects and no changing of the geometry within the environment occurs during the experiment.
- The environment provides enough visual texture for the stereo camera. The texture information is necessary for the visual odometry process. Environments without any visual texture, e.g. a white room with empty walls would void the use of MCU since the visual odometry process would fail to find any feature points.
- The test environment should contain no mirrors since its reflection could give the MCU false geometric information and bring about a corruption of the data scheme.
- The environment is well lit, i.e. adequate light is present so that the camera can produces proper image with low noise.

### 6.2.2 The Test Environment

Figure 6.1 and Figure 6.2 show the test environment to be used for the experiments. The test environment has an approximate dimension of approximately $4 \times 5 \times 3$ meters and it contains no specific or artificial landmarks.

### 6.2.3 The Measurement Tool

The measurement scales are attached to the tripod that holds the MCU hardware. They are used for the measurement of the actual rotation and translation of the MCU

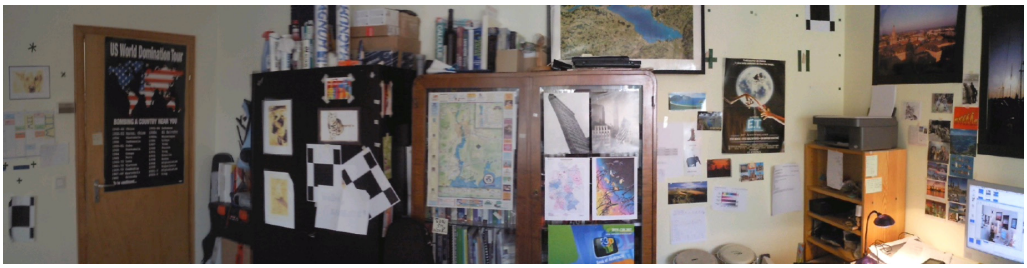


**Figure 6.1: A panoramic view of the test environment.**


**Figure 6.2: Some different views from the test environment.**

after each adjustment on the tripod during the experiments. Figure 6.3 shows the tripod and the measurement scales.

### 6.2.4 The Test Procedures

The following procedures are to be carried out during each test.

1. Initialization of the MCU system and the map data. Reset the starting MCU position to the origin.

**Figure 6.3: The setup for the displacement measurement tool.**

2. Navigate the MCU along the planned trajectory.

3. During each movement, the MCU estimates the 3D motion and incrementally updates the feature point map. The trajectory and the feature point map are recorded until the MCU finishes the given trajectory.

## 6.3 Experiment I: 3D Motion Estimation

In this experiment a set of different trajectories are used in order to test the performance of the MCU system for the 3D motion estimation task. The 3D motion estimation is derived using information from one camera, two cameras and three cameras respectively so that the performance between each system can be compared.

The trajectory and the number of cameras used during each test are listed below.

### 6.3.1 Small Rotation Estimation

**Test 1:** Estimate 15 degrees rotation around the y-axis using only X-camera.

**Test 2:** Estimate 15 degrees rotation around the y-axis using only Y-camera.

**Test 3:** Estimate 15 degrees rotation around the y-axis using only Z-camera.

**Test 4:** Estimate 15 degrees rotation around the y-axis using X- and Z-cameras.

**Test 5:** Estimate 15 degrees rotation around the y-axis using all three cameras.

**Test 6:** Estimate 25 degrees rotation around the y-axis using all three cameras.

### 6.3.2 Small Translation Estimation

**Test 1:** Estimate 50 mm translation along the y-axis using only Y-camera.

**Test 2:** Estimate 50 mm translation along the y-axis using only Z-camera.

**Test 3:** Estimate 50 mm translation along the y-axis using X- and Z-cameras.

**Test 4:** Estimate 50 mm translation along the y-axis using all three cameras.

**Test 5:** Estimate 10 mm translation along the y-axis using all three cameras.

**Test 6:** Estimate 100 mm translation along the y-axis using all three cameras.

### 6.3.3 Large Rotation Estimation

**Test 1:** Estimate 25 degrees rotation around the x-axis using all three cameras.

**Test 2:** Estimate 45 degrees rotation around the y-axis using all three cameras.

### 6.3.4 Large Translation Estimation

**Test 1:** Estimate 500 mm translation along the x-axis using all three cameras.

**Test 2:** Estimate 1000 mm translation along the z-axis using all three cameras.

### 6.3.5 $\ell^2$-shape Trajectory Estimation

Estimate the movement along the $\ell^2$-shape trajectory using all three cameras.

### 6.3.6 Additional Trajectories

**Test 1:** Rotation 90 degrees around the y-axis using all three cameras.

**Test 2:** Rotation 180 degrees around the y-axis using all three cameras.

**Test 3:** Movement along a trapezoidal trajectory using all three cameras.

### 6.3.1 Small Rotation

The following 3D motion estimation tests are done using some small rotations around the y-axis of the MCU reference frame. The MCU is rotated back and forth according to the pre-defined angle and the 3D motion information during the movement is recorded. The tests also includes the use of a one camera system, a two camera system and a three camera system in order to compare the different performances. The following tests are included in this section:

**Test 1:** Estimate 15 degrees rotation around the y-axis using only X-camera.

**Test 2:** Estimate 15 degrees rotation around the y-axis using only Y-camera.

**Test 3:** Estimate 15 degrees rotation around the y-axis using only Z-camera.

**Test 4:** Estimate 15 degrees rotation around the y-axis using X- and Z-cameras.

**Test 5:** Estimate 15 degrees rotation around the y-axis using all three cameras.

**Test 6:** Estimate 25 degrees rotation around the y-axis using all three cameras.

### Test 1: Estimate 15 degrees rotation around the y-axis using only X-camera.

In this test the MCU is rotated 15 degrees back and forth around the y-axis. Figure 6.4 shows the results of the 3D motion estimation derived using only information derived from the X-camera.

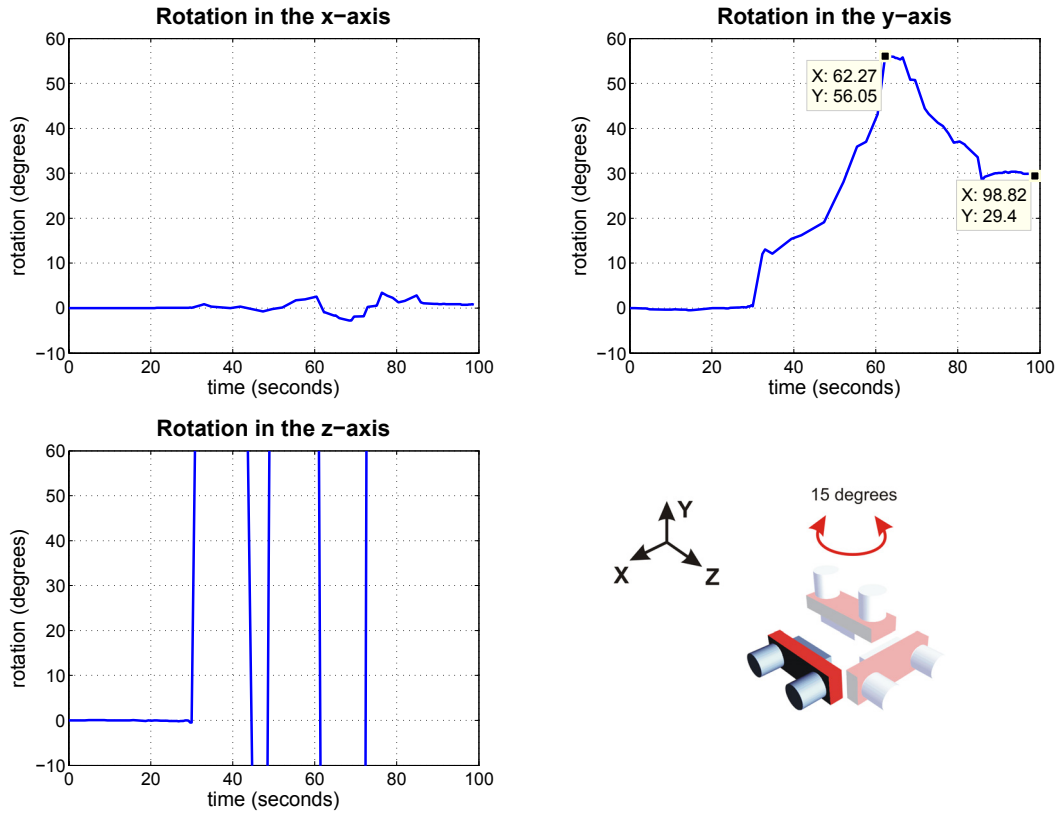


**Figure 6.4: Results of the rotation estimation using 15 degrees of rotation around the y-axis. Only the X-camera is used.**

### Test 2: Estimate 15 degrees rotation around the y-axis using only Y-camera.

In this test the MCU is rotated 15 degrees forth and back around y-axis. Figure 6.5 shows the results of the 3D motion estimation derived using only information derived from the Y-camera.

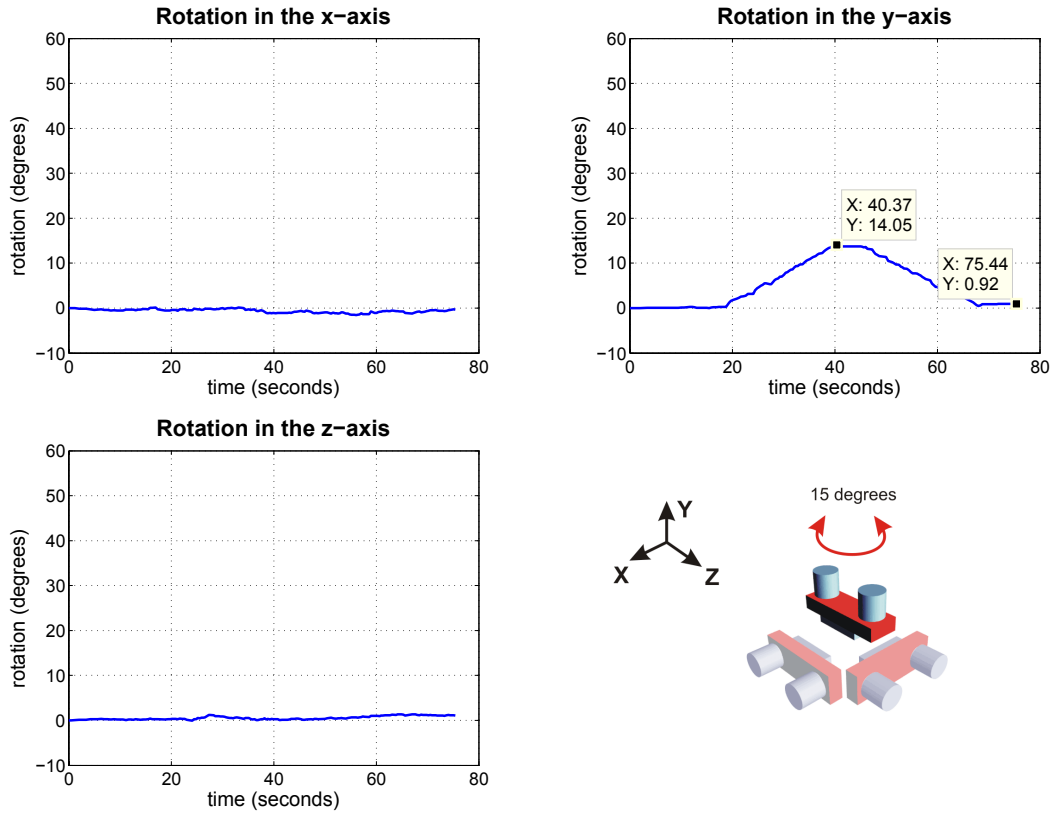


**Figure 6.5: Results of the rotation estimation using 15 degrees of rotation around the y-axis. Only the Y-camera is used.**

### Test 3: Estimate 15 degrees rotation around the y-axis using only Z-camera.

In this test the MCU is rotated 15 degrees back and forth around the y-axis. Figure 6.6 shows the results of the 3D motion estimation derived using only information derived from the Z-camera.

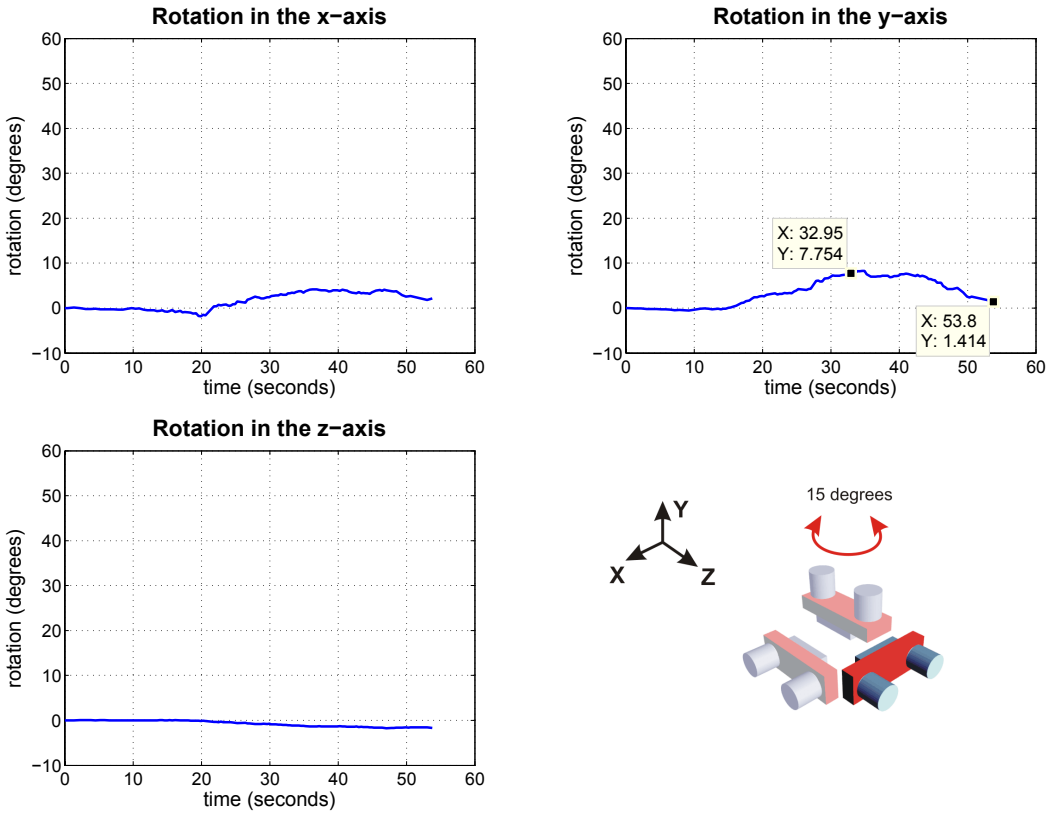


**Figure 6.6: Results of the rotation estimation using 15 degrees of rotation around the y-axis. Only the Z-camera is used.**

### Test 4: Estimate 15 degrees rotation around the y-axis using X- and Z-cameras.

In this test the MCU is rotated 15 degrees back and forth around the y-axis. Figure 6.7 shows the results of the 3D motion estimation derived using information derived from X- and Z-cameras.

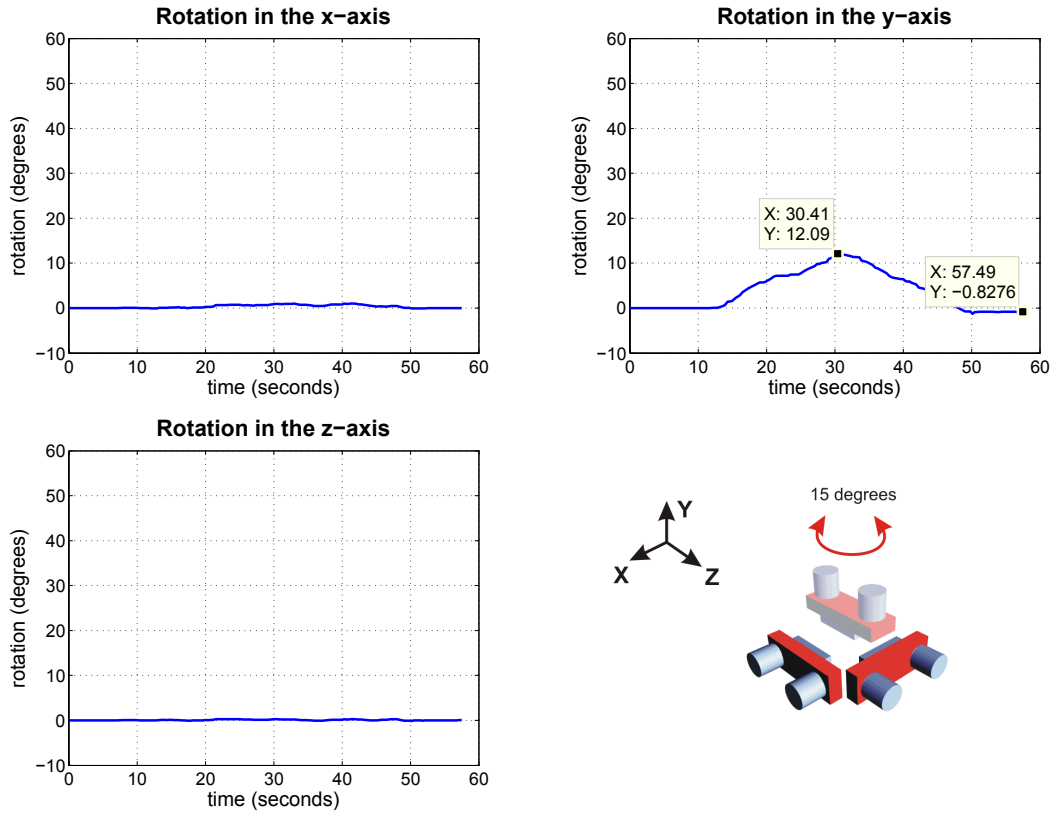


**Figure 6.7: Results of the rotation estimation using 15 degrees of rotation around the y-axis. The X- and Z-cameras are used.**

### Test 5: Estimate 15 degrees rotation around the y-axis using all three cameras.

In this test the MCU is rotated 15 degrees back and forth around the y-axis. Figure 6.8 shows the results of the 3D motion estimation derived using information derived from all three cameras.

