FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Fall Detection Using Rotatable Depth Sensors

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

David Fankhauser, BSc.
Matrikelnummer 1025876

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: PD. Dr. Martin Kampel

Wien, 6. Oktober 2016

David Fankhauser | Martin Kampel

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Fall Detection Using Rotatable Depth Sensors

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

David Fankhauser, BSc.
Registration Number 1025876

to the Faculty of Informatics
at the TU Wien
Advisor: PD. Dr. Martin Kampel

Vienna, 6th October, 2016

David Fankhauser | Martin Kampel

Erklärung zur Verfassung der Arbeit

David Fankhauser, BSc.
Schanzstraße 23
1140 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 6. Oktober 2016

David Fankhauser

Danksagung

Ich möchte meinem Betreuer, Martin Kampel, für die Unterstützung danken. Auch ein großes Danke an meinen Arbeitskollegen Christopher Pramerdorfer für die Unterstützung in jeglicher Hinsicht. Zu guter Letzt möchte ich meinen lieben Eltern Irene und Martin danken, die immer hinter mir standen und mir meine akademische Laufbahn ermöglicht haben.

Acknowledgements

I would like to thank my supervisor, Martin Kampel, for supporting me. Also, a big thanks goes to my co-worker Christopher Pramerdorfer for supporting me in every concern. Last but not least I want to thank my loving parents Irene and Martin who always stood behind me and enabled my academic career.

Kurzfassung

Stürze sind die wahrscheinlichste Ursache für unfallsbedingten Tod von Personen die 65 Jahre oder älter sind. Gestürzte Personen können mit Hilfe von Tiefensensoren die mittels strukturiertem Licht arbeiten, detektiert werden. Somit kann sofortige Hilfe geleistet werden und Morbidität und Mortalität gesenkt werden. Tiefensensoren die mittels strukturiertem Licht arbeiten schützen die Privatsphäre, sind unauffällig und funktionieren auch ohne sichtbares Licht. Jedoch haben sie einen entscheidenen Nachteil und zwar ein geringes Sichtfeld. Diese Limitierung kann mit Hilfe eines Schwenkneigekopfs gelöst werden, der einen Tiefensensor bewegt, sodass dieser einer Person folgt. Als Konsequenz wird das Sichtfeld des Sturzerkennungssystems um einen entscheidenden Anteil erhöht.

Diese Diplomarbeit schlägt eine Methode vor wie ein existierendes Sturzerkennungssystem, das sich in der Praxis bewährt hat, die induzierten Sensorbewegungen kompensieren kann. Synthetische Daten werden erzeugt, um verschiedene Artefakte, die mit solchen Tiefensensoren auftreten, zu isolieren und zu analysieren. Weitergehend werden auch Artefakte untersucht die nur auftreten wenn Tiefensensoren bewegt werden. Reale Daten werden aufgenommen um die Genauigkeit des Sturzerkennungssystems während Sensorbewegungen zu bestimmen. Resultate zeigen, dass Stürze bis zu $4m$ Entfernung unter Sensorbewegungen detektiert werden können, wobei Sensoren auf einer Höhe von $1.15m$ am besten funktionieren. Eine voll funktionierende Implementierung in Form von lose gekoppelten Komponenten wird vorgestellt, welche Benutzer Rückmeldungen liefert und Fernsteuerung ermöglicht.

Abstract

Falls are the leading cause of accidental death among people that are 65 or older. Fallen persons can be detected using structured light depth sensors to provide immediate help and decrease morbidity and mortality rates. Structured light depth sensors are privacy preserving, unobtrusive and can operate without viewable light. However, they suffer a major disadvantage, namely their narrow field of view. This limitation can be tackled utilizing a pan/tilt unit that rotates a depth sensor to follow a person. As a consequence the field of view of the fall detection system is increased by a significant amount.

This thesis proposes a method to extend an existing fall detection system that has proven to work reliably in practice to compensate induced sensor movements. Synthetic data is created to isolate and analyze different artifacts that arise when using structured light depth sensors. Furthermore, artifacts are investigated that explicitly occur when moving the sensor. Real data is recorded to evaluate the performance of the fall detection system during sensor pan and tilt. Results show that falls can be detected up to $4m$ under sensor movements achieving the best accuracies when mounted on $1.15m$ above the ground. A fully functional implementation in form of loosely coupled components is proposed that supports user feedback and remote control.

Contents

CHAPTER 1

Introduction

Between the years 2004 and 2050 the number of people worldwide who are older than 65 years will increase from 461 million to an estimated 2 billion [1], an increase of 434% in only 46 years. According to a study in 2016 [2], 30% of people above the age of 65, fall at least once a year. The risk of elderly people becoming seriously injured or even die after a fall, ranges between 10% to 20% [3]. This makes falling the leading cause of accidental death amongst the elderly population [4]. Moreover, falls not only lead to physical but also to psychological injuries. The fear of falling is one of the main reasons that elderly people limit themselves in daily living activities [5].

The elderly people who live in senior homes frequently fall when they are not observed by care staff [6]. They can recover on their own which results in undetected and unreported injuries, and thus, a significant under-estimation in fall statistics [7]. This consequently highlights the task of detecting and reporting falls as essential to a number of papers which covered surveys on published systems [8, 9, 10, 11, 12]. Furthermore, people feel safer and independent if there is a fall detection system within their homes [13].

Igual et al. [8] classify fall detection system into wearable devices and context-aware systems. The former is defined as an miniature body-borne computational sensory device that is worn under, over, or in clothing [1]. This devices include sensors such as accelerometers and gyroscopes. These sensors are build, for example, into smartphones allowing it to directly integrate a fall detection system into a device that is already carried all the time [14], which increases the acceptance. However, such systems produce a rather high number of false alarms [8]. The reason for this is that smartphones do not get carried in a standardized position when used in practice. Moreover, persons do not carry their smartphones all the time which makes this kind of fall detection impossible.

[1] https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/wearable-computing Last accessed 02 Dec 2016

The second class of fall detection systems are context-aware systems, which are deployed in the environment. The main advantage in relation to wearable devices is that the user does not have to carry a sensor. However, operation is limited to regions where the system is deployed. Common sensors are (depth)cameras, microphones and pressure sensors [8].

Pressure sensors measure the applied force. They can be installed on the floor and monitor vibrations produced by humans. Alwan et al. [15] use this vibration patterns to distinguish between daily activities and falls. However, such sensors can be used to detect uprising person[2]. This can provide help for staff in elderly care homes to provide help on frail patients. Such pressure systems are cheap, however they suffer fall detection accuracies below 90% [10]. Microphones can boost the accuracy when using the received audio signals in conjunction with floor vibrations. Litvak et al. [16] propose such a system with a sensitivity of 95% and specificity of 95%. This high numbers are often reached under experimental conditions, but they drop significantly when applied in realistic scenarios [17].

Camera-based systems are increasingly included in smart-home solutions and their prices have decreased rapidly [18]. A main advantage is that person's behavior can be passively modeled [19] while fall detection is performed. Furthermore, other events like stand-up detection can be performed simultaneously [20]. Concerning privacy, red-green-blue (RGB) cameras have a significant disadvantage, since persons can be identified on the video footage. However, depth sensors like the *Microsoft Kinect* only enables privacy, when only using depth information. This is an important aspect in a real-world application [21] as high numbers of research papers utilize depth data as input for an event detection system [22, 23, 24]. Employing such systems on cheap low-end hardware becoming increasingly popular, since hardware gets increasingly more powerful [25]. Thus, this paper focuses on depth-data based fall detection system.

1.1 Problem Definition

Low-cost depth sensors have a massive disadvantage when deploying them, namely their low field of view (FOV). For example, the Microsoft Kinect has a FOV range of 57° horizontally and 43° vertically. This leaves blind spots when installing the sensor on the wall or in a room corner.

This paper focuses on how the FOV can be extended by rotating the depth sensor depending on a person's location. Fall detection is performed simultaneously using *fearless*, an existing fall detection approach [26][27][28]. *fearless* has multiple advantages in contrast to other depth-data based fall detection systems. It is designed to perform a machine-learning approach on single-board hardware. Calibration is performed fully automatically, making it plug and playable. In addition, *fearless* is iteratively refined

[2] http://www. google.com/patents/US6067019 Last accessed 02 Dec 2016

Figure 1.1: Visualization of a fall event created by *fearless*.

since 2013 based on long-term evaluation under practical conditions [29, 30, 31]. A visualization of a fall event created by *fearless* can be seen in Figure 1.1.

The algorithmic pipeline of *fearless* starts with the automatic calibration of the system. This involves detecting the ground plane to estimate the extrinsic camera parameters such as the sensor's rotation and height. After the system is calibrated it processes depth frames at a frequency of 15Hz. *fearless* operates in 3D voxel space, meaning that depth frames are converted and analysed in world coordinates. To restrict the search space of detecting fallen persons, a background subtraction based motion detector is used in a preceding step. Persons are detected using a random forest classifier. A second classifier decides whether a detected person has fallen down and raises an alarm.

1.2 Structure

The content of this thesis is structured as follows. Chapter 2 summarizes the current state-of-the-art of systems that use movable depth sensors. This comprise of fall detection systems, particularly because literature is limited on such systems. The emphasis will be on systems that use movable structured light depth sensors, as it is done in this paper. The fall detection system (*fearless*) is explained in Chapter 3. Besides the algorithm of *fearless* its hardware components, i.e. structured light depth sensors will be explained.