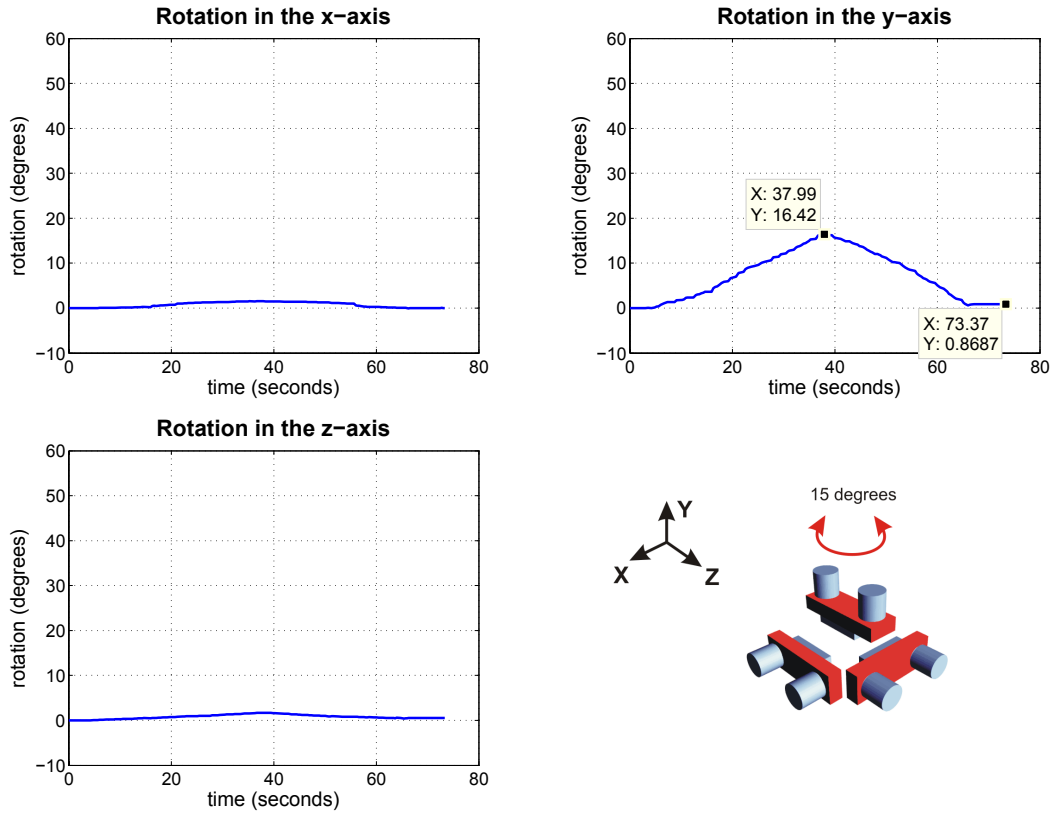


**Figure 6.8: Results of the rotation estimation using 15 degrees of rotation around the y-axis. All three cameras are used.**

### Test 6: Estimate 25 degrees rotation around the y-axis using all three cameras.

In this test the MCU is rotated 25 degrees back and forth around the y-axis. Figure 6.9 shows the results of the 3D motion estimation derived using information derived from all three cameras.

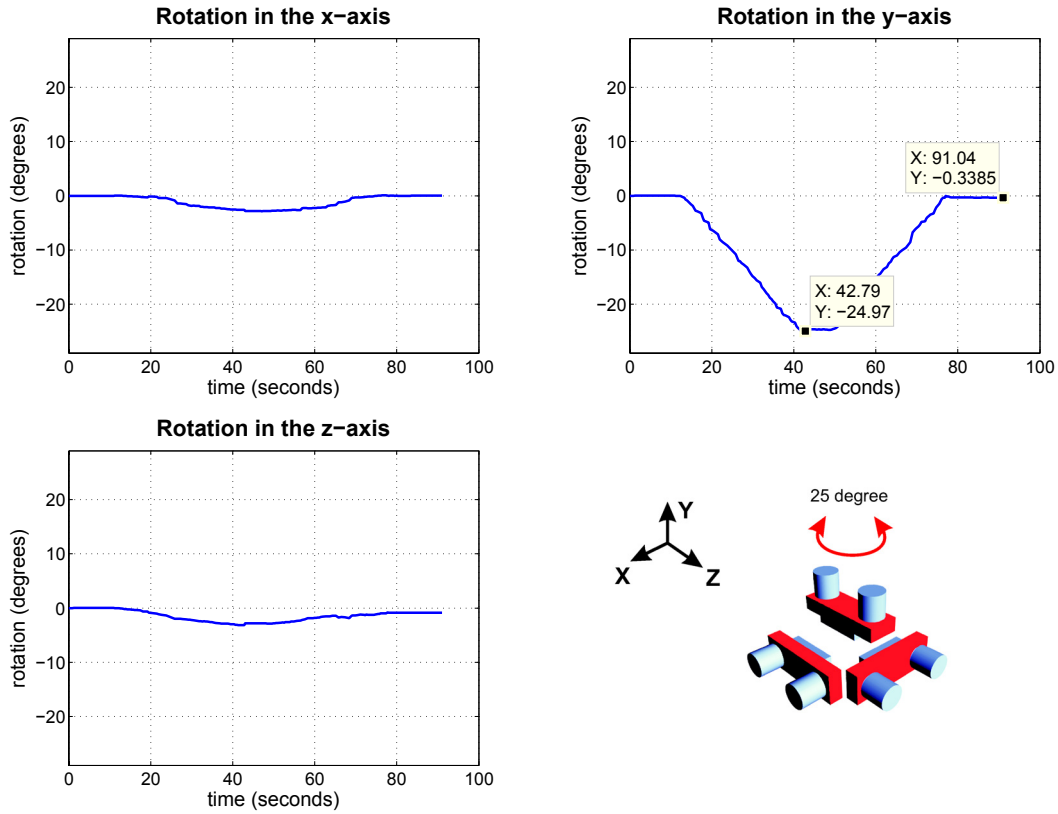


**Figure 6.9: Results of the rotation estimation using 25 degrees rotation around the y-axis. All three cameras are used.**

## Result Analysis

### Test 1 and Test 3

From Figure 6.4 and Figure 6.6, it can be seen that the rotation estimated results using only one camera contains number of errors. Although the direction of the rotation is correctly detected, the estimated angles are wrong. Some rotations around the x- and y-axes are also presented although no actual rotation around these two axes were given. This effect is expected due to the rotation ambiguity existing in the single camera system, as explained in Section 3.5.

### Test 2

Figure 6.5 shows a good rotation estimation result eventhough only one camera is used. This is the case where the rotation occurs around the optical axis of the camera and the effect of the motion ambiguity is low.

### Test 4

By using two cameras the estimated rotation becomes more accurate compared to the single camera system. Figure 6.7 shows a close estimated rotation compared to the actual rotation even though the optical axis of both cameras are perpendicular to the rotation axis. This is the case where information from both cameras are used to compensate the motion ambiguity, as explained in Section 4.4.

### Test 5

Figure 6.8 shows that the estimated rotation using three cameras is close to the actual rotation. The pose of the MCU after returning to the starting point is also close to the value of the start angle, which indicates the consistence of the 3D motion estimation process. Note that there are small rotations detected on the x- and z-axes due to a small misalignment of the rotational axis of the tripod and the internal coordinate of the MCU. This internal coordinate is assigned internally during the calibration of the MCU and it is not easy to relate it to the outside world, i.e. to the rotation and translation axes of the tripod.

### Test 6

Figure 6.9 shows that the estimated rotation using three cameras is close to the actual rotation.

### 6.3.2 Small Translation

The following 3D motion estimation tests are done using some small translations along the y-axis of the MCU reference frame. The MCU is translated back and forth according to the pre-defined distance, and the 3D motion information during the movement is recorded. The tests also include the use of a one camera system, a two camera system and a three camera system in order to compare the different performances. The following tests are included in this section:

**Test 1:** Estimate 50 mm translation along the y-axis using only Y-camera.

**Test 2:** Estimate 50 mm translation along the y-axis using only Z-camera.

**Test 3:** Estimate 50 mm translation along the y-axis using X- and Z-cameras.

**Test 4:** Estimate 50 mm translation along the y-axis using all three cameras.

**Test 5:** Estimate 10 mm translation along the y-axis using all three cameras.

**Test 6:** Estimate 100 mm translation along the y-axis using all three cameras.

### Test 1: Estimate 50 mm translation along the y-axis using only Y-camera.

In this test the MCU is translated 50 mm back and forth along the y-axis. Figure 6.10 shows the results of the 3D motion estimation derived using only information derived from the Y-camera.

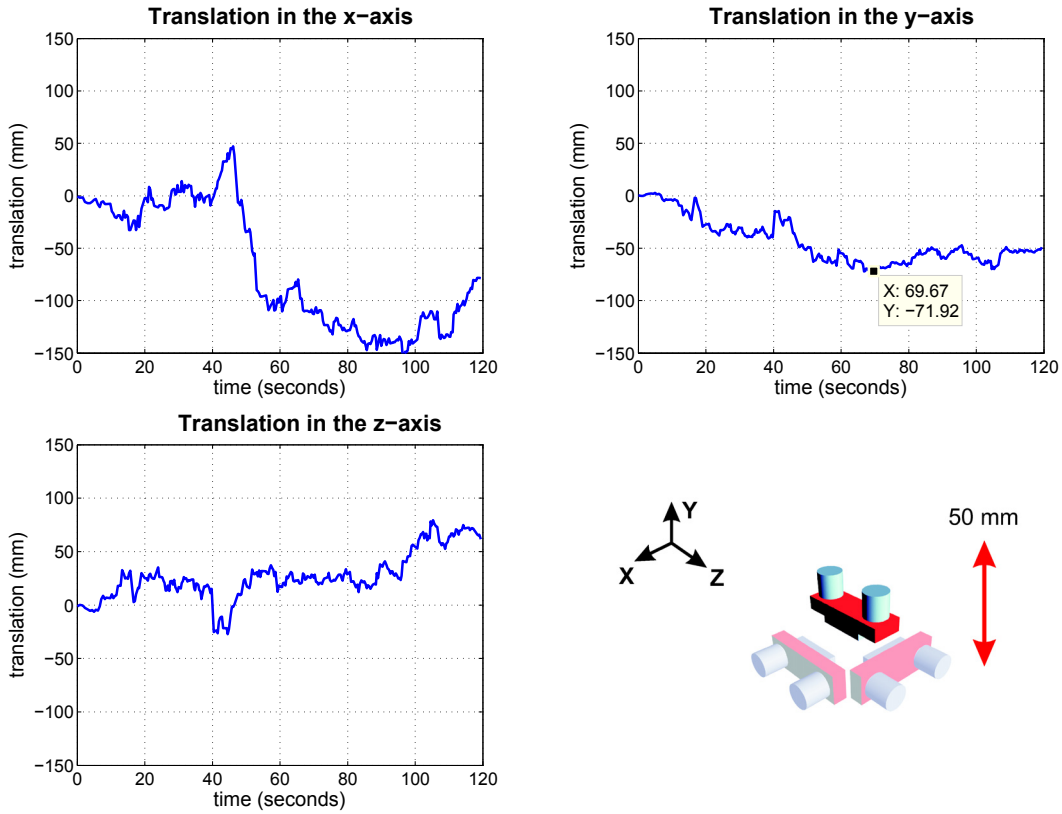


**Figure 6.10: Results of the translation estimation using 50 mm distance along the y-axis. Only Y-camera is used.**

### Test 2: Estimate 50 mm translation along the y-axis using only Z-camera.

In this test the MCU is translated 50 mm back and forth along the y-axis. Figure 6.11 shows the results of the 3D motion estimation derived using only information derived from the Z-camera.

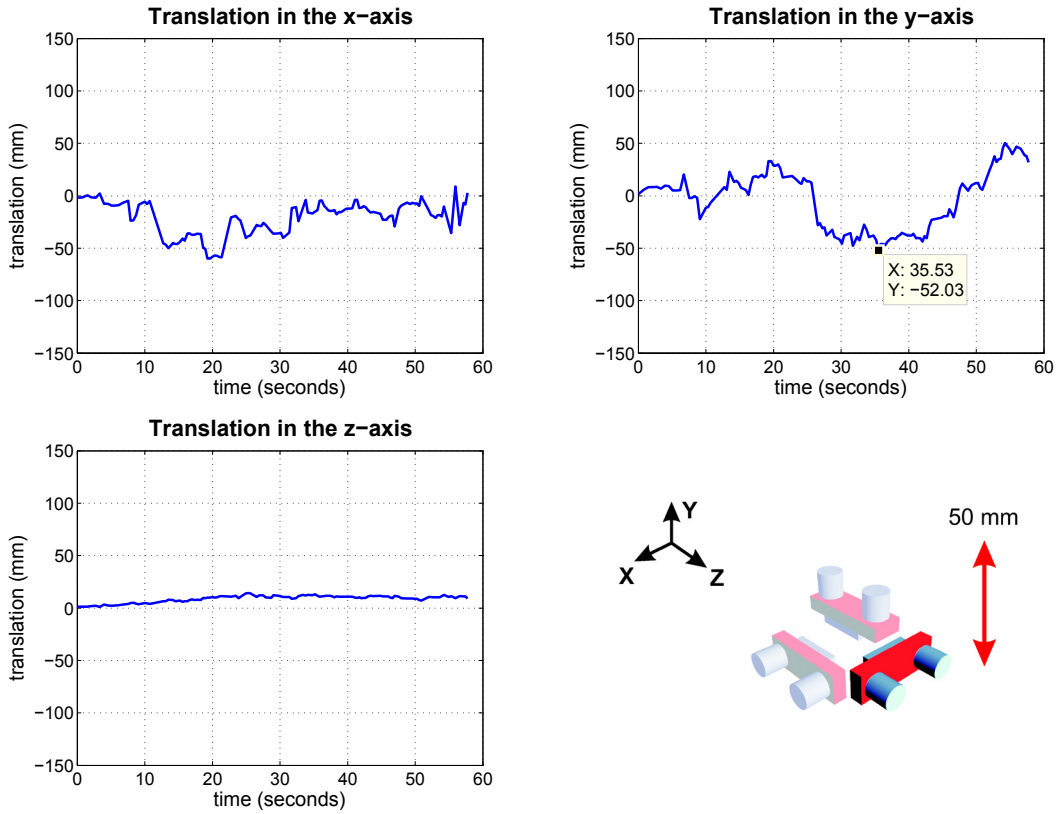


**Figure 6.11: Results of the translation estimation using 50 mm distance along the y-axis. Only Z-camera is used.**

### Test 3: Estimate 50 mm translation along the y-axis using X- and Z-cameras.

In this test the MCU is translated 50 mm back and forth along the y-axis. Figure 6.12 shows the results of the 3D motion estimation derived using information derived from the X- and Z-cameras.