The contributed changes to *fearless* to support movable depth sensors are explained in Chapter 4. They concern with the motion detection part and the fall detection part of *fearless*. The former stands at the beginning of the fall detection pipeline and every other step depends on its outcome. The *fearless* motion detector is static and thus replaced by a *dynamic motion detector*, that sources of inaccuracies are analyzed. This

chapter includes the collection of real and synthetic data to test the dynamic motion detector and the fall detection system. Furthermore, this chapter includes the detection of uncontrolled sensor movements.

The implementation of the novel fall detection system as a distribution of loosely coupled components combined with user feedback and remote control is explained in Chapter 5. Chapter 6 shows the results of the fall detection system while the depth sensor is moving, the evaluation of the dynamic motion detector and its sources of errors. Finally, the chapter shows the results of detecting uncontrolled sensor movements. Chapter 7 concludes this work.

1.3 Contributions

fearless suffers limitations that are eliminated in this thesis:

- It is designed for a static setup. When the depth sensor is moved the system will enter an invalid state producing arbitrary results. The main contribution of this work is to adapt *fearless* to compensate these movement changes and perform fall detection while its depth sensor is moving. This is done by developing a dynamic motion detector to support motion detection during sensor pan and tilt. Furthermore, a dynamic region of intereset (ROI) is implemented that moves along with the sensor FOV.
- Empirical observations have shown that depth sensors might get displaced when the room is cleaned, or people suffering from dementia manipulating it. Thus, a further contribution is to detect if the sensor is manipulated and react accordingly. This is done by developing a motion based tampering detection algorithm.
- *fearless* lacks a well-designed implementation. A third contribution is to implementing a fall detection system that is distributed in a sense of loosely coupled components to enable maintenance and expansion. This system comprises of user feedback via light-emitting diode (LED) display and remote control.

CHAPTER 2

State of the Art

Performing fall detection with moveable depth sensors is a unique problem. Hence, the availability of publications is limited. Mundher et al. [32] published a fall detection system on a movable robot in 2014. This system is described in Section 2.1. Section 2.2 focuses on state-of-the-art of more generic systems, namely systems that use movable structured light depth sensors. A large number of approaches that utilize hand-held depth sensors to build point cloud representations of objects and environments have been published [33, 34, 35]. However, since this work is about controlled movements which is generally associated with robotics, only systems that use a computer-controlled unit to move the depth sensor will be investigated.

Section 2.3 investigates state-of-the-art methods for performing motion detection with moving sensors. This is done, since *fearless* uses a static motion detector that is exchanged with the proposed motion detector in Section 4.2.

2.1 Fall Detection on a Robot

A robot is a machine that is guided by software such that it is able to perform complex tasks automatically [36]. Mundher et al. [32] use a Kinect sensor on a mobile robot system to follow persons and detect if its target person has fallen. They use the skeleton information provided by the Kinect software development kit (SDK) that are proposed by Microsoft Research [37]. This information is extracted from single depth images, thus no motion detection is performed. However, persons are only detected if they face the sensor which is a major limitation in the context of fall detection.

The robot starts and stops following a person using gesture recognition. The gestures are *right hand up* and *left hand up*, which are detected by comparing the hand joints to the shoulder joints. Fall detection is performed by computing the distances of the body joints to the floor. Thus, the ground plane must be estimated in a prior step. The

authors do not specify the exact height of the sensor, but their illustration suggest that its height is about 20cm above the ground. At such a low height the sensor must be tilted upwards to include the person to its full extent. However, this draws major disadvantages on the floor detection, since only a small portion of the floor is included in the depth image. Due to this, falls are detected in the *skeleton space coordinate system* [37]. In this coordinate system the joint positions are simply thresholded.

Falls are detected at distances of 2.0m, 2.5m, 3.0 and 3.5m with accuracies of 50%, 75%, 100% and 100%, respectively. However, only 4 scenarios are detected per distance to generate this results. Bagalà et al. [38] found that threshold based methods proved to be ineffective, since they are too simplistic. This applies to the approach from Mundher et al. [32] and in conjunction with the requirement of persons facing the sensor to be detected, a novel fall detection system on moving depth sensors is standing to reason.

2.2 Structured Light Depth Sensors on Robots

When dealing with robotics the simultaneous localization and mapping (SLAM) problem plays a major role [39]. SLAM is the computational problem of maintaining a representation of the unknown environment while keeping track of its own location. Usually three degrees of freedom (DOF) are given on a movable robot with a fixed sensor namely, height, tilt and roll. This work deals with a simpler version of the SLAM problem, since no translations are performed. Thus, only two DOF are given: pan, tilt.

Endres et al. [40] present a mapping system that generates highly accurate 3D maps using depth and RGB information by extracting visual feature points from the RGB images. The depth images are further used to localize this feature points in 3D. Since the Kinect provides a dense point cloud representation of the scene the depth image is subsampled to enabled real-time processing. Also, the robot's trajectory is used to project sensor data into common coordinate frames, which simplifies the matching. The authors released their source code online under an open-source license[1].

Biswas et al. [41] filters vertical planes in the depth data and downproject them onto 2D. The resulting map comprises of the walls of the scene which are subsequently used for localization and navigation. Their method is highly optimized for real-time application and runs with 30 frames per second (FPS) in a single thread at only 16% central processing unit (CPU) load. They achieved this number with an Intel Core i7 950.

Benavidez et al. [42] use depth data to compute the traversable area of the robot in image space using a gradient and a log filter. The color data is sent to a server, since the hardware resources for their approach are unsuitable to use on a robot. At the server, a neural network is utilized for pattern recognition and object tracking. The long-term goal of this system is to work autonomously in outdoor environments.

[1] http://ros.org/wiki/rgbdslam Last accessed 02 Dec 2016

Correa et al. [43] use a priorly defined topological map of the indoor environment to autonomously perform localization of their robot. The depth map is averaged over 80 rows of each column and used as the input of a artificial neural network. This network is trained to distinguish 8 cases such as 'path ahead', 'left path' et cetera. With this network the robot can locate itself in the topological map.

Maier et al. [44] use a priorly defined static map for localization and maintain a 3D environment representation model in form of an octree for navigation. This representation is build using only depth information, and when integrated over 28 frames, it can be utilized for real-time planning of collision-free paths. Real-time performance is enabled by scaling the depth image to 320x240 and operating with 6 fps on a remote quad core computer. Their approach estimates the robots pose with 6 DOF, allowing the robot to change its height or climb on objects.

Cunha et al. [45] use a priorly defined map of walls to perform localization. The depth data is used to detect walls similar to the *wall without contradiction* method [46]. They subsample the depth images to 128x96 which was proven to contain sufficient information for the wall detection. This enables an extremely fast 3 DOF pose estimation which runs at 162 FPS on an Intel Core i3 @ 2.4GHz.

Doisy et al. [47] propose two algorithms that only use depth information for a mobile autonomous robot that follows a person. Both algorithms use the person detection and tracking software provided by the Kinect SDK [37]. The first one reproduces the person path by using the position of the tracked person in every frame. The second one requires a pre-build map of the environment and uses it to compute the shortest path to the person.

Table 2.1 shows a compact comparison of the different approaches. This includes this work and the fall detection system from Mundher et al. [32]. The approaches are partitioned into DOF, operating location, sensor height, whether pre-knowledge is used, whether RGB information are used and whether the robots motion is used to support localization (also referred to as *odometry*).

It is highlighted that the sensor height plays an important factor in the context of fall detection. A low positioned sensor has to be tilted upwards to monitor persons to their full extend. However, in the context of fall detection the sensor should focus on monitoring floor regions. This can be compensated by using reasonable sensor heights (> 1m) as it is done in this work.

Although, all approaches use a form of localization algorithm, not all of them use odometry to enhance the localization accuracy. This paper however, solely uses odometry for localization by polling the pan and tilt values of the robot motors. This reduces the complexity of the SLAM problem significantly, which makes it possible to develop it in real-time for low-end hardware. However, the localization results completely depend on the robot's responses and allow no major inaccuracies.

None of these methods take sensor artifacts into account that arise when a sensor is

	DOF	Location	Height	Pre-Knowl.	RGB	Odometry
Endres et al. [40]	4	indoor	150cm*	no	yes	no
Biswas et al. [41]	3	indoor	30cm*	no	yes	yes
Benavidez et al. [42]	3	outdoor	40cm*	no	yes	yes
Correa et al. [43]	3	indoor	40cm*	yes	no	no
Maier et al. [44]	6	indoor	58cm	yes	no	yes
Cunha et al. [45]	3	indoor	80cm	yes	no	yes
Doisy et al. [47]	3	indoor	120cm*	yes and no	no	yes
Mundher et al. [32]	3	indoor	20cm*	no	no	no
Proposed method	2	indoor	115cm	no	no	yes

Table 2.1: Overview of systems using movable structured light depth sensors
*Not specified but estimated from provided images.

moved. Tourani et al. [48] demonstrates that such artifacts increase significantly with the moving speed of the sensor. This paper will therefore investigate the influence of sensor movement to the proposed algorithm.

2.3 Dynamic Motion Detection

Figure 2.1: Mosaic image of an outdoor scene [49].

Motion detection on pan/tilt units can be done using mosaic images [49, 50, 51]. This images are created in an initial phase or during runtime and are then used as a model for background subtraction [52]. Figure 2.1 shows an example of a mosaic which is an

outdoor scene from Bevilacaqua et al. [49].