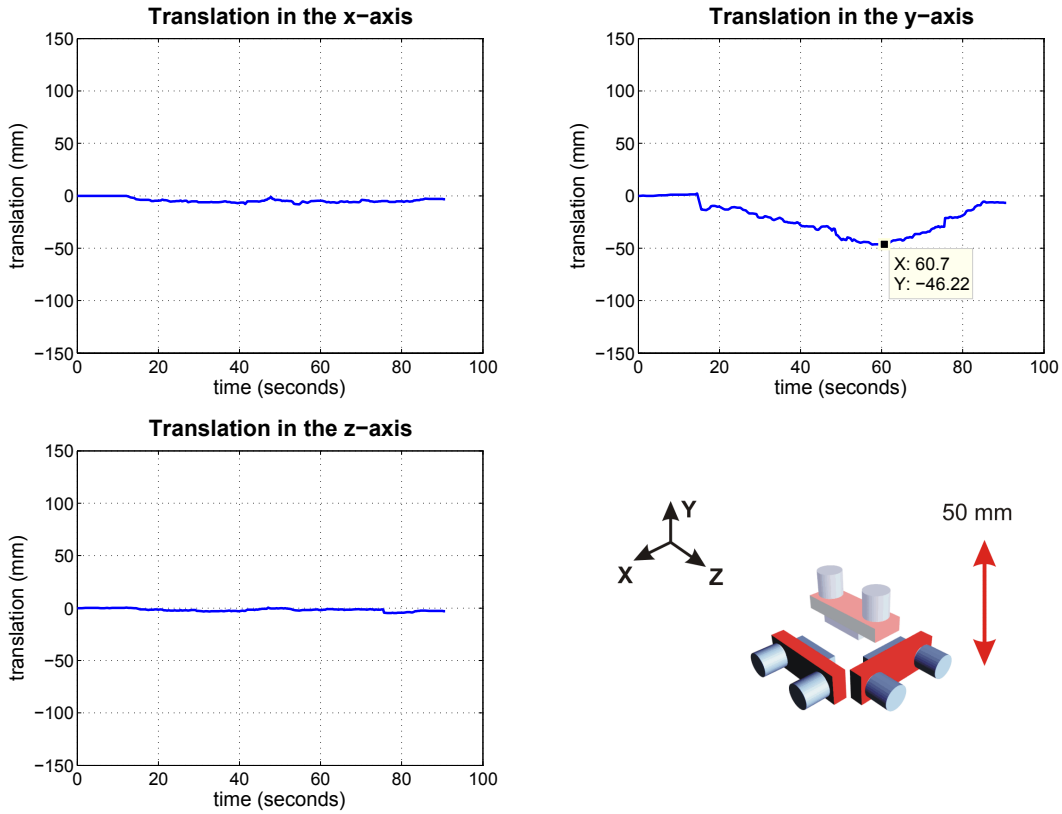


**Figure 6.12: Results of the translation estimation using 50 mm distance along the y-axis. The X- and Z-cameras are used.**

### Test 4: Estimate 50 mm translation along the y-axis using all three cameras.

In this test the MCU is translated 50 mm back and forth along the y-axis. Figure 6.13 shows the results of the 3D motion estimation derived using information derived from all three cameras.

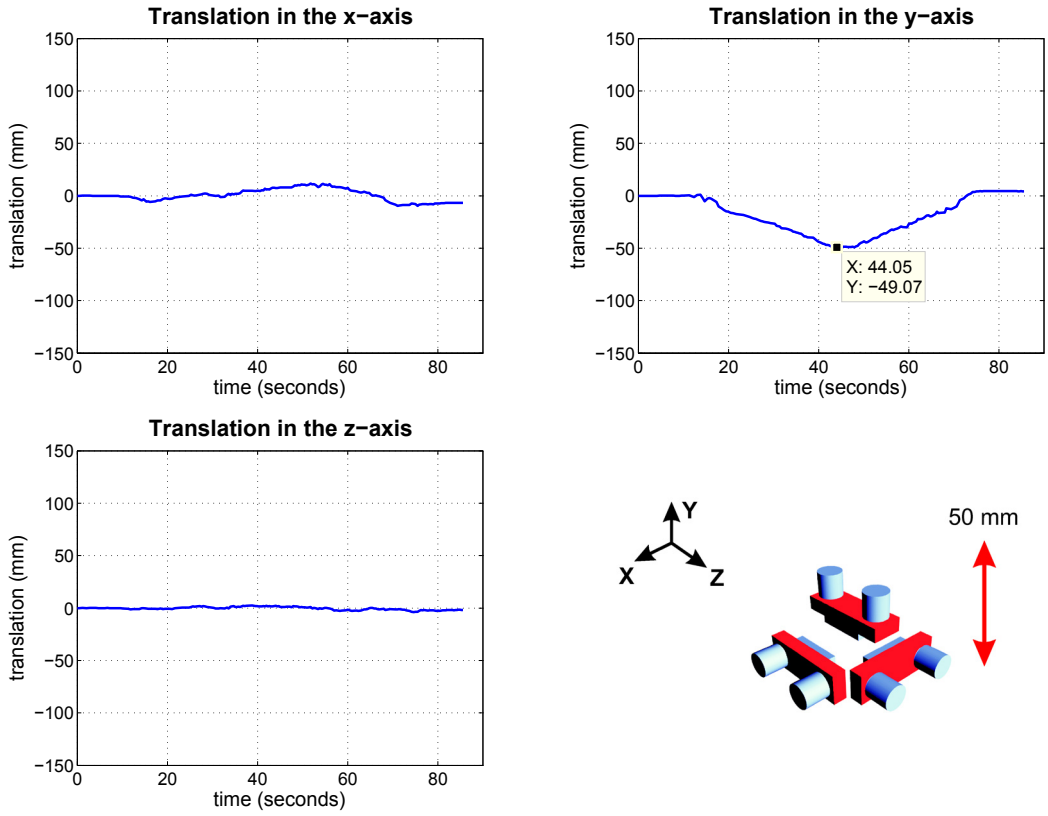


**Figure 6.13: Results of the translation estimation using 50 mm distance along the y-axis. All three cameras are used.**

### Test 5: Estimate 10 mm translation along the y-axis using all three cameras.

This test is designed to observe the ability of the MCU to detect a very small translation. The MCU is moved by 10 mm back and forth along the y-axis at low speed. Figure 6.14 shows the results of the 3D motion estimation derived using information from all three cameras.

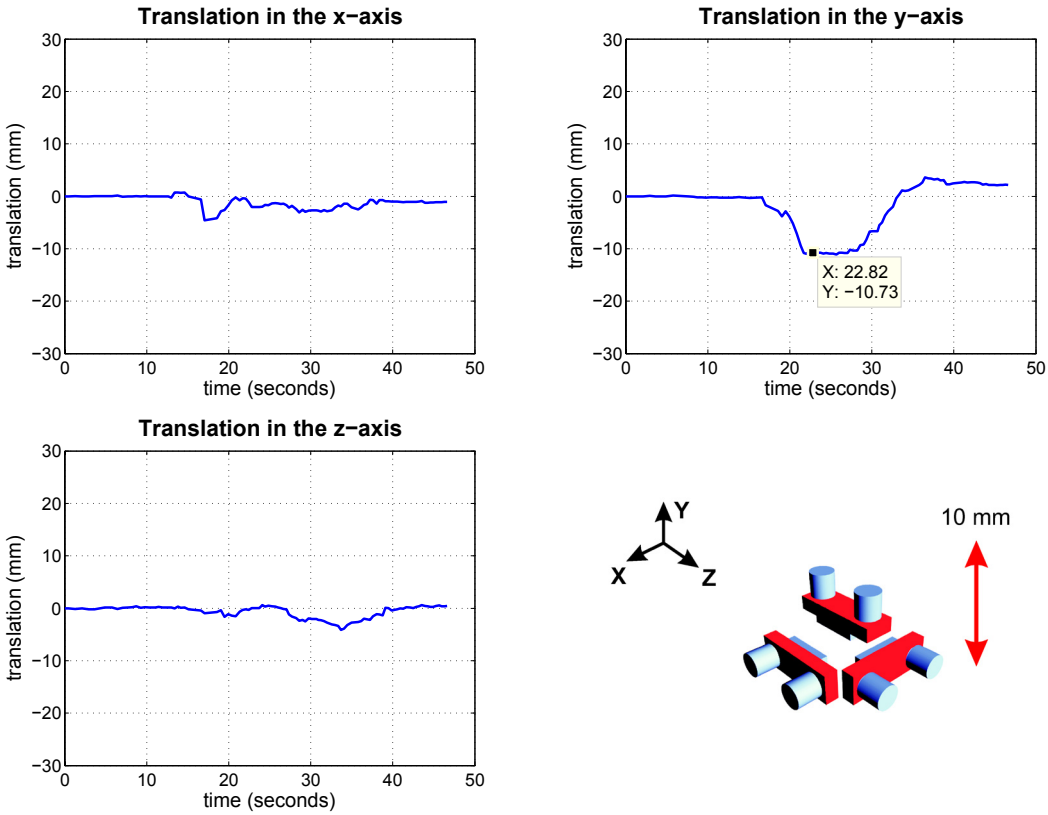


**Figure 6.14: Results of the translation estimation using 10 mm translation along the y-axis. All three cameras are used.**

### Test 6: Estimate 100 mm translation along the y-axis using all three cameras.

In this test the MCU is moved 100 mm back and forth along the y-axis. Figure 6.15 shows the results of the 3D motion estimation derived using information from all three cameras.

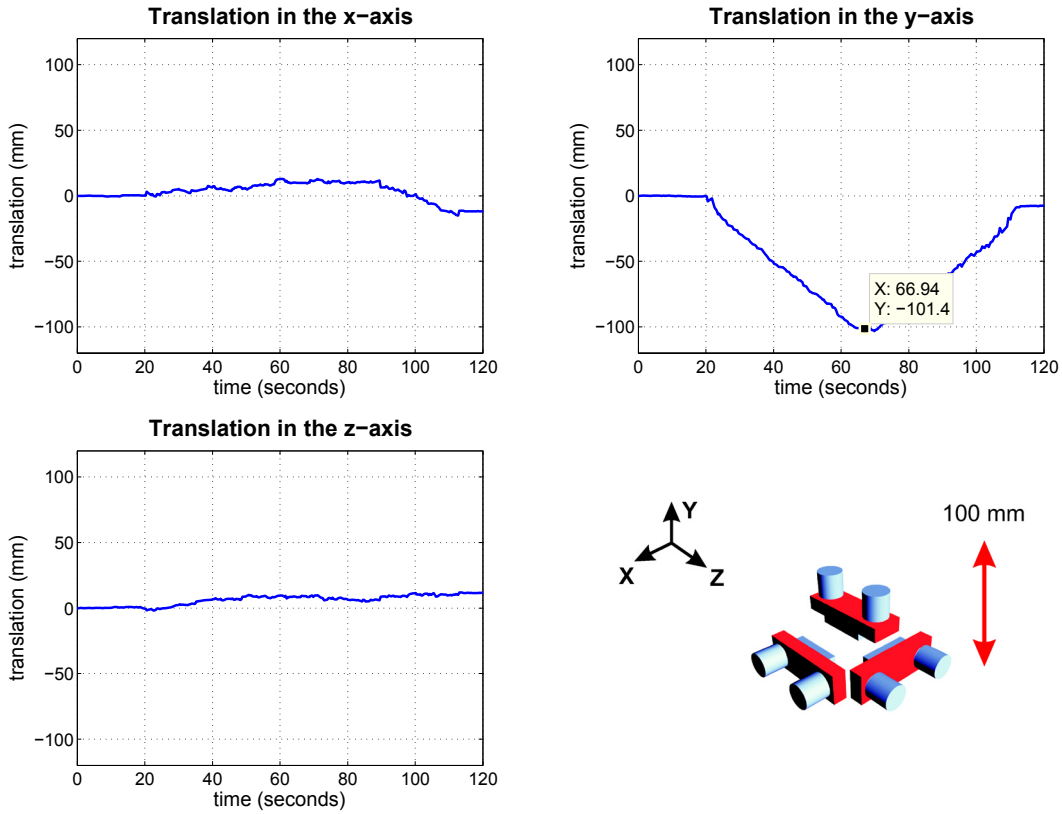


**Figure 6.15: Results of the translation estimation using 100 mm distance along the y-axis. All three cameras are used.**

## Result Analysis

### Test 1 and Test 2

From Figure 6.10 and Figure 6.11 it can be seen that the estimated translations are heavily corrupted. This is expected from the single camera system due to the effect of the motion ambiguity explained in Section 3.5.

### Test 3

By using two cameras, the result of the translation estimation is improved. Figure 6.12 shows a good translation estimation result where the estimated translation of -46.22 mm is close to the actual transformation of -50 mm. Also, unlike single camera cases, there are no false estimations of the translation along the x- and z-axes since the motion ambiguity is compensated using information from both cameras, as explained in Section 4.4.

### Test 4

Figure 6.13 shows the translation estimation result using three cameras. The translation estimation of -49.07 mm is very close to the actual translation of -50 mm and the false translation estimation along the x- and z-axis is smaller compared to the single camera system.

### Test 5

The three camera system shows a promising result for a very small translation estimation. This can be seen in Figure 6.14 where the estimated translation of -10.73 mm is close to the actual translation of -10 mm.

### Test 6

Figure 6.15 shows a good translation estimation result using three cameras. The estimated translation is -101.4 mm which is close to the actual translation of -100 mm.

### 6.3.3 Large Rotation

The following 3D motion estimation tests are done using two large rotations around the x- and y-axes of the MCU reference frame. The MCU is rotated back and forth according to the pre-defined angle and the 3D motion information during the movement is recorded. In addition to the previous tests, the speed information and the detected 3D motion are also presented in some of the tests. The following tests are included in this section:

**Test 1:** Estimate 25 degrees rotation around the x-axis using all three cameras.

**Test 2:** Estimate 45 degrees rotation around the y-axis using all three cameras.

**Test 1: Estimate 25 degrees rotation around the x-axis using all three cameras.**

In this test the MCU is rotated 25 degrees back and forth around the y-axis. Figure 6.16 shows the results of the 3D motion estimation and Figure 6.17 shows the speed information using information derived from all three cameras.

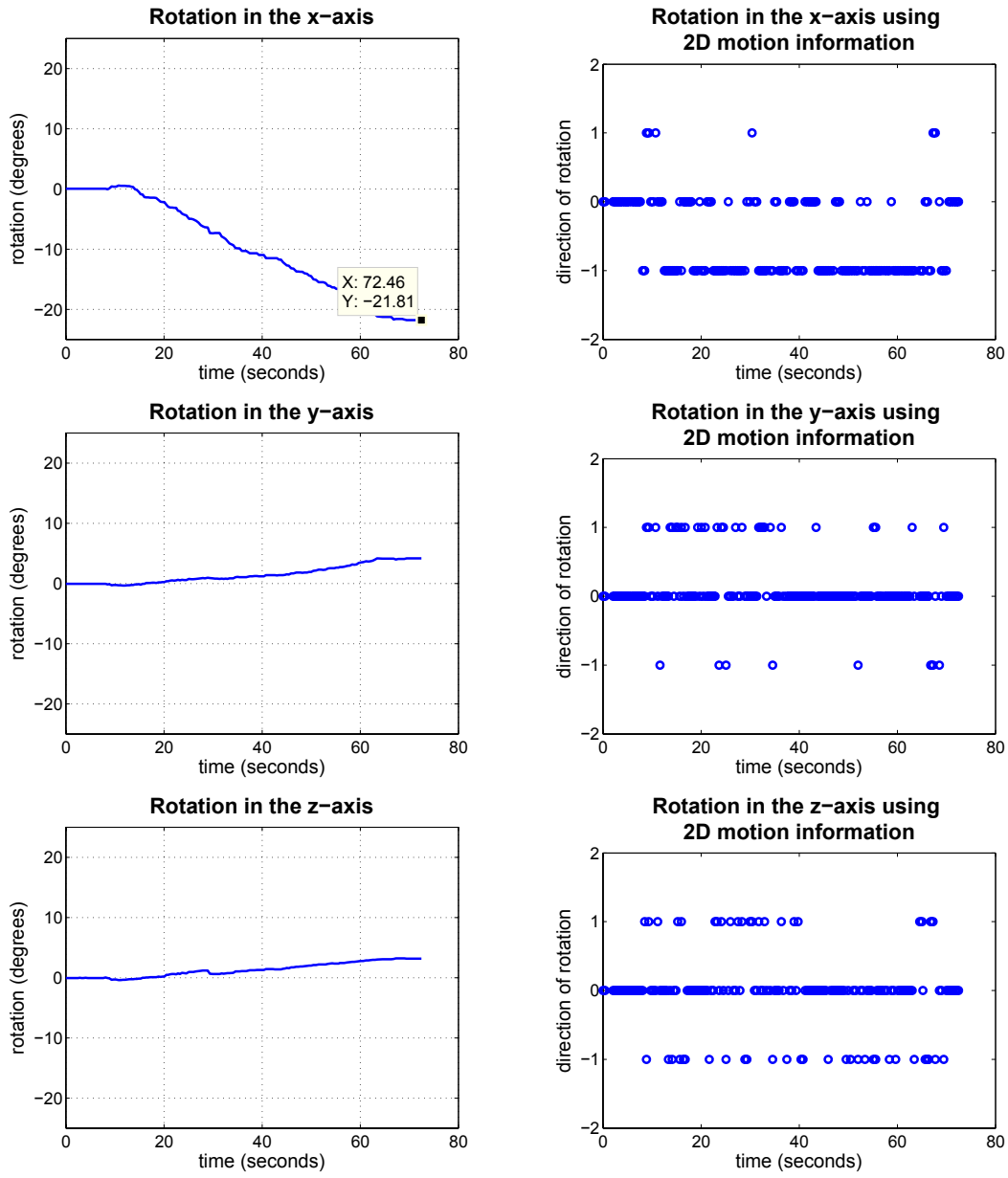


**Figure 6.16: Results of the rotation estimation using 25 degrees rotation around the x-axis. The plots on the left-hand side show the rotation estimation and the plots on the right-hand side show the detected direction of rotation during the test (1=counter-clockwise, -1=clockwise).**