Hayman et al. [53] perform motion detection using the pan and tilt values from a robot. They build a mosaic image using this values at runtime and handle errors that arise when moving the camera. This approach could be used in this work. However, due to the increased size of the mosaic image, the algorithm runs only at 10 FPS on images that are sampled down to 176x144 on a 2GHz P4 laptop. This is a major drawback, since the method should run besides a fall detection system on a single-board computer. Additionally, this work uses depth data instead of color data. This enables projection from and into world coordinates and thus rotation-center correction can be applied. In conclusion it seems reasonable to develop a novel dynamic motion detection algorithm that utilizes the advantages of depth data.

The current state of the art of fall detection systems that use moveable depth sensors is limited. However, robotic systems that utilize depth sensors for scene reconstruction is a trending research topic. Combining the knowledge of those systems helps to develop related problems associated to fall detection for elderly people.

CHAPTER 3

fearless

The foundation of this work is an existing fall detection system: **F**ear **E**limination **A**s **R**esolution for **L**oosing **E**lderly's **S**ubstantial **S**orrows, in short *fearless*. The method is depth-data based and operates in real-time on single-board computers that cost less than 50 euros. In contrast to other depth-data based fall detection systems, it is plug and playable meaning that the sensor can be placed anywhere in the room. The features used are invariant to sensor rotation, position and are robust to partial occlusions. Regarding privacy, the depth data is processed locally, meaning that alarms are only raised if a fall is detected.

fearless started as a research project in 2009 and its first output was published in 2011 by Planinc et al. [26, 27]. In 2013, Pramerdorfer [28] proposes a new version of *fearless* in the context of his master thesis. Further development is done in cooperation with an industry partner. *fearless* 2 [29] was released in 2015 and improves the core components of the original system. A long-term evaluation of *fearless* 2 under practical conditions was published in 2016 by Pramerdorfer et al. [31]. It shows the evaluation results under naturalistic conditions, that include nursing and assisted living homes on 53 devices. The installations were active for 5246 full days (125904 hours). The authors claim that this is the first long-term evaluation of a fall detection method that is based on depth-images. Additionally, the system is tested under experimental conditions which consists of a dataset of 146 fall sequences and 415 activities of daily living. Examining the contrast between experimental and naturalistic conditions is important, since new sources of false alarms can be identified.

In 2016 *fearless* 3 [30] was released, which introduced a better scene understanding by utilizing 3D instead of 2D computations and explicitly modeling occlusions. This work is build upon *fearless* 3 and the following sections describe the hardware components and the algorithmic pipeline.

3.1 Components

fearless consists of an inexpensive single-board computer and a structured light depth sensor. Figure 3.2 shows an example of the hardware setup using an *ASUS Xtion Pro* depth sensor that is connected to an *ODROID U3* via USB.

Figure 3.1: The two hardware components of the *fearless* system: single-board computer (left), depth sensor (right)

The single-board computer must be powerful enough to support more than 15 FPS, since *fearless* 3 operates with exactly 15 FPS.

3.1.1 Structured Light Depth-Sensors

A structured light depth sensor is used as an input device obtaining the data for the fall detection system. *fearless* supports multiple depth sensors including *ASUS Xtion Pro*, *Orbbec Astra* and the first generation of the *Microsoft Kinect*. They all share the same principle: structured light. Since the input data is the foundation of *fearless*, this principle is explained in further detail.

Figure 3.2 illustrates how depth data can be obtained by projecting a known pattern onto an object. This pattern is diffracted by the geometric shape. Observing the illuminated object from another angle yields to a deformed pattern, which can be analyzed to deduce the *disparity d*. In the simplest case, a sequence of stripe patterns with increasing black-white frequency is projected onto an object over time. This results in a binary code that encodes the disparity [55]. Knowing the *focal length f* and the

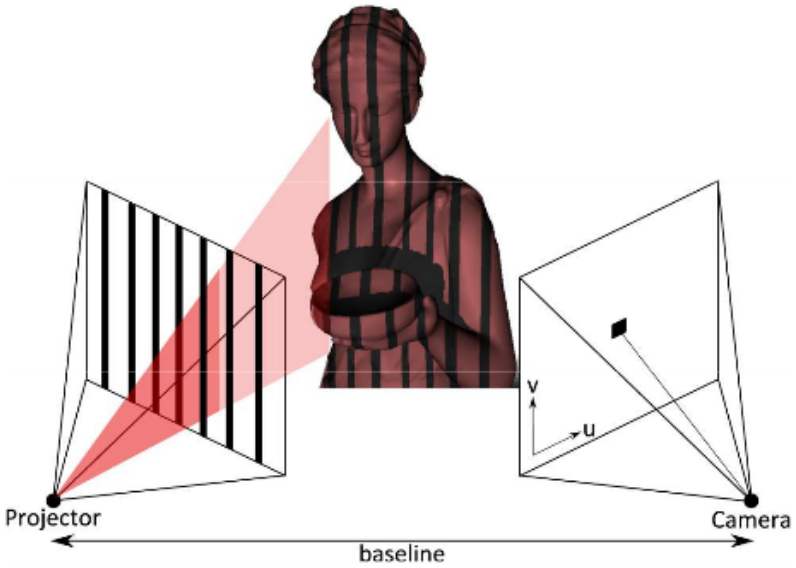


Figure 3.2: Principle of structured light. [54]

baseline b between the camera and the projector and assuming parallel optical axes, the disparity can be converted to the *depth* Z via triangulation, see Equation 3.1.

$$Z = \frac{b * f}{d} \tag{3.1}$$

However, structured light depth sensors as used by *fearless* implement the LIGHT CODING^{TM} technology that was developed by PrimeSense (bought by Apple Inc. in 2013). This technology describes the projection of a regular orthogonal grid infrared (IR) of light and dark speckles into the scene. The projected pattern is observed with a IR complementary metal oxide semiconductor (CMOS) sensor and deciphered to create a video graphics array (VGA) size depth image. Since the technology used is patented, detailing the creation of the dot pattern and the deciphering process can only be speculated. Yet, it can be assumed that sub-windows of the pattern are efficiently matched to the reference pattern to find the disparities used to compute the depth via triangulation. The density of the dot pattern decreases proportionally to the distance of the depth sensor. Thus, such sensors suffer a decreasing depth accuracy and increasing noise level. Therefore, a high upper depth boundary is essential in the context of fall detection, since sensors are installed on the wall or the ceiling and should observe the room to its full extent. Also, the FOV is an important criterion of a depth sensor. In conjunction with the depth range it defines the view frustum of the sensor.

[1] orbbec3d.com Last accessed 02 Dec 2016

	depth range	field of view	depth accuracy
Microsoft Kinect	0.8m-4.0m	57° H, 43° V	3.584mm@1.05m [56]
ASUS Xtion Pro	0.8m-3.5m	58° H, 45° V	2.573mm@1.05m [56]
Orbbec Astra	0.6m-8m	60° H, 49.5° V	5mm@2m[1]

Table 3.1: Comparison of different depth sensors.

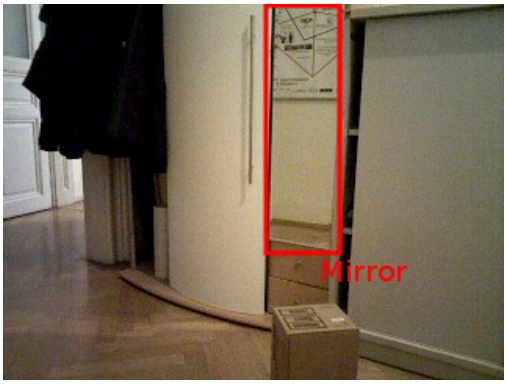
Table 3.1 shows a comparison of three depth sensors that are compatible with *fearless*. The Orbbec Astra has the highest depth range and the widest field of view. Although the Xtion Pro and the Kinect have a maximum range of $\leq$ 4m they can be used for higher ranges. However, they suffer major accuracy drops that are modeled by *fearless*. Guidi et al. [56] evaluate five depth sensors including the Kinect from Microsoft and the Xtion Pro from ASUS. Unfortunately, the Orbbec Astra is not tested by them and consequently, the depth accuracy can only be set in relation between the Kinect and the Xtion Pro. The latter shows a better accuracy at 1.05m. The Orbbec Astra is relatively new (shipped since 2016), hence at the time of writing this paper there are no published papers which assess the Orbbec Astra. Therefore, the depth accuracy is taken from the official Orbbec website.

Due to the fact of projecting and observing an IR pattern several problems arise in practice when this pattern is disturbed. A dominant problem is sunlight overlapping the projected IR pattern. This is because, sunlight that passes through a filter (i.e. curtain) may have enough remaining IR light to extinguish the projected pattern. Also, light sources emit IR light leading to errors when observed directly.

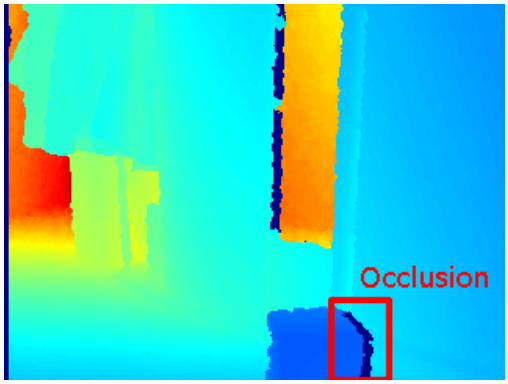
Therefore, projected IR light can only be observed if it is reflected to the camera. Glossy materials reflect the incoming light in a specular way. Thus, a light-ray of the IR projector bounces from a glossy object to another location where it may overlap with the correctly projected pattern. The IR sensor subsequently, receives reflected rays and reconstructs the depth based on these values. Hence, monitoring a flat glossy object like a mirror yields to noisy depth values of the reflection, see Figure 3.3. The depth values correspond to the distance from the sensor to the mirror to the reflected object. On the contrary, non-planar glossy surfaces distort the IR pattern which then cannot be reconstructed, yielding to black pixels in the depth image.

IR light can be incorrectly reflected but also absorbed by the material. This effect is amplified if the light is projected non-orthogonal onto the material. The higher the angle between projected light ray and surface normal the less light gets reflected.

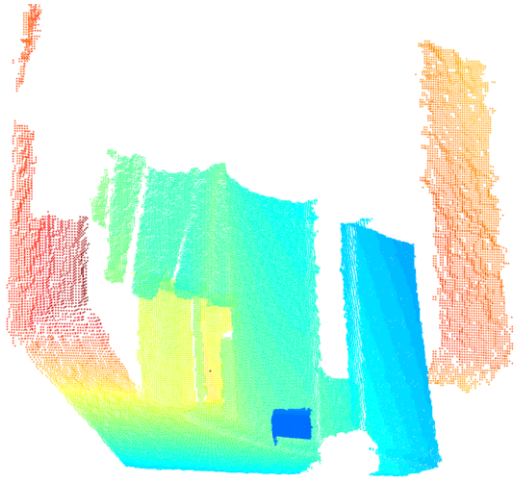
The IR projector and sensor are approximately 7.5cm apart. A high baseline is important to obtain a big triangulation angle, hence better depth accuracy. However, points may be occluded from the projector point of view but not from the sensor. The amount of occlusion is proportional to the distance of the occlude to the depth sensor. The depth image in Figure 3.3 illustrates an object that occludes parts of the background. The camera can 'see' a part of the unprojected space which leaves a hole in the depth



(a) Color image.



(b) Depth image.



(c) Point cloud.

Figure 3.3: Effect of a mirror on structured light depth sensors.

image.

All these problems play a vital role in the *fearless* system since the environment where it is installed cannot be controlled.

3.2 Algorithm

The algorithmic pipeline consists of four main components that are explained in greater details:

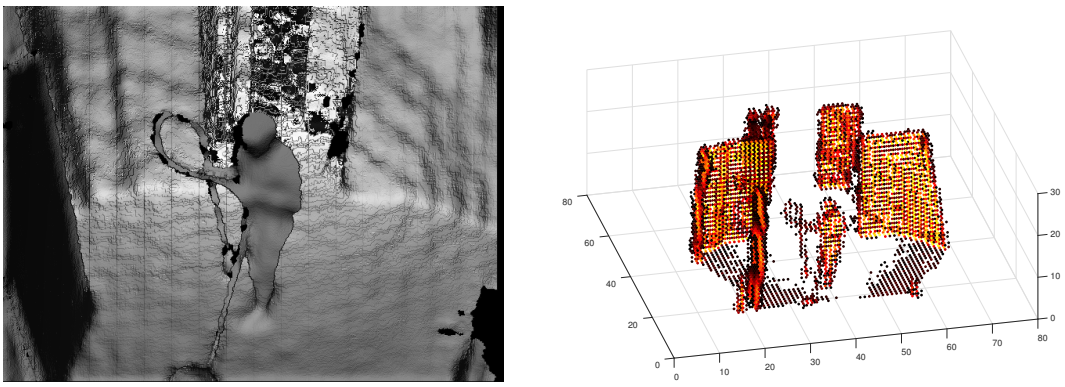
- **Scene Analysis.** Automatic calibration and scene analysis.
- **Motion detection.** Motion detection based on background subtraction.
- **Person tracking.** Person detection and tracking over time.

- **Event detection.** Detection of person events.

Further, the internal representation of objects and the modeling of occlusion is discussed.

3.2.1 Data Representation

fearless converts each depth image into world coordinates and subsequently discretizes the 3D points into a grid structure. The number of points falling into one cell is called *occupancy* and thus, these grids are called *occupancy grid*. Since the density of depth points decreases with the distance to the sensor, the occupancy value is corrected by dividing it through the squared distance to the sensor [57].



(a) Depth image. (b) Occupancy grid.

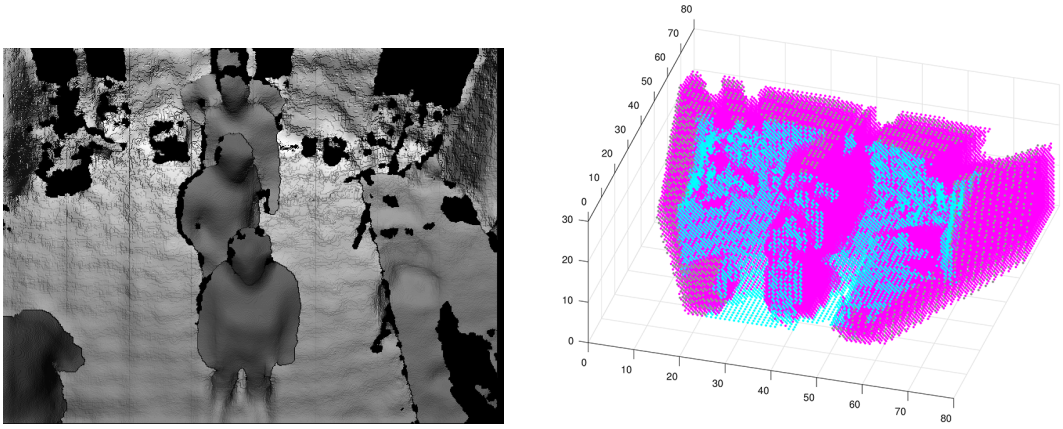
Figure 3.4: Shaded depth image of a scene and its corresponding occupancy grid. The data points in the occupancy grid are color-coded from black (low occupancy) to yellow (high occupancy). Points with occupancies values below 100 are not drawn.

The size of an occupancy grid is defined by the *cell size* $= 7.5cm$ and the *ROI*, which is set to $x \in [-3m, 3m]$, $y \in [-0.2m, 2m]$, $z \in [0.5m, 6m]$. The values are fixed, resulting in a grid size of 80x30x80. However, *fearless* restricts the ROI by means of finding the maximum depth in a preprocessing step to increase the computation speed. Noise reduction is performed after creation by filtering the grid with a 3D radial basis function kernel based on the *city block distance*.

Using motion detection, sub-occupancy grids can be extracted to represent moving objects and persons. Features are extracted based on these sub-grids to identify persons and classify their state.

A person can never be fully visible to the depth sensor due to occlusions and as a result it creates an incomplete sub-occupancy grid. *fearless* tries to model these occlusions by introducing *occlusion grids* which are similar to occupancy grids. They are defined as a discretization of world coordinates with the same ROI and cell size. However, occlusion

grids are created by casting a ray from the sensor to each cell, detecting if the cell is occluded. Hence, a cell of an occlusion map entails if it (i) is occluded (if yes also the occlude), (ii) lies within the FOV of the sensor and (iii) whether it lies on an object surface.



(a) Depth image. (b) Occlusion grid.

Figure 3.5: Shaded depth image of a scene and its corresponding occlusion map. Cyan: surface cells, magenta: occluded cells, gray: cells outside the FOV.

Figure 3.5 shows an occlusion map of a scene with four persons. Each person except for the one in the front is partially occluded. By using the occlusion map the visible part of a person can be extended along the y-axis (i.e. to the ground plane) yielding to an *occlusion score* to what extend a person is occluded. The occlusion score is used when detection person events.

Computing occlusion grids is expensive, since each cell must be ray-casted to the sensor. Thus, *fearless* creates the occlusion grid only at system start-up and updates it every frame based on result of a frame differencing motion detector.

Limitations

Occupancy and occlusion grids are created based on the ROI. The ROI is statically defined once at startup. Concerning pan and tilt movements of the sensor as it is the context of this thesis, *fearless* 3 only supports static scenes.

3.2.2 Scene Analysis

The scene analysis is performed when *fearless* is started. It entails (i) the detection of the floor plane to subsequently convert image points into world coordinates, (ii) the detection of static objects for example chairs, beds and desks to find resting areas and (iii) the generation of occlusion grids, which are used to explicitly model occlusions.