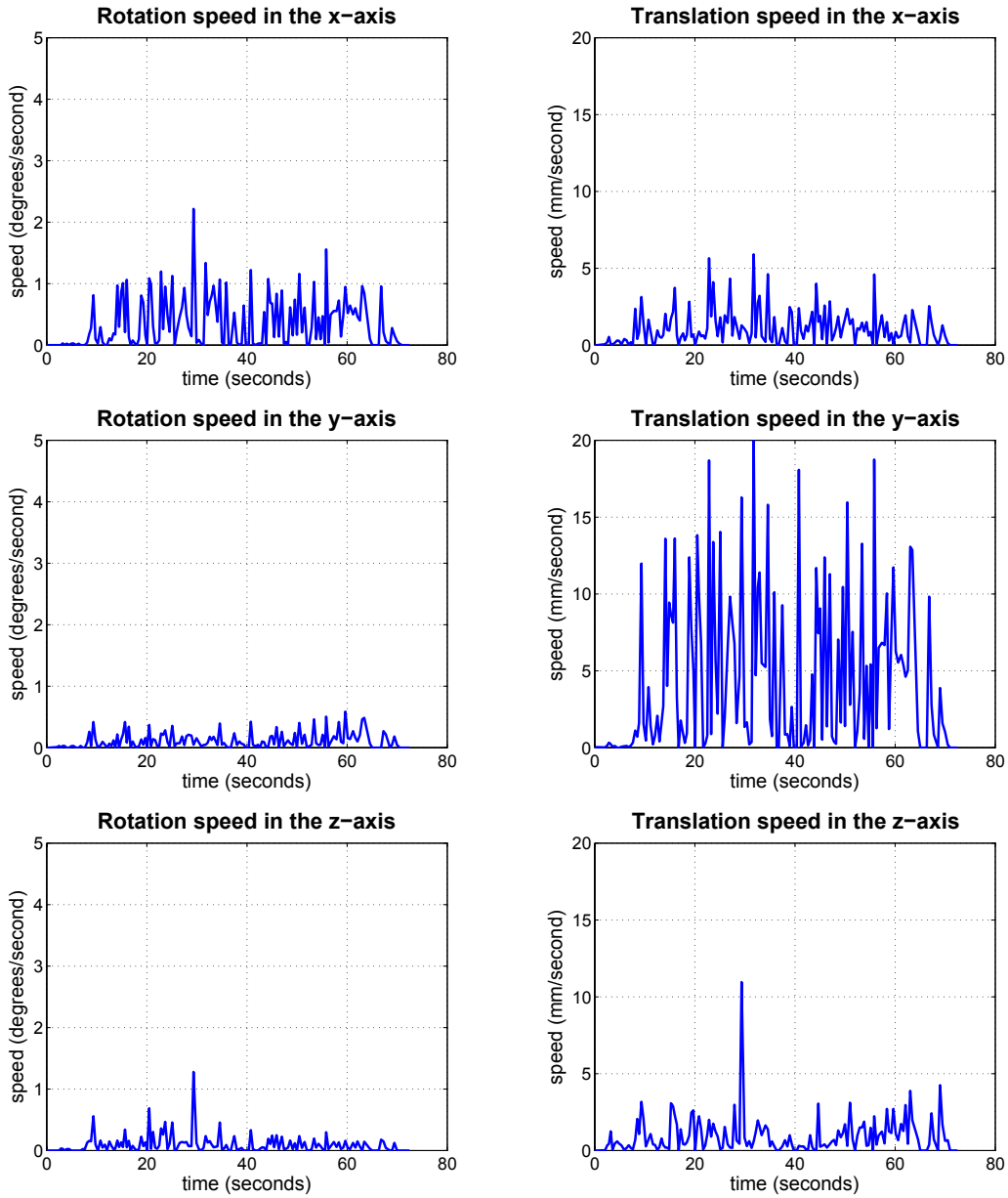


**Figure 6.17: The rotation and translation speed of the MCU during the 25 degrees rotation around the x-axis.**

### Test 2: Estimate 45 degrees rotation around the y-axis using all three cameras.

In this test the MCU is rotated 45 degrees back and forth around the y-axis. Figure 6.18 shows the results of the 3D motion estimation and Figure 6.19 shows the speed information using information derived from all three cameras.

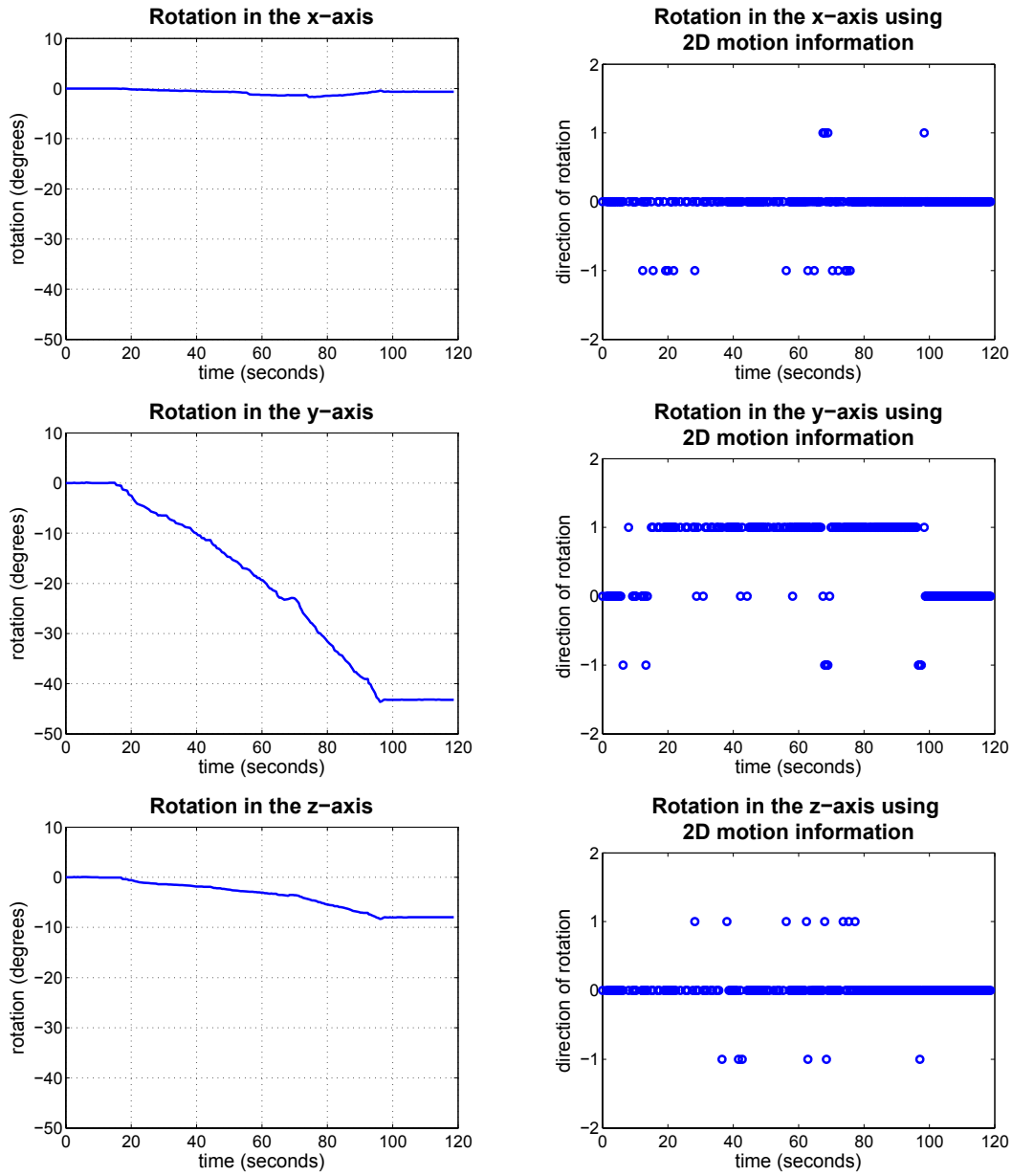


**Figure 6.18: Results of the rotation estimation using 45 degrees rotation around the y-axis. The plots on the left-hand side show the rotation estimation and the plots on the right-hand side show the detected direction of rotation during the test (1=counter-clockwise, -1=clockwise).**

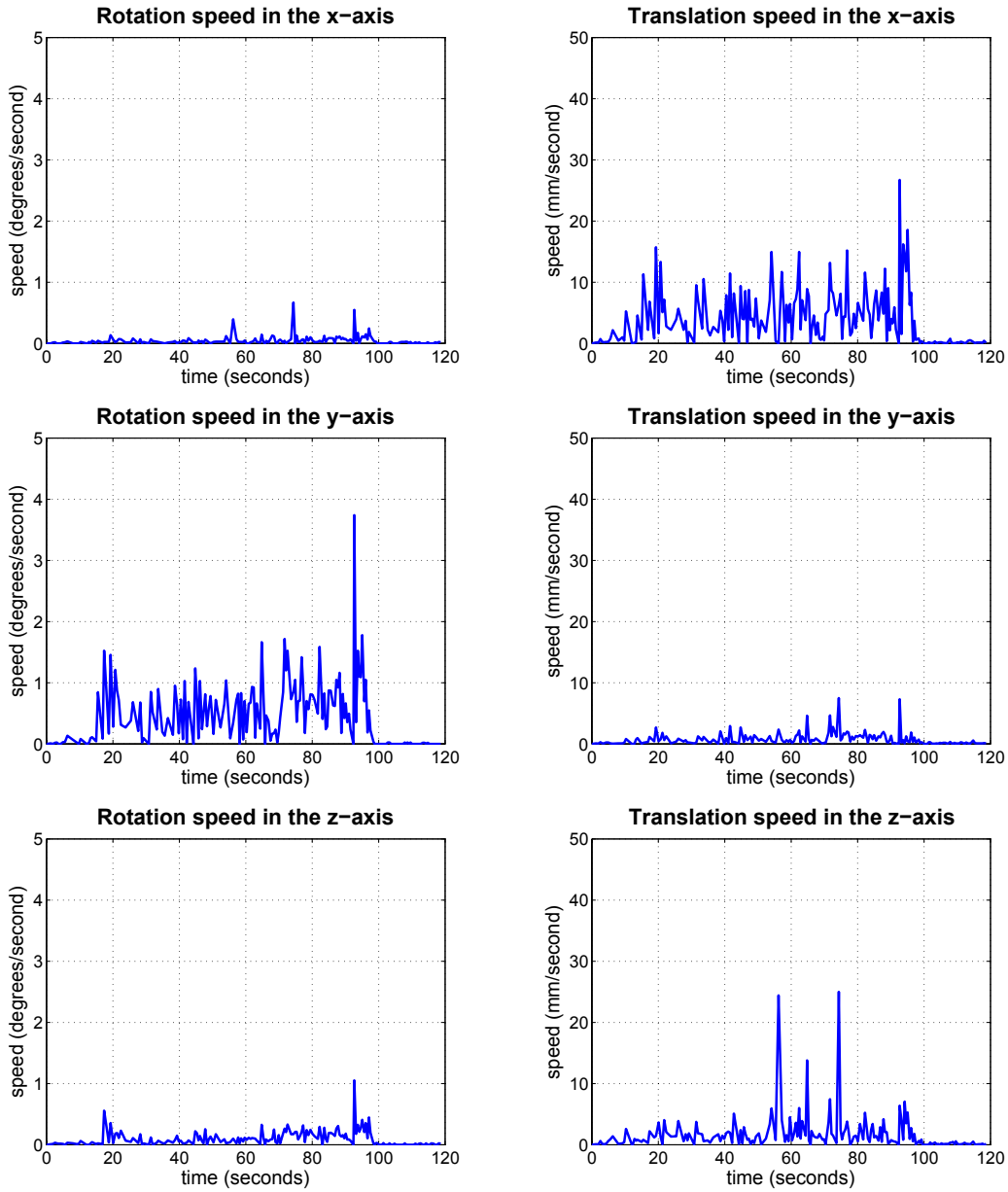


**Figure 6.19: The rotation and translation speed of the MCU during the 45 degrees rotation around the y-axis.**

## Result Analysis

### Test 1

Figure 6.16 shows the good rotation estimation results using three cameras. The estimated target angle is around -21.81 degrees, which is close to the actual value of -25 degrees. In addition, the 3D motion detection results which are derived using the procedures described in Section 4.4 are also presented. Two numerical values present the direction of the rotation: 1 for counter-clockwise and -1 for clockwise rotation around the rotation axis. Figure 6.17 shows the estimated speed derived during the test. This includes the rotation speed and translation speed on all three axes. The speed is calculated using the estimated translation and rotation and the time stamp information.

Note that there are small rotations detected on the y- and z-axes due to a small misalignment of the rotation axis of the tripod and the internal coordinate of the MCU. This internal coordinate is assigned internally during the calibration of the MCU and it is not easy to relate it to the outside world, i.e. to the rotation and translation axes of the tripod.

### Test 2

Figure 6.18 shows the good motion estimation results using three cameras. The estimated target angle is around -43 degrees, which is close to the actual value of -45 degrees. The 3D motion detection results are also presented and they are in agreement with the actual rotation during the test. Figure 6.19 shows the estimated speed including the rotation speed and translation speed of the MCU during the test.

Note that there are small rotations detected on the z-axis due to a small misalignment of the rotation axis of the tripod and the internal coordinate of the MCU. This internal coordinate is assigned internally during the calibration of the MCU and it is not easy to relate it to the outside world, i.e. to the rotation and translation axes of the tripod.

### 6.3.4 Large Translation

The following 3D motion estimation tests are done using two large translations along the x- and z-axes of the MCU reference frame. The MCU is translated back and forth according to the pre-defined distance and the 3D motion information during the movement is recorded. The following tests are included in this section:

**Test 1:** Estimate 500 mm translation along the x-axis using all three cameras.

**Test 2:** Estimate 1000 mm translation along the z-axis using all three cameras.

**Test 1: Estimate 500 mm translation along the x-axis using all three cameras.**

In this test the MCU is translated by 500 mm back and forth along the x-axis. Figure 6.20 shows the results of the 3D motion estimation and Figure 6.21 shows the speed information using information derived from all three cameras. Figure 6.22 shows the screen captured image obtained during the test.

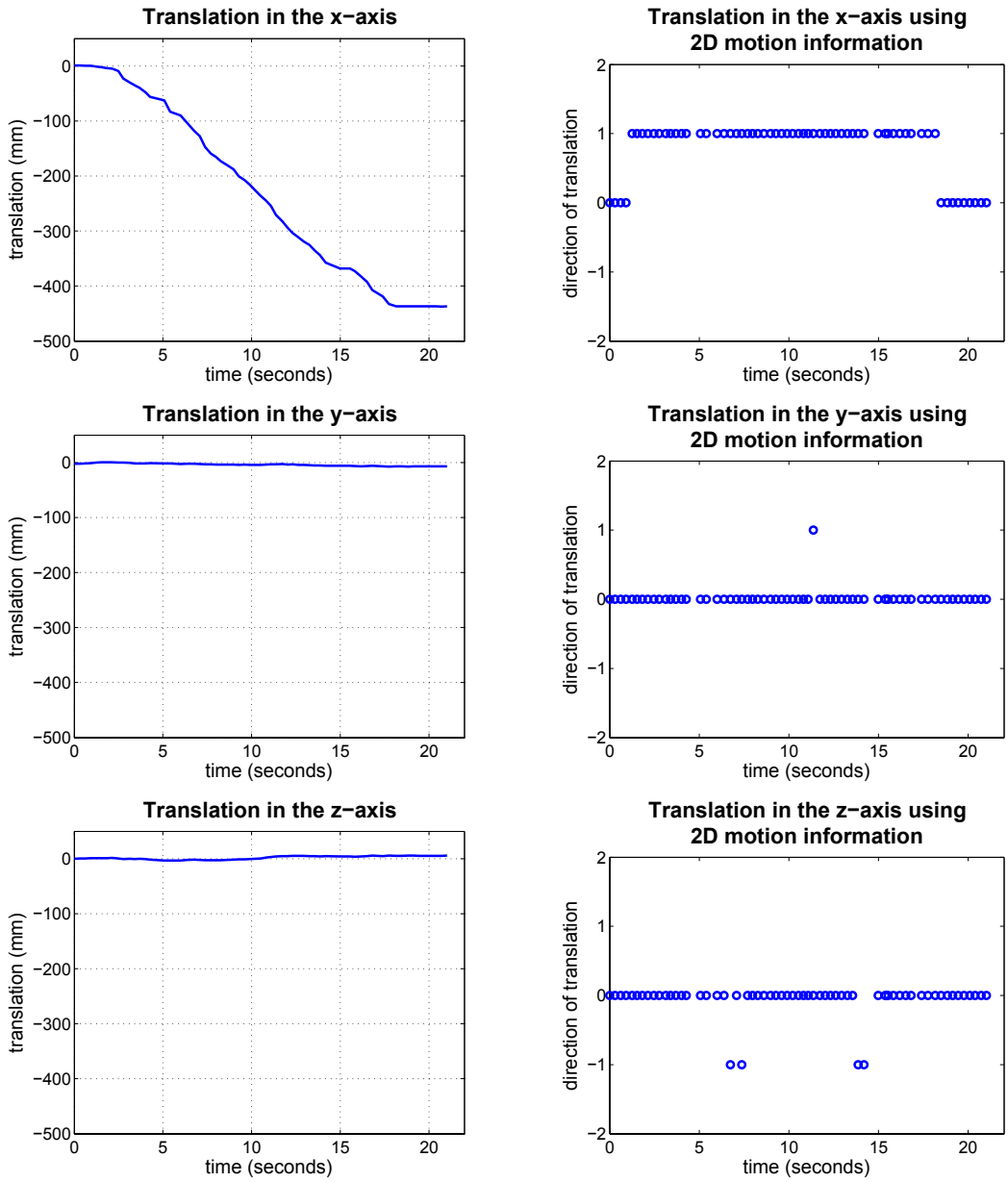


**Figure 6.20: Results of the translation estimation using 500 mm distance along the x-axis. The plots on the left-hand side show the translation estimation and the plots on the right-hand side show the detected direction of translation during the test (1=positive direction, -1=negative direction).**