Pinhole Camera Model

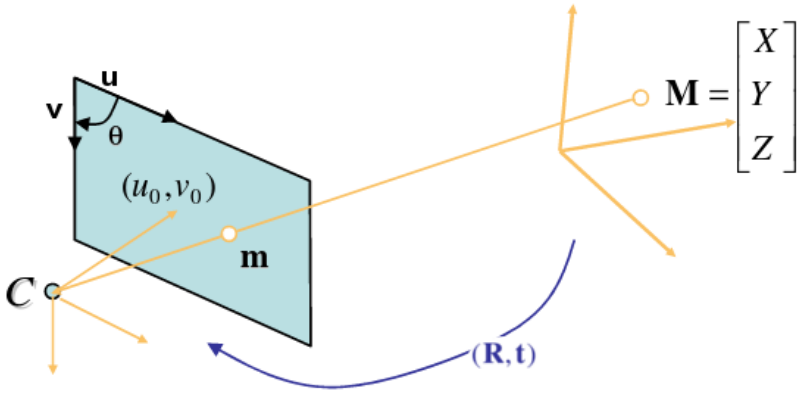


Figure 3.6: The pinhole camera model [58]

fearless uses the pinhole camera model [58]. Figure 3.6 illustrates how the model projects a 3D point $\mathbf{M} = [X, Y, Z]^T$ onto the image plane through the optical center $\mathbf{C}$ resulting in a 2D point $\mathbf{m} = [u, v]^T$. This mapping is called central projection. To be physically correct the image plane should be positioned behind the optical center, since a ray passed the optical center and subsequently hits the image plane. However, for illustration purposes it is placed in front of the optical center, which is mathematically equivalent. Defining the homogeneous coordinates $\tilde{\mathbf{m}} = [u, v, 1]^T$ and $\tilde{\mathbf{M}} = [X, Y, Z, 1]^T$ the relationship between $\mathbf{M}$ and its image $\mathbf{m}$ can be defined as

$$\tilde{\mathbf{m}} = \mathbf{A} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \tilde{\mathbf{M}} \tag{3.2}$$

$$\text{with } \mathbf{A} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

where $\mathbf{A}$ consists of the intrinsic camera parameters. The *focal length* f is the distance from the optical center to the image plane. In a true pinhole camera model both values f_x and f_y correspond to f, however the result of $\tilde{\mathbf{m}}$ should be in image coordinates thus pixels. Therefore $f_x = -m_x f$ and $f_y = -m_y f$ where m_x and m_y are the number of pixels per unit distance on the image sensor. Consecutively, the intersection of the z-axis with the image plane called *principle point* (u_0, v_0), which is $(0, 0)$ in the true pinhole model equals to the image center. The *skewness* s is defined as $s = f_x cot\theta$, with θ being the angle between the axis of the image plane. Assuming a rectangular image plane this parameter vanishes.

The intrinsic camera matrix $\mathbf{A}$ is found using camera calibration which transforms a 3D point from camera coordinates into image coordinates. Smisek et al. [59] calculated

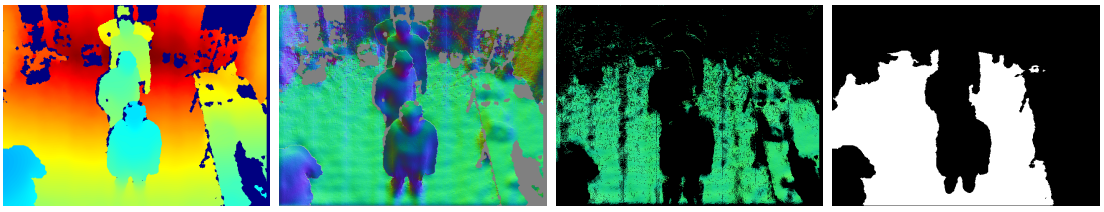
that $f_x = f_y = -585.6$px and $u_0 = 316$px, $v_0 = 247.6$px for the Microsoft Kinect. The principal point is shifted by approximately 3px from the image center in both directions. *fearless* uses these values to model the relationship between points in image coordinates and camera coordinates. Lens distortion is neglected since the accuracy is found sufficient.

The extrinsic matrix $\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}$ consists of a rotation and translation transforming world coordinates to camera coordinates. It is found automatically by detecting the ground plane in camera coordinates and comparing its normal vector against the known normal vector of the floor in world coordinates.

Floor Detection

Before converting the depth image into camera coordinates using the inverse of the intrinsics matrix $\mathbf{A}$, *fearless* applies a bilateral filter [60] to smooth the depth values locally. The cross-product of the tangents in x and y direction yields to a normal for each data point. The tangents are computed using the corresponding difference quotient. The points in camera coordinates are filtered to reduce the candidates for the ground plane estimation by restricting their corresponding normals. A sensor tilt of $[0°, 75°]$ and roll of $[-20°, 20°]$ are required.

A plane can be defined through three non-collinear 3D points. Applying random sample consensus (RANSAC)[61], *fearless* tries to find the correct ground plane iterating over random triples of the filtered depth points. The depth points are randomly subsampled to speed up the fitting process. An inlier is defined as a point lying at a perpendicular distance of ≤ 10cm to the found image plane. The set of inliers is used to define the final plane by using singular value decomposition (SVD) [62]. Since big planar objects (i.e. tables) may block the view and can result in a incorrectly detected ground plane, a search for a parallel plane is performed that is at least 500cm lower and must include at least half of the inliers of the original plane. Figure 3.7 shows the floor detection process based on a scene with a person queue.



(a) (b) (c) (d)

Figure 3.7: Floor detection process of *fearless*. Image (a) shows the depth smoothed with a bilateral filter. Pixels are colored based on their depth value. The computed normals (b) are mapped from $[-1, 1]$ to $[0, 255]$ for illustration purpose. Image (c) shows the filtered points after restricting their normals, which are used as basis in the RANSAC schema. The final inlier mask can be seen in Image (d).

Detection of Static Objects

Static objects are used to find out whether a person is sitting or lying when performing event detection. They are detected at system startup by analyzing cells in the occupancy grid that neither belong to the floor nor to the wall. Cells belonging to the wall are found analogue to the prior floor detection. The resulting occupancy grid is projected along the y-axis (i.e. from top) to obtain a binary image that encodes if a 3D column contains at least one cell with an occupancy over 100. Using connected component labeling [63] a first segmentation of static objects is achieved. Segments that are smaller than $0.25m^2$ or greater than $3.0m^2$ are discarded. A second segmentation is obtained by projecting the filtered occupancy grid along the z-axis (i.e. from front), to split objects that are on top of each other. The height of the remaining objects must lie in the interval $[0.25m, 0.8m]$ to be classified as a static object. Figure 3.8 shows a room with five detected static objects.

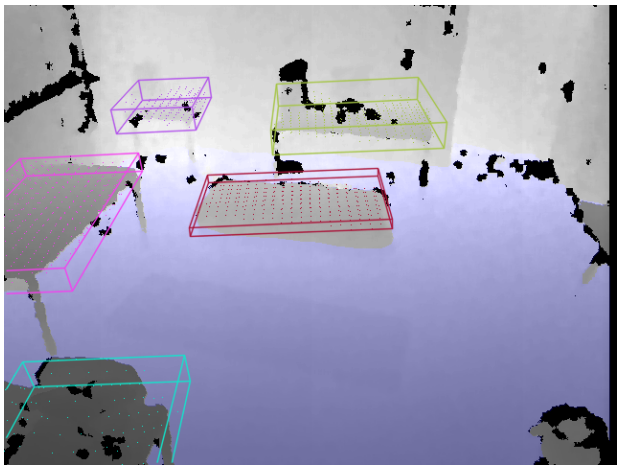


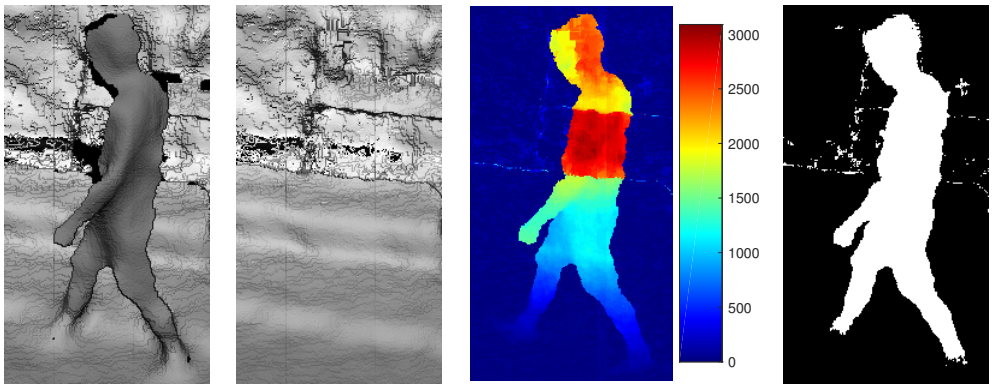
Figure 3.8: Static object segmentation of *fearless* 3 [30]. The ground plane is drawn in blue and the detected objects are illustrated with a random colored bounding box.

fearless periodically analyses static objects to detect whether they have been moved from their original position. When an object is moved, the occupancy of the original cells drop i.e. they become *invisible*. A *visibility score* is computed based on the ratio of visible cells to the total number of cells of an object, including occluded cells. If the visibility score falls below 0.25% the static object is no longer monitored and used for event detection.

3.2.3 Motion Detection

Motion detection comprises of the creation of *motion masks* encoding which depth points have changed. Such changes are created over time i.e. when a person walks by but also through noise that increases with the distance to the sensor. Moreover, it is used to restrict the search-space of objects by regions where motion occurs.

fearless uses background subtraction [64] to fulfill the task of motion detection. This is achieved by a *background model* depicting the static scene, which is then subtracted from the current depth image to obtain a difference image. A threshold relative to the depth value is then applied to filter significant changes between the frame and the model. It should be noted that differences higher than the threshold are classified as *foreground* and all other points as *background*. A motion mask is a binary image representing this classification. Figure 3.9 visualizes this steps.



(a) input frame (b) model (c) difference (d) classification

Figure 3.9: Motion Detection via background subtraction. The model (b) is subtracted from the input frame (a) resulting in a difference image (c). Using a threshold which is dependent on the depth value, a classification into foreground and background (d) is obtained.

Only points closer to the sensor are considered as motion whilst points 'behind' the background model are regarded as senseless. Such regions are called *ghosts* and can occur if a static object is moved. They need to be removed from the background model, since they distort the motion mask. This can be done periodically by overriding the ghost regions of the background model with the current depth image.

fearless uses the first frame on system startup to generate the background model. The model should not contain holes, however this occurs due to problems with structured light depth sensors. Thus, the first frame is interpolated before it is used as background model. This is attained by horizontally and vertically scanning for the first non-zero depth pixel.

Updating the background model is crucial, since the scene can change over time. In the case of *fearless* the system runs a full day before a daily system reset is done. Persons interact with objects and can change their location, hence the background model must adapt to such changes. This is implemented in *fearless* by adding ± 4.5mm per second to the model to balance it with the current frame. Thus, the model 'grows' to new objects and encloses them after a while. The time needed depends on the depth difference and is approximately 3 minutes for a chair. A fallen person will be included in the background

model if it does not move. This makes it important to detect the fall as soon as possible.

Detecting events in *fearless* is done on objects. Objects are sub-grids of occupancy maps that are identified by foreground regions. Occlusion grids however are updated using a frame differencing motion detector which is a simpler version of background subtraction. The previous frame is used as a model yielding to a motion classification from frame to frame. The occlusion grid is subsequently updated by ray-casting through each foreground pixel.

Limitations

Conventional background subtraction is limited to static scenes because changing the rotation of the sensor, which is the topic of this thesis, results in an invalid background model. Thus, *fearless* can only support moving sensors by stopping the fall detection system before the sensor is moved and restarting afterwards. Through the restart the system re-detects the ground plane of the new sensor position and can operate normally until the next sensor movement.

Further, *fearless* requires that the input data to be in front of the background model in order to get classified as foreground. Otherwise, ghosting is detected and the input data is written to the model. Detecting motion based on absolute background subtraction and disabling ghost detection is not supported.

3.2.4 Person Tracking

Fall detection is performed on all detected persons that are tracked over time. *fearless* distinguishes objects from persons. Objects are segmented from the occupancy grid and must only fulfill a size constraint of $[100, 600]$ cells. Persons however are converted from objects when they reach a person confidence of $\geq 75\%$. Their models are optimized for arbitrary pose of adults. Further, motion is estimated in 3D and the occlusion grid is analyzed to obtain a visibility score. When performing event detection this information is used for temporal integration of person state estimates.

Tracking

Figure 3.10 explains the tracking process. The location and velocity of a persons in frame $t-1$ is used to estimate the new location in frame t. A *kalman filter* [65] smooths the estimates of these values. This approach is called *detection-by-tracking* since a person is classified only once and tracked afterwards.

After persons are located the corresponding data points in the occupancy grid are cleared. Objects are updated analog. The grid only contains points that were previously classified as motion, hence only unmonitored objects remain which are later segmented and detected.

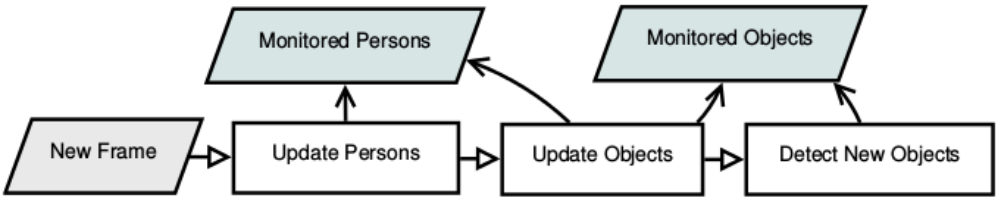


Figure 3.10: Person tracking pipeline of *fearless* 3 [30].

Feature Extraction and Person Classification

fearless uses a random forest classifier [66] to perform person classification. The classifier uses *slice features* which are obtained by vertically dividing the sub-occupancy grid into 15 slices and counting the occupancy cells that are non-zero per slice. Figure 3.11 illustrates the temporal evolution of the 15-dimensional feature vector.

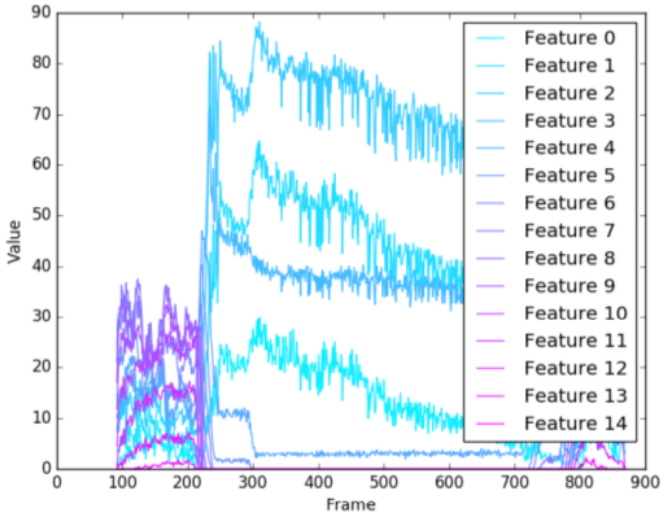


Figure 3.11: Slice features of an upright person that gets detected at frame 100 and falls around frame 220 [30]. The lower slice features increase significantly when the person is on the floor whereas the upper features vanish. This indicates that these features are discriminative for fall detection.

The random forest classifier training set consists of 12846 well visible person samples and 3577 non-person samples. All samples are clustered using agglomerative clustering to remove redundancies resulting in 8000 samples overall. Cross-validation is used to validate the number and depth of trees in the random forest.

3.2.5 Event Detection

The state of tracked persons is estimated in each frame. A random forest classifier is used with the same features and the same protocol that is used for person classification. This

classifier can distinguish between a person on the floor who is *lying*, kneeling or sitting and a person who is upright (*active*). Additionally, the overlap of the static objects is used to classify whether the person is *resting*. These three states (**resting**, **lying**, **active**) are further used for temporal integration.

Temporal Integration

The state estimates can be noisy since they are heavily dependent on the occupancy grid and the visible depth points of the person. Thus, Kalman filters [65] are used to combine individual state confidences which are obtained from consecutive frame measurements. The reliability of the estimates correlate with the person confidence, the degree of occlusion and the degree of invisibility due to the sensor field of view. Hence, the contribution of each estimate is chosen based on these measurements which yields to a powerful classification framework that takes scene conditions into account. Figure 3.12 illustrates temporal integration on a person walking around in a scene and sitting down.

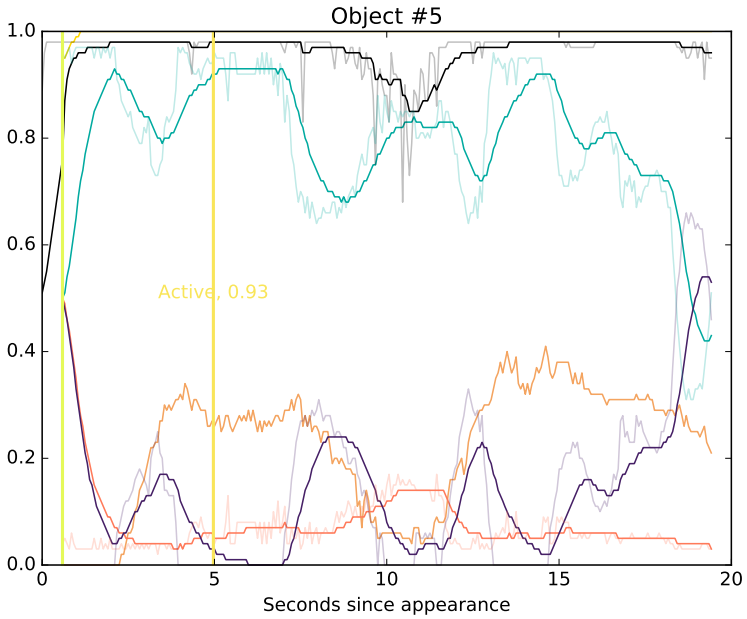


Figure 3.12: State estimates before (light colors) and after (dark colors) temporal integration of a person walking through the scene and sitting down [30]. The three state estimates are plotted in violet (resting), orange (lying) and cyan (active). The person confidence (black) is decreased midway of the sequence due to occlusions (red).

Events are detected by monitoring the state estimates after temporal integration. If an estimate exceeds 60% *fearless* raises an event:

- **Active.** Person is upright after being detected or resting.

- **Resting.** Person sat or lay on a static scene object.
- **Fall.** Person fell, lay or sat on the floor.
- **Recovery.** Person got off the floor.

Event Filtering

fearless detects events in each frame, hence, if the state estimates exceeded the threshold, event filtering would be initiated to reduce the number of reported events. Therefore, following event filters are implemented:

- **Delay.** An event is delayed for 5 seconds to allow the confidence of this event to increase. After the delay the event with the highest confidence is reported.
- **Timeout.** After an event is reported, *fearless* waits 5 minutes to report another event of the same type and person. Alternatively, if a consecutive event has a higher confidence than the reported one the filter is skipped.

Starting from 2011 this pipeline was refined using real data from elderly people. The algorithms shown in this chapter are the current state of the *fearless* system. More than 5 years of development and improvement are packed in this pipeline. That is the main reason *fearless* is chosen as a foundation of this thesis. However, the system lacks features for fall detection with moveable depth sensors which are introduced in this work.

CHAPTER 4

Methodology

Performing fall detection during sensor changes is build upon *fearless* [30]. *fearless* does not support movable sensors, thus several contributions are proposed to eliminate these limitations. This includes the adaption of the motion detection algorithm, to enable correct motion classification for dynamic sensors which is described in Section 4.2.