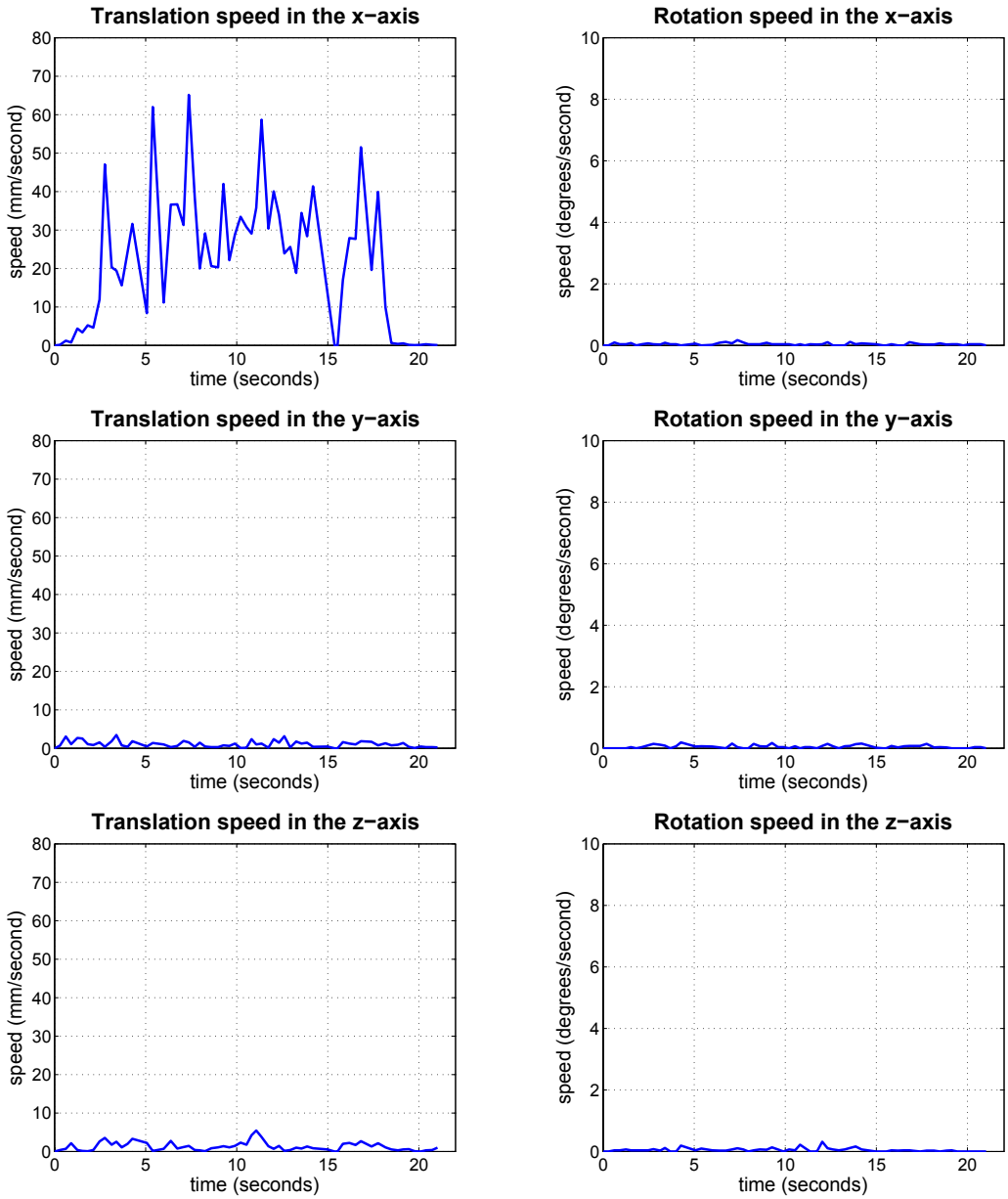


**Figure 6.21: The rotation and translation speed of the MCU during the 500 mm movement along the x-axis.**

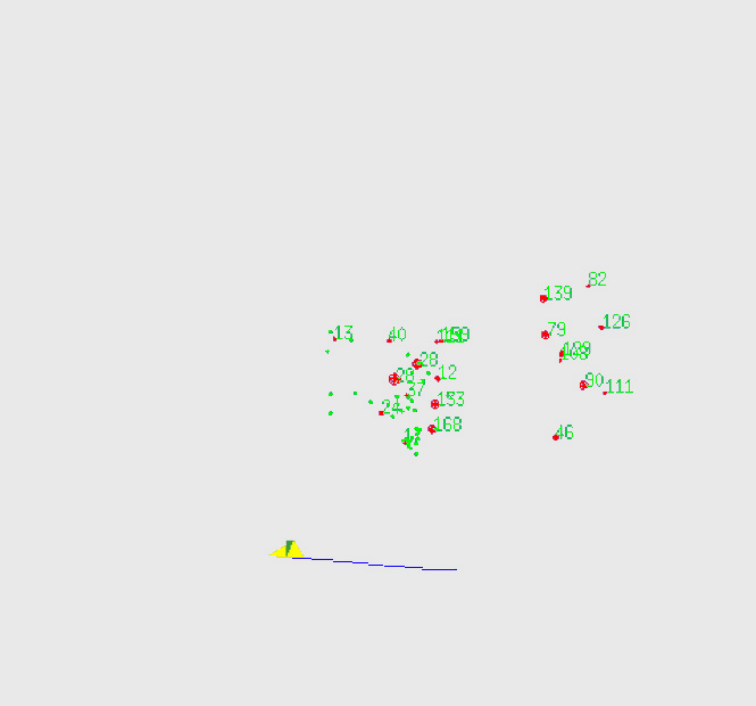


**Figure 6.22: The estimated 3D trajectory captured by the MCU during the 500 mm movement along the x-axis.**

### Test 2: Estimate 1000 mm translation along the z-axis using all three cameras.

In this test the MCU is translated 1000 mm back and forth along the z-axis. Figure 6.23 shows the results of the 3D motion estimation using information derived from all three cameras. Figure 6.24 shows the screen captured image obtained during the test.

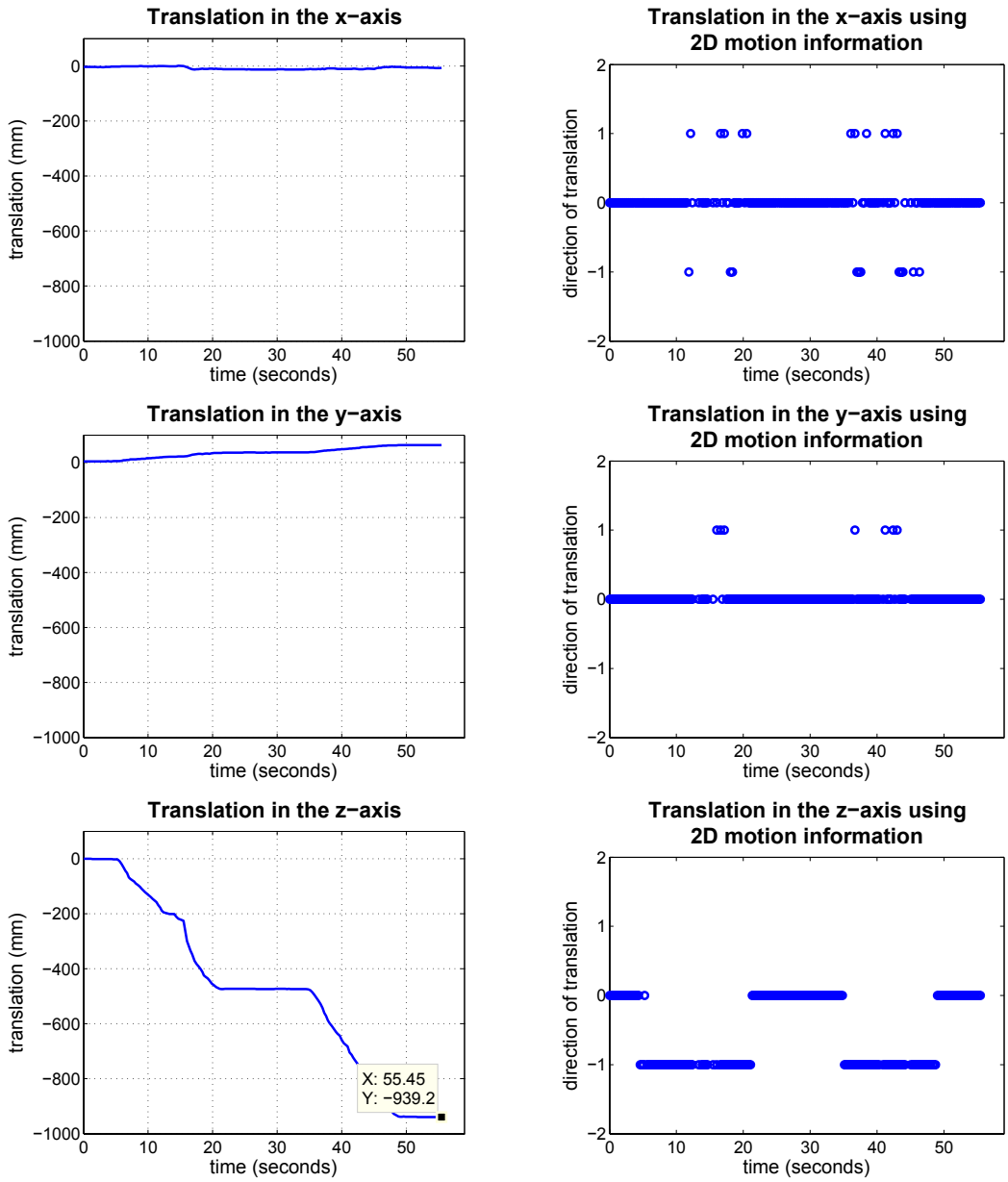


**Figure 6.23: Results of the translation estimation using 1000 mm distance along the z-axis. The plots on the left-hand side show the translation estimation and the plots on the right-hand side show the detected direction of translation during the test (1=positive direction, -1=negative direction).**

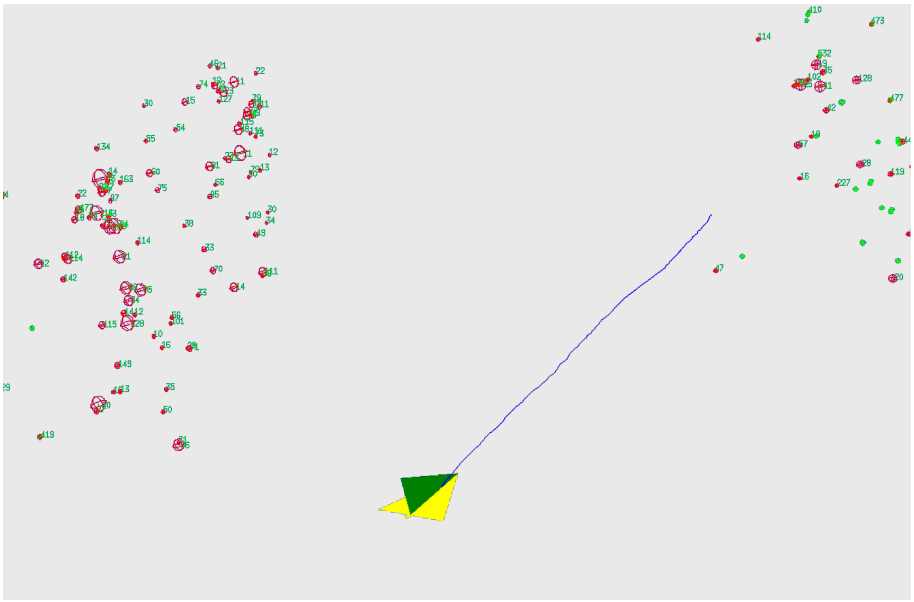

**Figure 6.24: The estimated 3D trajectory captured by the MCU during the 1000 mm movement along the z-axis.**

## Result Analysis

### Test 1

From Figure 6.20 the translation estimation result of -441 mm is acceptable compared to the actual translation of -500 mm. The detected 3D motion information derived according the procedure described in Section 4.4 is also presented. The rotation and translation speeds shown in Figure 6.21 are obtained using the estimated motion information and the time stamp information. Figure 6.22 shows the screen-captured image of the map building software during the test. It shows the feature points being used for the 3D motion estimation task as well as the travelled trajectory, which closely represents the actual trajectory given to the MCU hardware.

### Test 2

From Figure 6.23 the translation estimation result of -939.2 mm is acceptable compared to the actual translation of -1000 mm. It can also be seen that the movement of the MCU stopped at around the 20th second and is resumed at the 35th second, which is correct since there was a 15 second interval with no movement during the test. Figure 6.24 shows the screen-captured image of the map building software during the test. It shows the feature points being used for the 3D motion estimation task as well as the travelled trajectory, which closely represents the actual trajectory given to the MCU hardware.

### 6.3.5 The ∂-shape Trajectory

In this test the MCU was moved slowly along the ∂-shape trajectory and the estimated 3D trajectory was captured. The ∂-shape trajectory consists of several straight line sections as illustrated in Figure 6.25. The results of the 3D motion estimation and the speed are shown in Figure 6.26 and Figure 6.27 respectively. A 3D plot of the estimated 3D trajectory is shown in Figure 6.28. Some screen capture images of the estimated 3D trajectory and the feature point map are shown in Figure 6.29.