Data is collected from the pan tilt unit (PTU) and synthetically created as described in Section 4.1, to test and evaluate the developed methods. Regarding real data, a number of problems must be considered due to sensor limitations. These problems are analyzed in Section 4.3.

fearless does not only have no dynamic motion detection but it lacks the support for updating its ROI. The ROI is statically defined in world-space, depending on the sensor orientation where *fearless* is initialized. Section 4.4 explains the proposed solution to update the ROI depending on the sensor pan and tilt. In conjunction with dynamic motion detection, fall detection on moving sensors is supported.

The contributed changes to *fearless* allows for controlled sensor changes. However, uncontrolled changes such as a person intentionally or unintentionally altering the sensor position yield to an abnormal state. Experience has shown that persons might move the depth sensor. This happens for example if an individual cleaned their room or intentionally tampered with the sensor, especially if they suffer from dementia. Section 4.5 describes how sensor tampering is detected using motion detection and subsequently reinitializing *fearless*.

4.1 Data Collection

To test and evaluate *fearless* with moving sensor support a device specifically for the task of precisely moving a structured light depth sensor, namely the PTU is used. Additionally, synthetic data is generated to investigate the influence of non-perfect data as it is received

from the PTU to the proposed fall detection system. This data can be accurately modeled using a pinhole camera model because it suffers no inaccuracies, such as lens distortion or a decreasing depth resolution, in contrast to real data. Also, no problems due to rapid sensor movements arise.

4.1.1 Pan/Tilt Unit

The PTU is a remotely controllable device with a mounted depth sensor[1]. It was built to evaluate the developed algorithms while panning and tilting the sensor and to further examine the problems that arise. Additionally, the user receives feedback from the built-in RGB LED.

The casing of the depth sensor is replaced with a 3D printed casing to fixate it on the unit which makes the sensor exchangeable. Two stepper motors panning and tilting the unit head are controlled by a 16MHz ATmega32U4 micro controller. The motor position is set for the number of *steps* but can be converted into degrees, see Equations 4.1 and 4.2. The pan range lies between $\pm 70°$ and the tilt range between $5°$ to $-25°$. Besides the USB connection to the mounted depth sensor, the PTU can be connected via micro-USB to a computer and requires a 4A/5V power supply. Two RGB LEDs are added to the casing on the bottom part of the unit. Figure 4.1 shows the PTU on a tripod.

Figure 4.1: The PTU on a tripod.

The unit is equipped with three contacts: two of which are used to detect the pan range while the third is used to find the upper tilt bound. When supplying the PTU with power, it carries out a self-calibration cycle to detect its bounds:

1. pan left until left contact is reached.

[1] Asus Xtion PRO. Casings for other depth sensors can be printed

2. pan right until right contact is reached.
3. pan to center of found range.
4. tilt to top until top contact is reached.
5. tilt fixed number of steps down.
6. tilt to center of found range.

After the self-calibration phase the unit is ready to receive and send commands.

Communicating with the PTU is done via the USB *communication device class* CDC together with the *abstract control model* ACM. This makes the connected computer "think" it is communicating over the old-fashioned serial port. The benefit using this standardized type of communication is that, it is a simple way to exchange raw data and supported by all operating systems. A baud rate of 115200Bd must be defined for writing and reading to the serial port. The protocol for sending and receiving commands include five parameters. These are:

- **p** - pan
- **t** - tilt
- **r** - red color
- **g** - green color
- **b** - blue color

The pan and tilt parameter are set in steps and the color of the LED is set as an unsigned char $[0, 255]$.

The unit sends and receives commands at 50 Hz, consisting of an arbitrary subset of parameters, each split by a space and postfixed with ':value'. An example for setting the units LED to solid green, the pan to 5000 and tilt to 3000 steps looks like this:

```
r:0 g:255 b:0 p:5000 t:3000
```

An answer from the unit always comprises of the five parameters, which returns the status of the current position and the LED values. The step values are clamped to the minimum and maximum bounds of the motors. Therefore, sending the unit a pan/tilt value outside the range would therefore result in the unit going to the bound of its pan/tilt range. A joystick at the back of the PTU enables the user to control the unit manually.

A command always overwrites the last one. It is thus possible to send the unit to a new position even though the last command is not finished. Also, the unit accelerates and decelerates smoothly which prevents jittery depth images.

step division	accuracy pan	accuracy tilt	speed pan	speed tilt
1	0.0937°	0.0199°	75.5967°/s*	45.9922°/s
2	0.0469°	0.0100°	75.5967°/s*	22.9961°/s
4	0.0234°	0.0050°	54.0378°/s	11.4981°/s
8	0.0117°	0.0025°	27.0189°/s	5.7490°/s
16	0.0059°	0.0012°	13.5095°/s	2.8745°/s

Table 4.1: Accuracy and speed of the PTU with different step divisions.
*Maximal speed of the unit.

One step motor comprises 200 full steps per 360° and the motors use a transmission unit with a ratio of $1 : 19 + \frac{38}{187}$ for panning and $1 : 90.25$ for tilting. A step can be subdivided into $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$ steps reducing the speed but increasing the accuracy. Equations 4.1 and 4.2 are used to convert steps into degrees or visa-versa.

$$deg_{pan} = \frac{steps}{division} * \frac{360°}{200} * \frac{1}{19 + \frac{38}{187}} \tag{4.1}$$

$$deg_{tilt} = \frac{steps}{division} * \frac{360°}{200} * \frac{1}{90.25} \tag{4.2}$$

Using a step division of 8, 5000 steps are equal to

$$\frac{5000}{8} * \frac{360°}{200} * \frac{1}{19 + \frac{38}{187}} = 58.5840° \tag{4.3}$$

for panning and 3000 steps are equal to

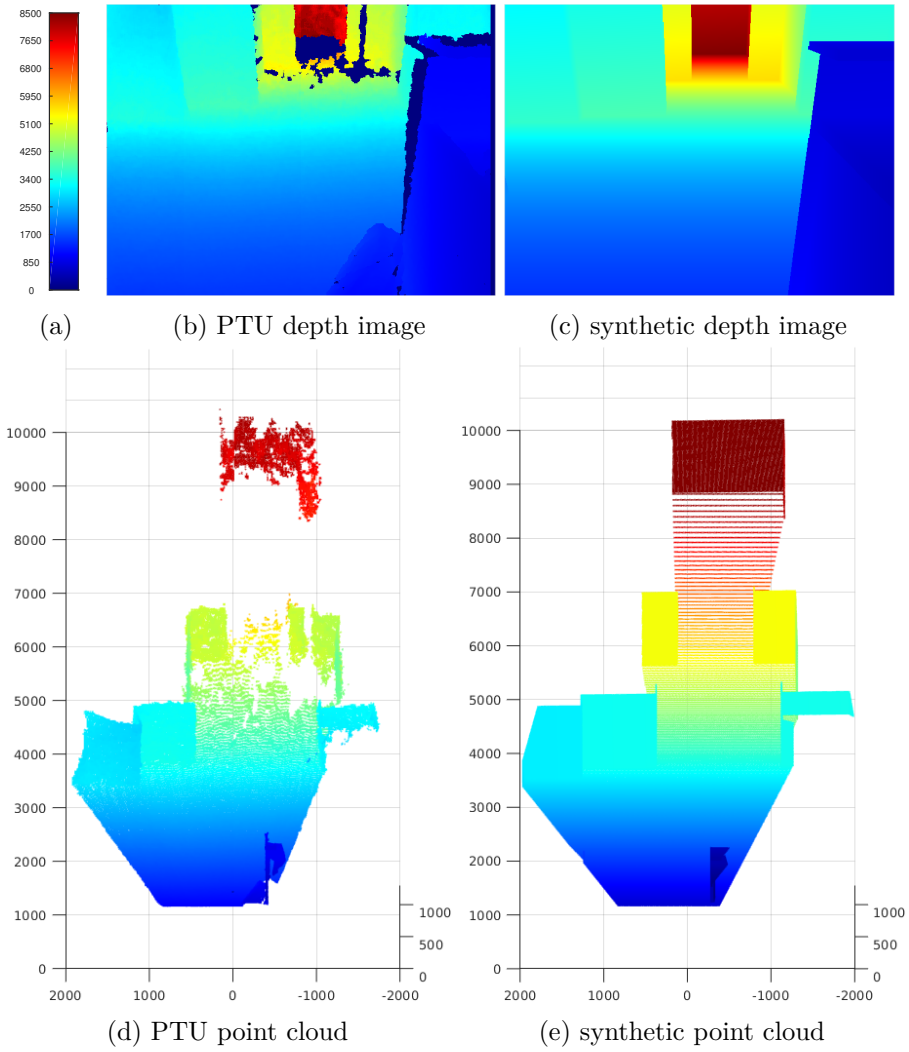
$$\frac{3000}{8} * \frac{360°}{200} * \frac{1}{90.25} = 7.4792° \tag{4.4}$$

for tilting. Both stepper motors run at 2306 steps per second when they are fully accelerated. Moreover, the accuracy can be obtained by converting a single step. Table 4.1 provides a summary of the accuracies and speeds at all step divisions. When using half or full steps the panning motor cannot keep up with the desired speed and only achieves 75.5967°/s.

Operating with low step division (i.e. high speed) makes the synchronization between the images from the depth sensor and the pan/tilt values from the unit a crucial task. The mounted depth sensor has a latency of about $\frac{2}{15}s$ whereas the PTU delivers the pan and tilt values with a latency in a microsecond range.