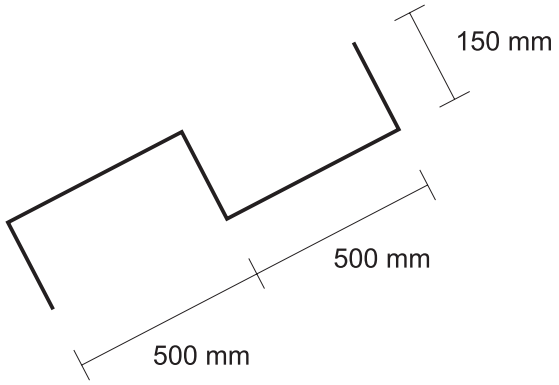


**Figure 6.25: The ∂-shape trajectory.**

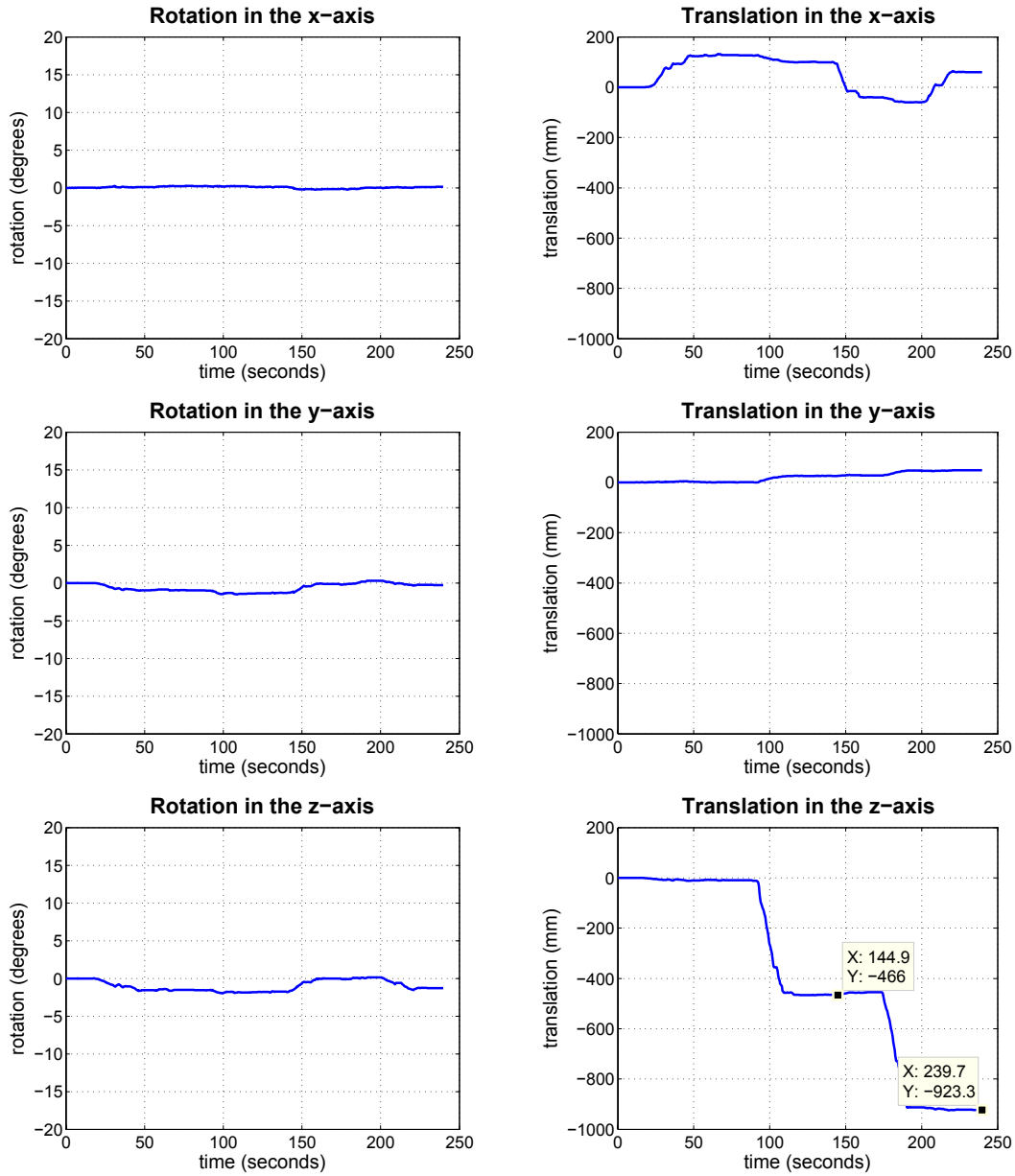


**Figure 6.26: Results of the motion estimation of the MCU travelled along the ⊐-shape trajectory.**

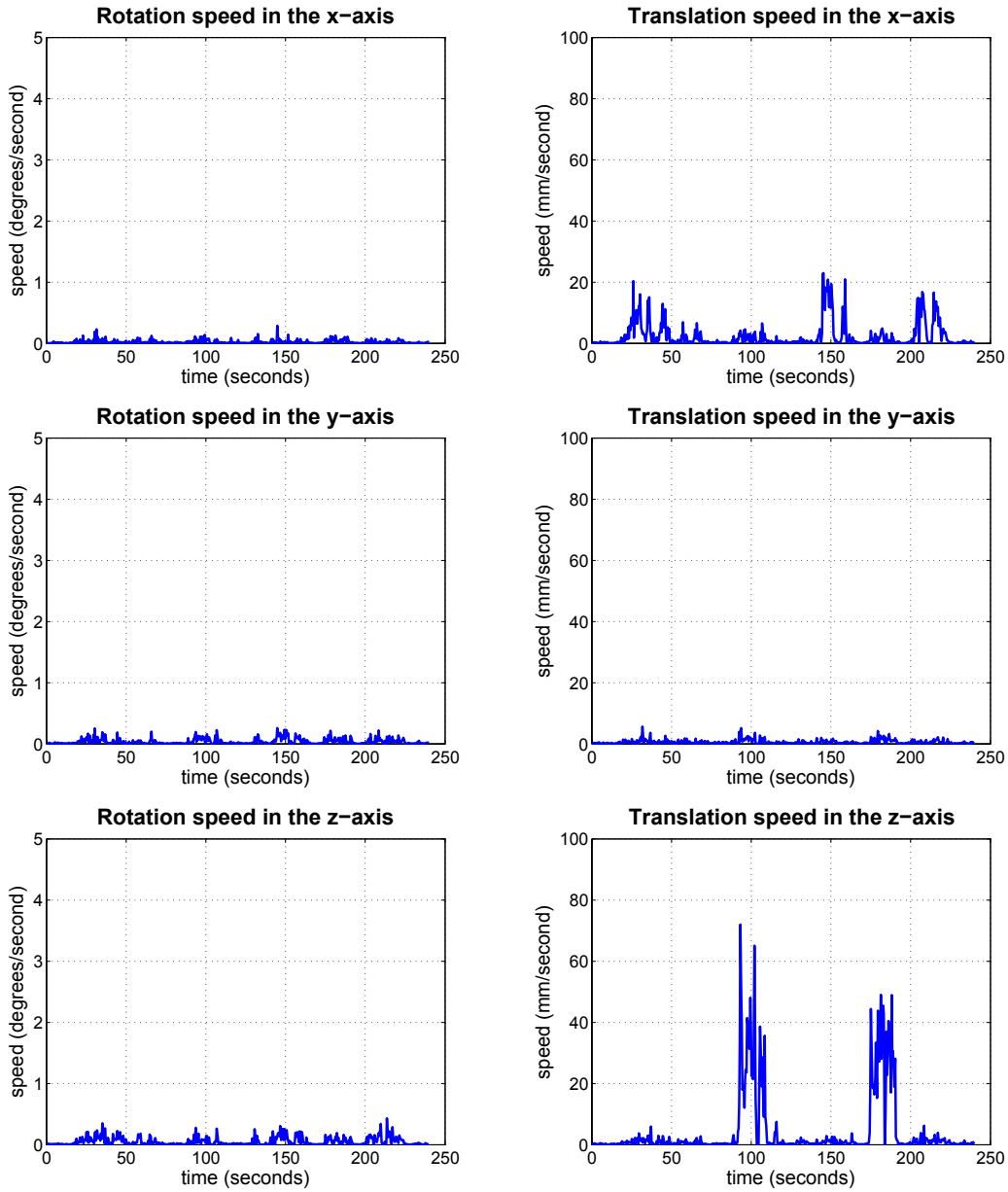


**Figure 6.27: The estimated rotation and translation speed of the MCU travelled along the 2-shape trajectory.**

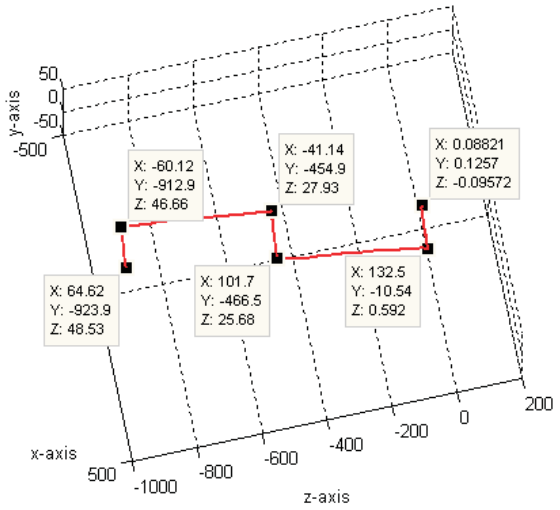


**Figure 6.28: A 3D plot of the estimated ↄ-shape trajectory derived from the MCU.**

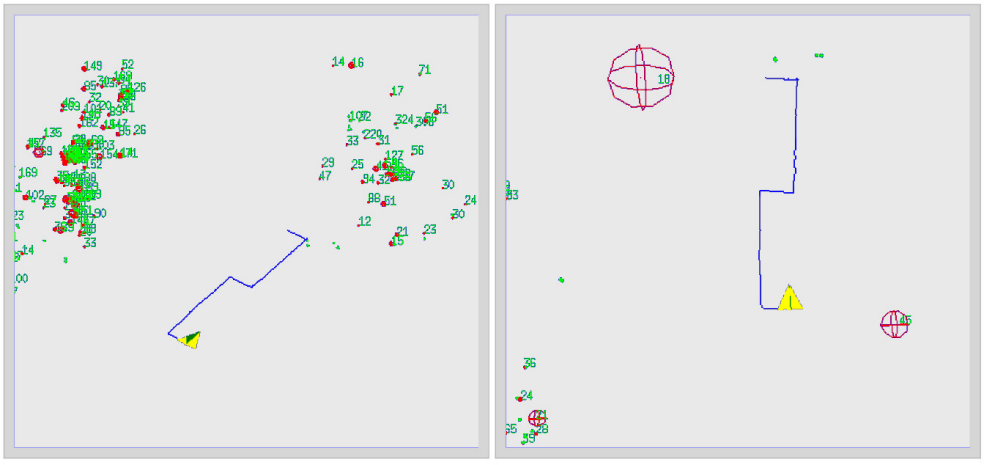

**Figure 6.29: Some screen capture images of the estimated 3D trajectory acquired from the movement along the ↄ-shape trajectory.**

## Result Analysis

Figure 6.26 and Figure 6.27 show the estimated translation and rotation and the estimated speed of the MCU captured along the test trajectory. The estimated trajectory closely resembled the actual test trajectory, which can be confirmed by checking the geometric dimension of the captured trajectory shown in Figure 6.28. In addition, the screen captured images of the map building software during the test are shown in Figure 6.29.

### 6.3.6 Additional Trajectories

Some additional trajectories were used for the 3D motion estimation experiment. The screen capture images of the map building software are presented in order to show the 3D feature point map that is incrementally built during the test. The MCU hardware is shown in the visualized 3D map using a triangle-shape polygon, the estimated 3D trajectory of the MCU is shown using a solid yellow line, the feature points are shown using red circles and the green numbers indicate how often each individual feature point has been seen by the MCU since the beginning of the test. The following tests are included in this section:

**Test 1:** Rotation 90 degrees around the y-axis using all three cameras.

**Test 2:** Rotation 180 degrees around the y-axis using all three cameras.

**Test 3:** Movement along a trapezoidal trajectory using all three cameras.

## Test 1: A 90 degrees rotation around the y-axis

In this test the MCU is rotated 90 degrees around the y-axis. Figure 6.30 shows the 3D feature points and the estimated 3D trajectory of the MCU.

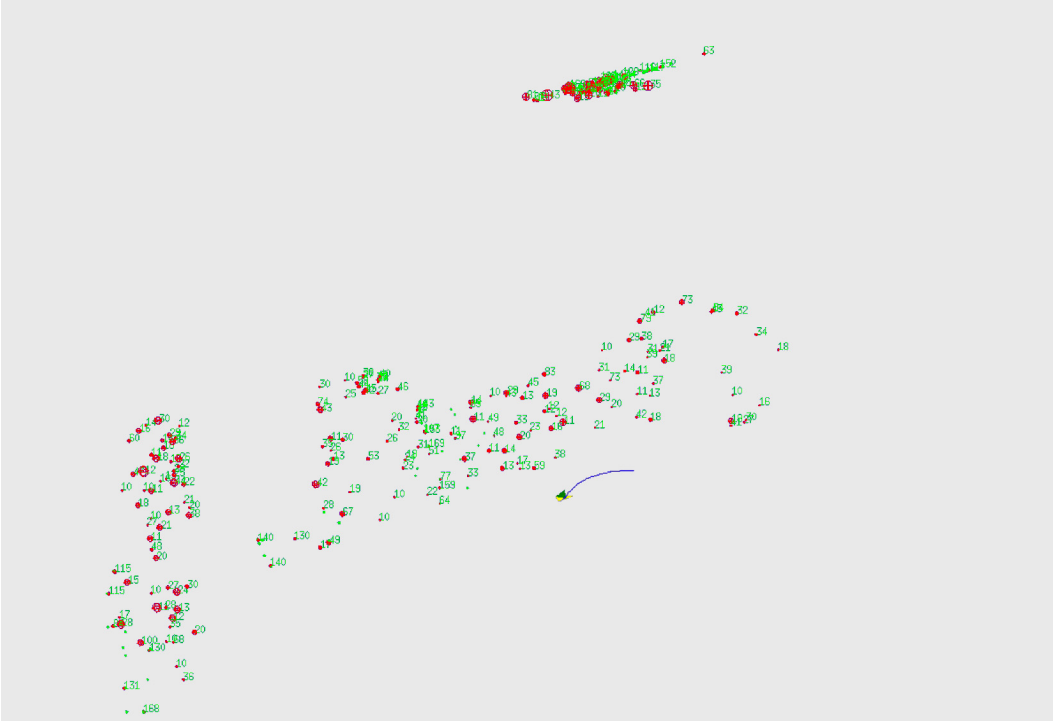

**Figure 6.30: A screen capture image that shows the 3D feature points and the estimated 3D trajectory of the MCU during a 90 degree rotation around the y-axis.**

## Test 2: A 180 degrees rotation around the y-axis

In this test the MCU is rotated 180 degrees around the y-axis. Figure 6.31 shows the 3D feature points and the estimated 3D trajectory of the MCU.

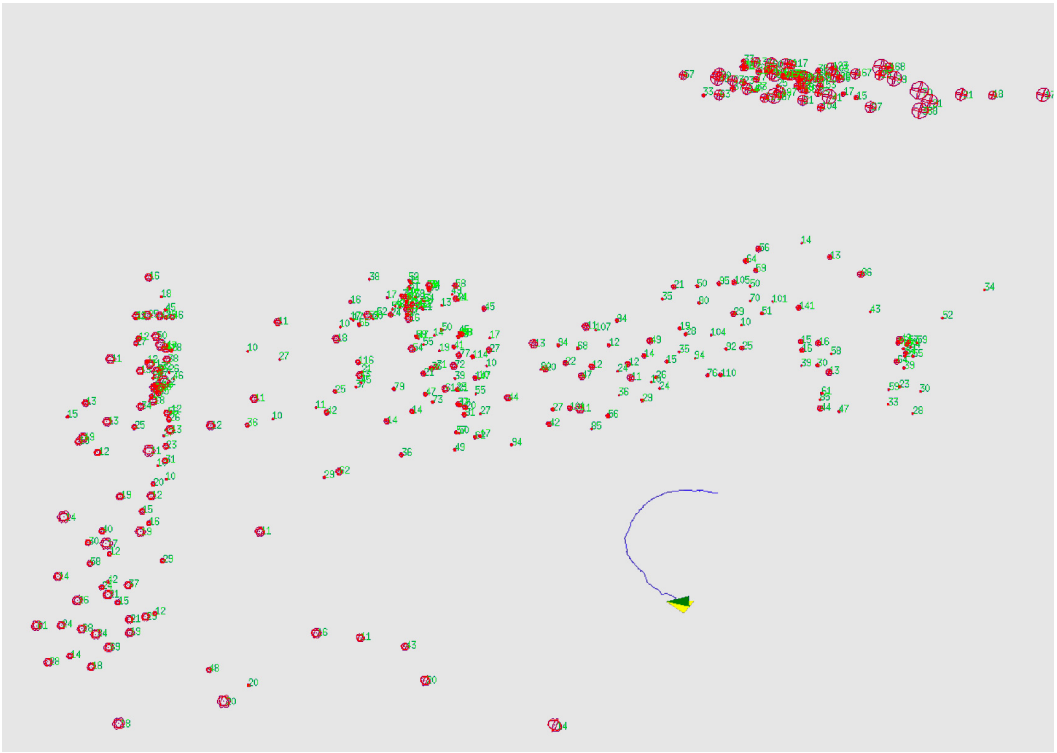

**Figure 6.31: A screen capture image that shows the 3D feature points and the estimated 3D trajectory of the MCU during a 180 degree rotation around the y-axis.**

### Test 3: Movement along a trapezoidal trajectory using all three cameras.

In this test the MCU is moved along a trapezoidal trajectory. Figure 6.32 shows the 3D feature points and the estimated 3D trajectory of the MCU.

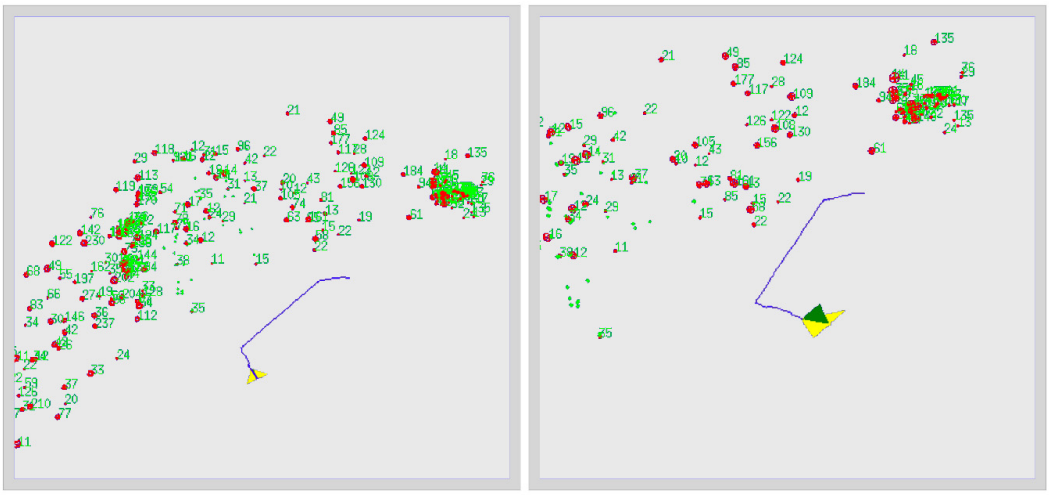

**Figure 6.32: A screen capture image that shows the 3D feature points and the estimated 3D trajectory of the MCU travelled along a trapezoidal trajectory.**

## Result Analysis

Figure 6.30, Figure 6.31 and Figure 6.32 show the estimated trajectories of the MCU captured from the movement along the following trajectories: 90 degree rotation around y-axis, 180 degree rotation around the y-axis and the movement along a trapezoidal trajectory. The estimated trajectories are closely resembled the given test trajectories. In addition, the screen captured images also show the 3D features and the 3D trajectories that are being maintained by the FastSLAM algorithm during the test in real-time.

## 6.4 Experiment II: 3D Photorealistic Map Building

In this experiment several 3D photorealistic maps were captured from the test environment shown in Figure 6.1. Most of the maps presented in this section were built using the 3D images acquired during the tests within Experiment I. Below is a list of the 3D photorealistic maps presented in this section.

### 6.4.1 A 3D photorealistic map from the ↄ-shape trajectory.

### 6.4.2 A 3D photorealistic map from the 90 degrees rotation trajectory.

### 6.4.3 A 3D photorealistic map from the trapezoidal trajectory.

### 6.4.1 A 3D photorealistic map from the ∂-shape trajectory.

In this test the MCU uses the 3D images and 3D motion estimation obtained during the ∂-shape trajectory test in Section 6.3.5 to construct a 3D photorealistic map of the test environment. Figure 6.33 and Figure 6.34 show the screen captured images of the resultant map.

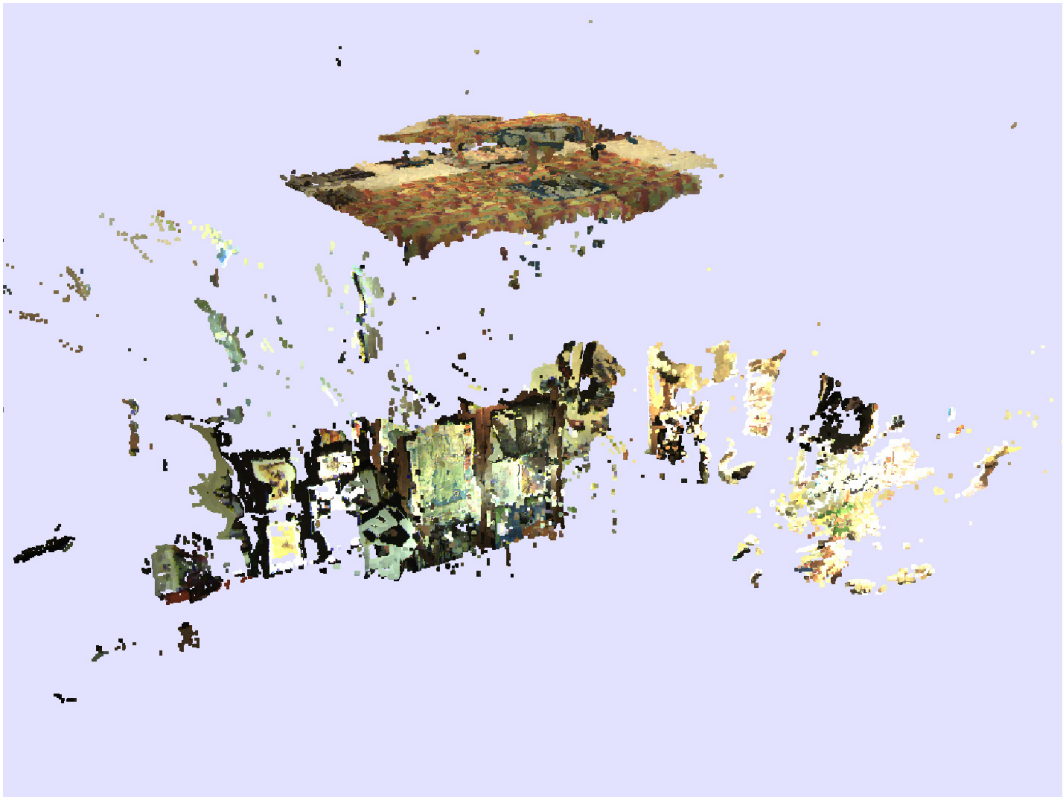

**Figure 6.33: A 3D photorealistic map of the environment obtained during the movement along the ∂-shape trajectory.**

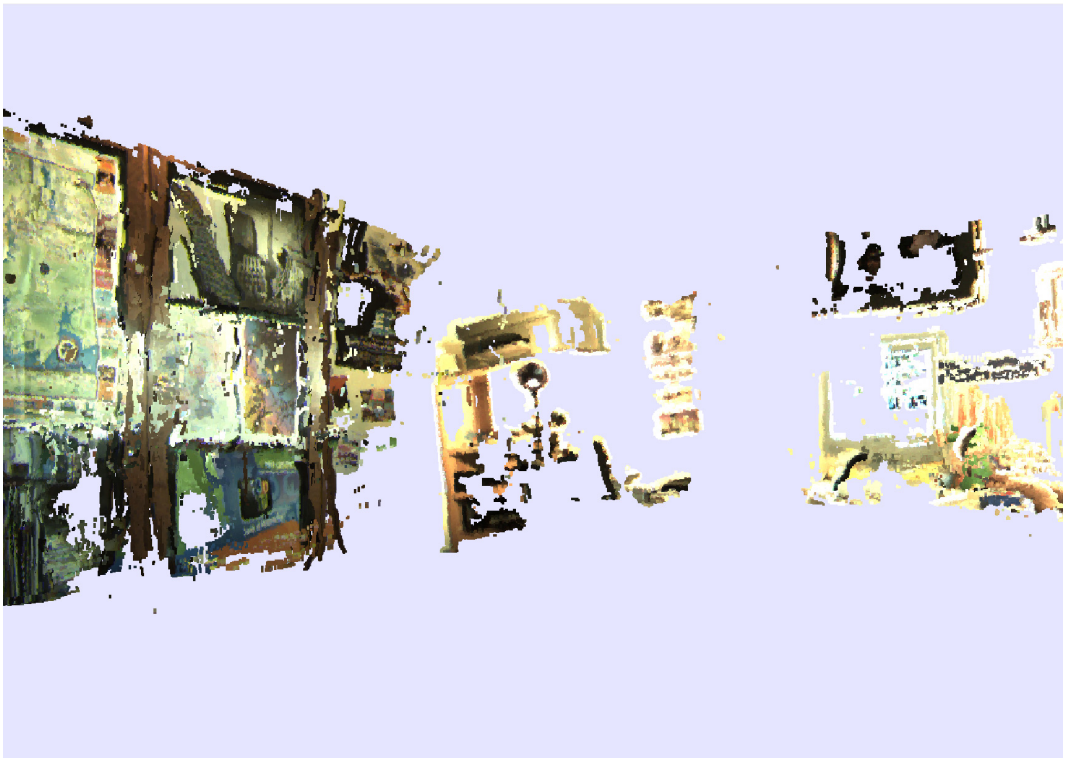

**Figure 6.34: Another view of the 3D photorealistic map of the environment obtained during the movement along the $\mathcal{C}^2$-shape trajectory.**

### 6.4.2 A 3D photorealistic map from the 90 degrees rotation.

In this test the MCU uses the 3D images and 3D motion estimation obtained during the 90 degree rotation test in Section 6.3.6 to construct a 3D photorealistic map of the test environment. Figure 6.35 shows the screen captured image of the resultant map.

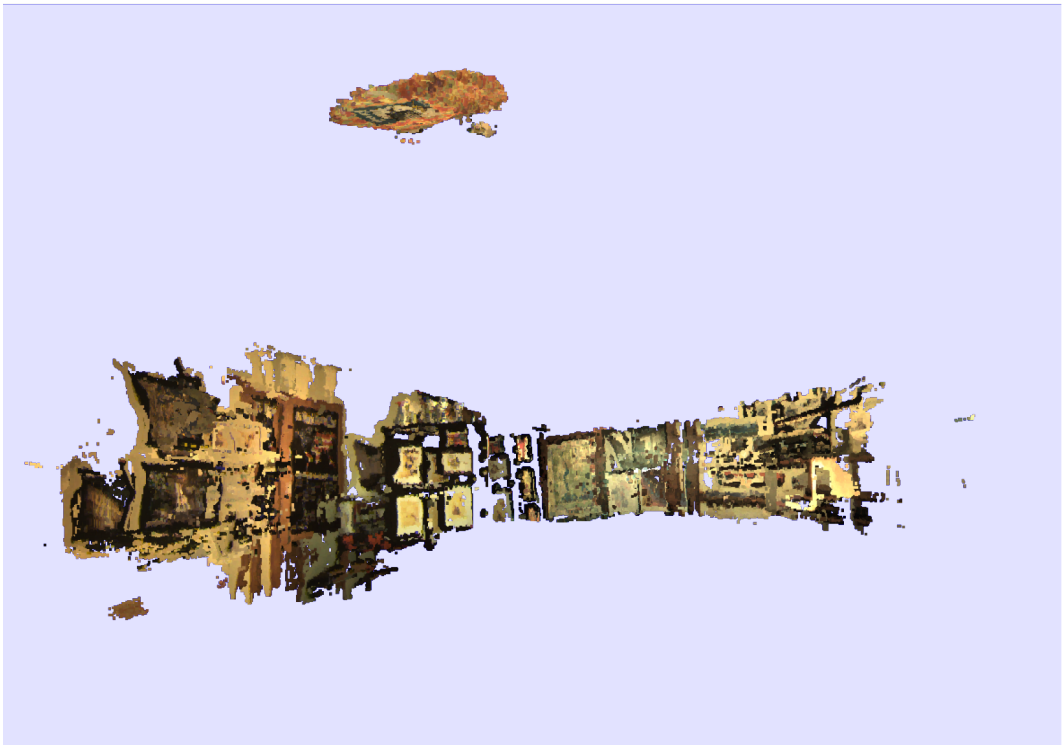

**Figure 6.35: A 3D photorealistic map of the environment obtained during a 90 degrees rotation .**

### 6.4.3 A 3D photorealistic map from the trapezoidal trajectory.

In this test the MCU uses the 3D images and 3D motion estimation obtained during the trapezoidal trajectory test in Section 6.3.6 to construct a 3D photorealistic map of the test environment. Figure 6.36 and Figure 6.37 show the screen captured images of the resultant map.

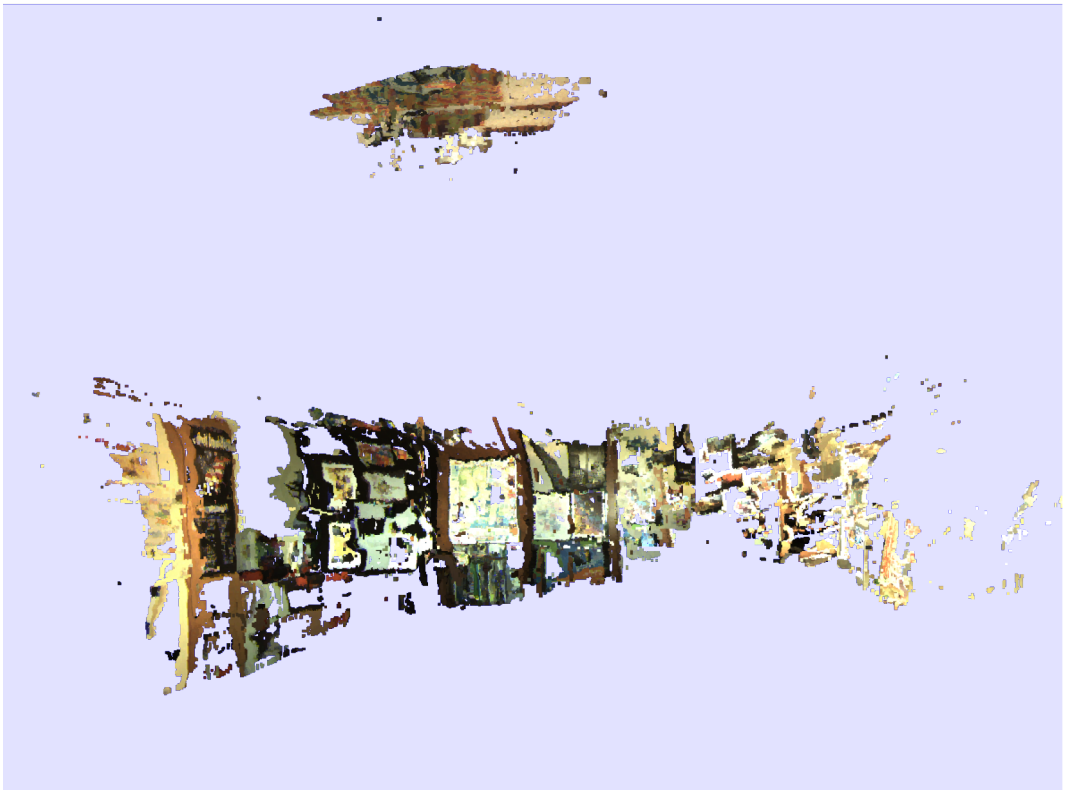

**Figure 6.36: A 3D photorealistic map of the environment obtained during the movement along the trapezoidal trajectory.**

**Figure 6.37: A close-up view of the 3D photorealistic map of the environment obtained during the movement along the trapezoidal trajectory.**

## Result Analysis

This section shows some results of the 3D photorealistic obtained using the MCU system. The maps presented here contain around one to two million 3D points per map, which provides dense geometric and detailed texture descriptions of the environment. The quality of the captured map can be seen better from a close-up view of the model such as the one in Figure 6.37.

## 6.5 Conclusion

This chapter presented the result of the experiments for 3D motion estimation and 3D map building tasks using the MCU system. Several trajectories were introduced and the MCU is capable of detecting the on-going trajectory at good accuracy in real-time. The motion estimation experiment shows that the MCU system gives a better result compared to the single camera system in both rotation and translation tests. Several 3D photorealistic maps acquired during the motion estimation experiment are presented. The constructed maps successfully replicated the test environment and they provide good geometric descriptions and detailed textures in respect to the real environment.

# Chapter 7

# Conclusions

This work presents a state of the art multi-camera unit as a powerful 3D visual sensor for the real-time 3D motion estimation and map building tasks. The MCU hardware is designed and successfully implemented using three stereo cameras whose optical axis are pointing perpendicular to one other, i.e. each pointing in the direction of the x-, y- and z-axes. The stereo cameras are used due to their good overall abilities in terms of speed, resolution and size compared to the available visual sensors of the same application. The finished MCU hardware is compact in size and weight and it can be conveniently used as a hand-held device for a 3D map building task. The MCU hardware, together with the accompanied algorithm described in this work, is capable of detecting and estimating its 3D movement correctly and robustly using only the visual information available, and no auxiliary odometry devices are required.

The 3D motion estimation process is successfully implemented using the iterative closest point algorithm, where the corner features extracted from the environment are used as input. An outlier detection scheme that utilizes a unique arrangement of a three camera system is also implemented and it provides robust 3D motion detection by using only 2D motion vectors derived from three cameras. By using a multiple camera design, the motion ambiguity that is commonly found in a single camera system is eliminated. This was proven by the simulation results and the real world experiments, where the multi-camera system performs better than the single camera and two camera systems.

A probabilistic 3D map building approach is used in order to cope with the measurement and map points' location uncertainty. The position of the 3D feature points extracted from the environment and the location of the MCU related to the environment are maintained using the FastSLAM algorithm. The 3D map building system using the FastSLAM algorithm is tested in a MATLAB simulation and good results are obtained. Afterwards, the FastSLAM algorithm is implemented using C++ language

in order to integrate it into the MCU software system. As a result, a complete 3D motion estimation and 3D map building system using the MCU is successfully implemented. The complete system is then tested in a real world environment and good results for 3D motion estimation are obtained. The MCU system is able to estimate its travelled trajectory at good accuracy. It can also incrementally merge the 3D images acquired from the stereo cameras into a 3D photorealistic map that correctly replicates the actual geometric and textual properties of the environment in a real-time manner.

## Possible Future Works

The current implementation of the MCU is only a first attempt to prove the concept of such multi-camera design. Further improvement that can be done to the MCU is to replace the stereo camera with a better visual sensor, e.g. a high resolution PMD camera. Another improvement is a design that uses more cameras in order to increase the accuracy and the redundancy of the 3D motion estimation process. In terms of software improvement, a better and faster map building algorithm is to be implemented. A faster computer with larger memory may also increase the overall speed of the system. Another possible configuration might be to dedicate a separate computer for the 3D photorealistic map building task. Currently, the 3D photorealistic map rendering software consumes a lot of processing time as well as heavy disk usage due to the vast number of 3D points within the map. Therefore a second computer that solely takes care of this task can improve the overall speed of the real-time map building system even further, allowing bigger environment sizes to be handled.

# Appendices

# Appendix A

# Hardware Drawing

## A.1 Mechanical Drawing of the STH-MDCS Stereo Camera

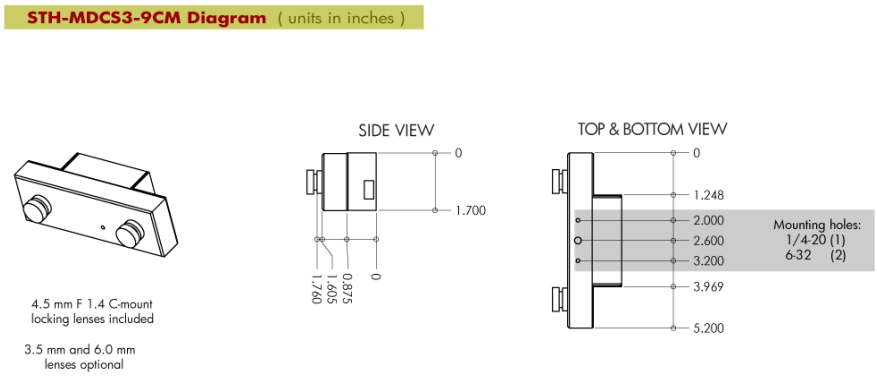


**Figure A.1: Drawing of the STH-MDCS stereo camera with dimensions. (photo credit: Videre Design)**

## A.2 Mechanical Drawing of the Multi-Camera Unit

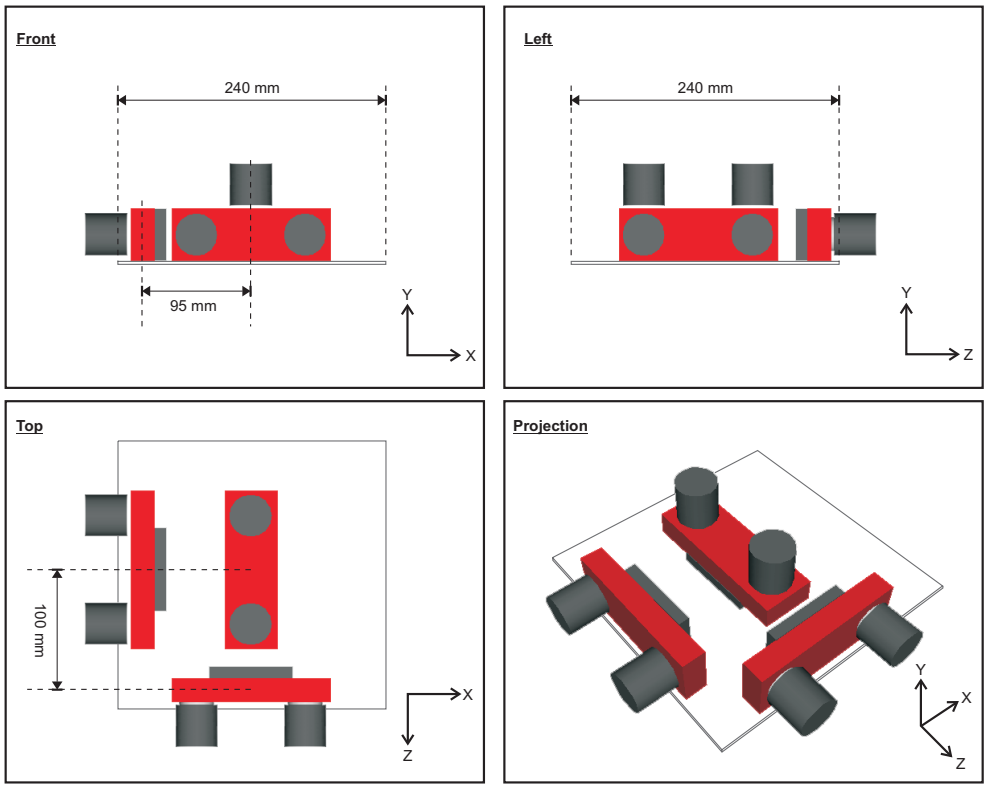


**Figure A.2: Drawing of the Multi-Camera Unit with dimensions.**

# Appendix B

# Error Characteristics of the Stereo Camera

## Error Characteristics of the Stereo Camera

The most useful information from a stereo camera is the depth information. Quality of the depth information depends on several factors such as the spatial resolution of the cameras sensor, the video signal noise, the algorithm being used, the distance to the observed object, etc. Even a small amount of mistake will lead to a significant error for the depth calculation. This error increases proportionally to the distance of the point observed and the longitude error is higher than the lateral Figure B.1.

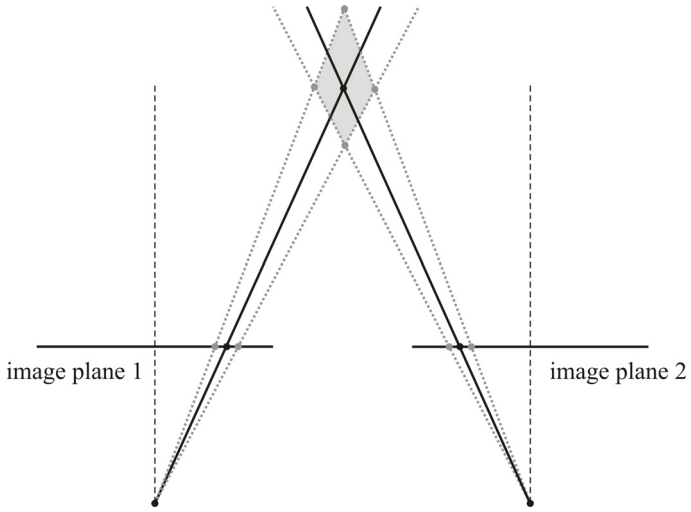


**Figure B.1: Stereo camera depth calculation error.**

## Stereo Camera Error Analysis

A stereo camera is subjected to high measurement errors once the target object is moved further away along the direction of its optical axis. In this section the real measurements were taken using the stereo camera which is used within the MCU system in order to evaluate its error characteristics. This includes the measurement error in the x-, y- and z-axes respectively. A test setup as shown in Figure B.2 is used to generate the test results. The test target with four rectangular shapes is used as reference points and the stereo camera is translated in the x-, y- and z-axes while the position of the centroid of the test target is being determined at each individual location.

The results show that the change of the stereo camera location in the x- and y-axes does increase the measurement error within the corresponding axis. However the change of the stereo camera location along the z-axis or the camera optical axis yields a large measurement error as the distance between the camera and the test target increase. These error measurements are shown in Figure B.3 and Figure B.4 respectively.


**Figure B.2: Setup for the testing of stereo camera depth calculation error.**

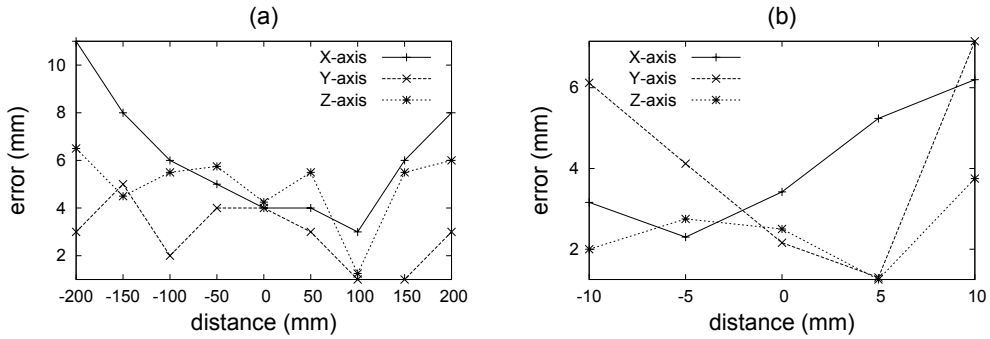


**Figure B.3: The measurement error versus camera displacement along (a) the x-axis and (b) the y-axis.**

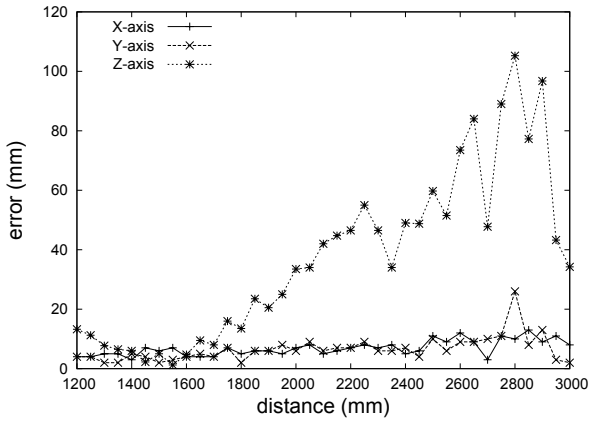


**Figure B.4: The measurement error versus camera displacement along the z-axis.**

# References


Adiv, G. (1989). Inherent Ambiguities in Recovering 3-D Motion and Structure from a Noisy Flow Field. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11.

Angelopoulou, E. & Wright, J.R. (1999). Laser Scanner Technology. Tech. rep., Department of Computer Science & Information Science, University of Pennsylvania.

Anton, H. (1994). *Elementary linear algebra*. John Wiley & Sons, Inc., $7^{\text{th}}$ edn.

Arun, K.S., Huang, T.S. & Blostein, S.D. (1987). Least-Squares Fitting of Two 3-D Point Sets. In *IEEE Transactions on Pattern analysis and machine intelligence*, vol. 9.

Bailey, T., Nieto, J. & Nebot, E.M. (2006). Consistency of the FastSLAM Algorithm. 424–429.

Belle, W.V. (2007). An Adaptive Filter for the Correct Localization of Subimages: FFT based Subimage Localization Requires Image Normalization to work properly. `http://werner.yellowcouch.org/Papers/subimg/index.html`.

Besl, P.J. & Mckay, N.D. (1992). A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**.

Biber, P., Fleck, S., Wand, M., Staneker, D. & Strasser, W. (2005). First Experiences with a Mobile Platform for Flexible 3D Model Acquisition in Indoor and Outdoor Environments - The Wägele. *In 3D-ARCH 2005, 3D Virtual Reconstruction and Visualization of Complex Architectures*.

Blanco, J.L. (2008). Derivation and Implementation of a Full 6D EKF-based Solution to Bearing-Range SLAM. Tech. rep., University of Malaga, Spain.

Davies, E.R. (2005). *Machine Vision: theory, algorithms, practicalities*. Morgan Kaufmann Publishers, San Francisco, CA, 3rd edn.

Davison, A.J. & Cid, Y.G. (2004). Real-Time 3D SLAM with Wide-Angle Vision. In *Symposium on Intelligent Autonomous Vehicle, IAV2004*, Lisboa, Portugal.

Druon, S., Aldon, M.J. & Crosnier, A. (2006). Color Constrained ICP for Registration of Large Unstructured 3D color Data Set. *IEEE International Conference on Information Acquisition*.

FARO Technologies Inc. (2009). FARO laser scanner PHOTON 120/20. `http://laser-scanner.faro.com/faro-laser-scanner-photon/`.

Figliola, R.S. & Beasley, D.E. (1995). *Theory and design for mechanical measurements*. John Wiley & Sons, Inc., 2nd edn.

Firoozfam, P. & Negahdaripour, S. (2003). A Multi-Camera Conical Imaging System for Robust 3D Motion Estimation, Positioning and Mapping from UAVs. In *IEEE International Conference on Advance Video and Signal Based Surveillance 2003*, Miami, USA.

Fischler, M.A. & Bolles, R.C. (1981). Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In *Communication of the ACM*, vol. 24.

Garcia, M.A. & Solanas, A. (2004). 3D Simultaneous Localization and Modeling from Stereo Vision. *International Conference on Robotics & Automation*.

Ghobadi, S.E., Hartmann, K., Weihs, W., Netramai, C., Loffeld, O. & Roth, H. (2006). Detection and Classification of Moving Objects - Stereo or Time-of-Flight Images. In *International Conference on Computational Intelligence and Security*, Guangzhou, China.

Jain, S. (2003). A survey of Laser Range Finding.

Joochim, C., Netramai, C. & Roth, H. (2009). Coordination of SLAM and Artificial Landmark Recognition using 2D/3D Sensors for Mobile Robot Vision. In *International Conference on Pattern Recognition and Information Processing*, Minsk, Belarus.

K.A. Schmersal GmbH & Co. Industrieschaltgeräte (1999). Sicherheits-Laserscanner LSS 300: Technische Beschreibung.

Knöppel, C. (2001). *Stereobasierte und spurgenaue Erkennung von Straßenfahrzeugen im Rückraum eines Straßenfahrzeuges*. Ph.D. thesis, Otto-von-Guericke-Universität Magdeburg, Magdeburg.

Konolige, K. & Beymer, D. (2007). SRI Small Vision System: User manual for software version 4.4.

Lewis, J. (1995). Fast Normalized Cross-Correlation. Tech. rep., Industrial Light & Magic.

Lewis, J. (2005). Fast Template Matching. *Vision Interface 95*.

Montemerlo, M. (2003). *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem With Unknown Data Association*. Ph.D. thesis, Carnegie Mellon University.

Montemerlo, M., Thrun, S., Koller, D. & Wegbreit, B. (2002). FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, 593–598.

Montemerlo, M., Thrun, S., Koller, D. & Wegbreit, B. (2003). Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *International Conference on Artificial Intelligence*, 1151–1156.

Negahdaripour, S. & Yu, C.H. (1988). Robust Recovery of Motion: Effects of Surface Orientation and Field of View. In *Computer Vision and Pattern Recognition, CVPR'88*.

Netramai, C. & Roth, H. (2007). Real-Time Photorealistic 3D Map Building Using Mobile Multiple Stereo Cameras Setup. In *3DTV Conference 2007*, Kos Island, Greece.

Netramai, C. & Roth, H. (2010). Real-Time 3D Motion Estimation and Map Building Using Enhanced Multi-Camera System. In *The $11^{th}$ International Conference on Modern Information and Electronic Technologies*, Odessa, Ukraine.

Netramai, C., Melnychuk, O., Joochim, C. & Roth, H. (2008a). Combining PMD and Stereo Camera for Motion Estimation of a Mobile Robot. In *The 17$^{th}$ IFAC World Congress*, Seoul, Korea.

Netramai, C., Melnychuk, O., Joochim, C. & Roth, H. (2008b). Motion Estimation of a Mobile Robot using different types of 3D Sensors. In *The 4$^{th}$ International Conference on Autonomic and Autonomous Systems (ICAS)*, Gosier, Guadeloupe.

Nüchter, A., Lingemann, K., Hertzberg, J. & Surmann, H. (2005). Heuristic-Based Laser Scan Matching for Outdoor 6D SLAM. Koblenz, Germany.

Pollefeys, M. (2000). Tutorial on 3D Modeling from Images. *Lecture Notes, In conjunction with ECCV 2000*.

Ringbeck, T., Möller, T. & Hagebeuker, B. (2007). Multidimensional measurement by using 3-D PMD sensors. In *Advance in Radio Science*, vol. 5.

Rusinkiewicz, S. & Levoy, M. (2001). Efficient Variants of the ICP Algorithm. *Third International Conference on 3-D Digital Imaging and Modelling*.

Sáez, J.M. & Escolano, F. (2004). A Global 3D Map-Building Approach using Stereo Vision.

SICK AG. (2009). LMS200-30160 online datasheet. `https://www.mysick.com`.

Sim, R., Elinas, P., Griffin, M. & Little, J.J. (2005). Vision-based SLAM using Rao-Blackwellised Particle Filter. In *Reasoning with Uncertainty in Robotics, IJCAI Workshop 2005*, Edinburgh, Scotland.

Sorkine, O. (2005). SVD - theory review and applications in computer graphics spring2004/05. `http://mrl.nyu.edu/~sorkine/courses/cg/cg2005/`.

Stroustrup, B. (1997). *The C Programming Language*. Addison Wesley, Indianapolis, IN, 3$^{\text{rd}}$ edn.

Surmann, H., Nüchter, A., Lingemann, K. & Hertzberg, J. (2006). 6D SLAM – Mapping Outdoor Environment. Gaithersburg, Maryland.

Thrun, S. (2002). Robotic Mapping: A Survey. Tech. rep., School of Computer Science, Carnegie Mellon University.

Trajković, M. & Hedley, M. (1998). Fast Corner Detection. *Image and Vision Computing*, **16**.

Tsai, R.Y. (1986). An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision. Miami Beach, FL.

Videre Design LLC (2009). Introduction for stereo cameras from Videre Design. `http://www.videredesign.com/vision/stereo_intro.htm`.

Xu, Z., Schwarte, R., Heinol, H., Buxbaum, B. & Ringbeck, T. (2005). Smart Pixel – photonic mixer device (PMD). Tech. rep.

Zhang, Z. (1994). Iterative Point Matching for Registration of Free-Form Curves. *International Journal of Computer Vision*, **13**.