Further inaccuracies arise since the center of rotation is not the optical center of the IR sensor. Due to the hardware setup, a rotation of the PTU leads to a small translation (< 5cm) of the IR sensor. Thus, the distance of two arbitrary data points in the sensed image is not consistent after rotation.

4.1.2 Synthetic Data



(a) (b) PTU depth image (c) synthetic depth image

(d) PTU point cloud (e) synthetic point cloud

Figure 4.2: A scene recorded with the PTU (a) and created synthetically (c). The corresponding point clouds (d), (e) illustrate the differences in world coordinates. The color scale (a) maps colors to the distance to the sensor.

Data recorded with the PTU suffers a number of problems related to the depth sensor and the induced movements. To analyze these problems synthetic data is generated. This data can be modified to be perfectly accurate or isolate single problems. The influence on the algorithms used can then be quantified.

Synthetic data is generated using *Blender*, an open source 3D content-creation program. The internal camera parameters correspond to the depth sensor used in the PTU. Figure 4.2 shows a comparison of a scene that is created synthetically and recorded with the PTU. The synthetically created scene has no major drawbacks, except for the

limited density which occurs when the angle between the sensor ray to the ground plane decreases. This effect leads to a visible stripe pattern in the synthetic point cloud at ranges $\geq$ 6m, however, this produces no inaccuracies.

On the contrary, real-world data suffers a significant drop in depth resolution at increasing sensor distance. This can be observed in Figure 4.2 at distances $\geq$ 9m. Further, the lens distortion of the IR sensor leads to a pincushion effect that is proportional to the sensor distance. The PTU point cloud depicts this effect at the left image edge at 3.5m.

The PTU data is generated at a moderate panning speed (13.5°/s), hence no effects occur due to rapid sensor movements. However, at increasing speed two problems arise: (i) rolling shutter and (ii) motion blur. Rolling shutter is the effect of image sensors that read the sensor data top-down. This results in a skewed image while the sensor is moving. Comparatively, motion blur occurs when the image sensor is moved during a single exposure. As a result, objects get blurred along the moving direction.

Another problem in real world is the shifted center of rotation. Unlike the PTU the center of rotation can be placed exactly at the optical center of the camera. Thus, the distance between two arbitrary depth point on the sensed image remain constant after rotation. However, the effect of having a different rotation center can be emulated by virtually creating the same setup as it is given by the PTU.

4.2 Dynamic Motion Detection

fearless 3 uses a static background model that is stored in image space. Performing motion detection while rotating the sensor requires the model to compensate the induced movements. This correction is done using the corresponding pan and tilt values. These values refer to the world coordinate system: panning corresponds to a rotation around the worlds y-axis and tilting to the worlds x-axis. The image plane however, is oriented after the camera coordinate system. Its orientation only matches the world coordinate system if the sensor is not tilted, see Figure 4.3. To overcome this limitation the background model is converted into world coordinates.

4.2.1 Background Model

The background model can be created by scanning the whole FOV and projecting the current segment back into image coordinates. This method, however suffers two disadvantages, which makes it infeasible. First, the amount of data from the model which must be stored exceeds the limited resources given on a single-board computer. These are not only memory limitations, but a lack of computational power to handle the amount of data. Second, keeping data points that are not visible can lead to problems if the scene changes as areas cannot be updated accordingly if they are outside the FOV. Third, a static position is required. Considering future work, a motion detector should be developed that is capable of supporting translations as well.

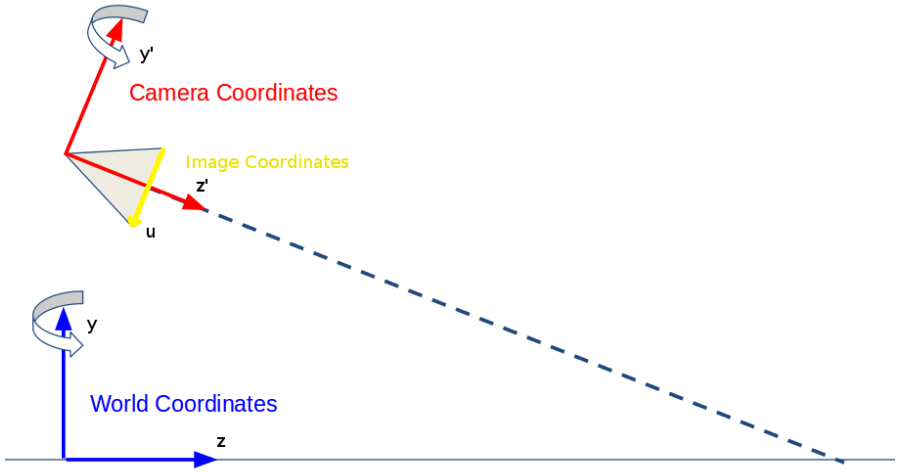


Figure 4.3: Panning a sensor in different coordinate systems.

For these reasons, a method is implemented that stores only visible data points. If the sensor moves, points that fall out of the view are discarded, while new data points become visible and are added to the background model. Figure 4.4 shows the background model of a sensor moving left-up.

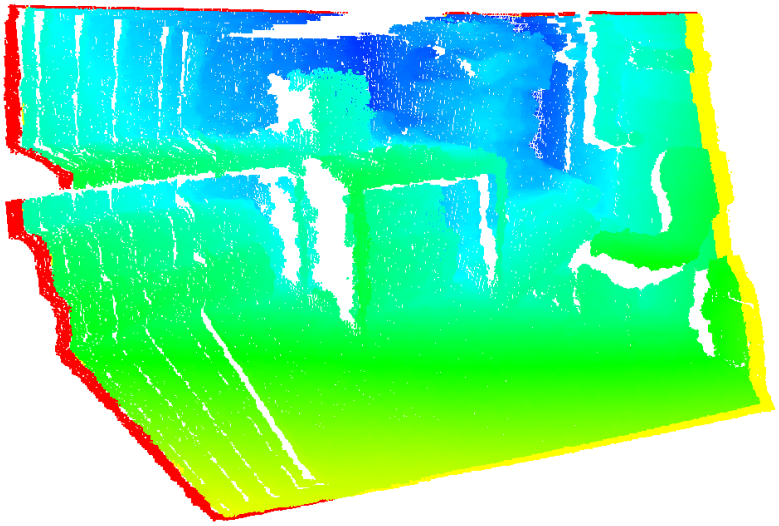


Figure 4.4: Point cloud of a background model in world coordinates when it is updated. Yellow points are discarded, red points are new.

When a new frame is available, the three dimensional model points are projected into image coordinates. Points that are outside the image bounds are discarded (yellow area in Figure 4.4). Figure 4.5 shows two kinds of holes that appear after projection:

projection artifacts and unobserved areas.

Projection artifacts originate from data points not equally falling into the pixel grid. There are pixels where more than one data point is projected to. However, these points are direct neighbors in world coordinates and the resulting error is negligible. Compensating for pixels into which no data point falls is done by filling them with the median of all non-zero points in the 8-connected neighborhood. This interpolated data points are not added to the background model, since it would increase the point cloud size with data that is interpolated. They are only used for the comparison between background model and current depth frame.

Unobserved areas, the second kind of projection holes, are regions where no data points exist. Filling this area is done using the current data. However, the current frame may have holes while the background model must not have empty regions. Thus, remaining holes are interpolated by searching the local neighborhood vertically and horizontally, using the first depth value found. In a subsequent step these new points are transformed into world coordinates and added to the model.

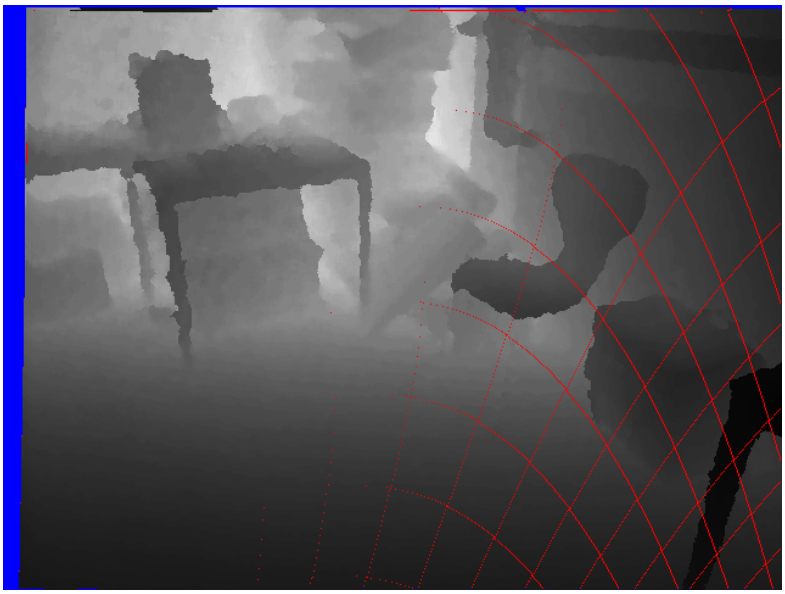


Figure 4.5: Holes after projecting the model in world coordinates into image coordinates. Projection artifacts are visualized red and unobserved areas blue.

4.2.2 Ghost Detection

When the sensor rotates, all data points along the rotation will be added to the background model, which includes persons. No foreground will be detected as long as the person is not moving. Additionally, a ghost is created if the person starts moving and must be detected and removed immediately.

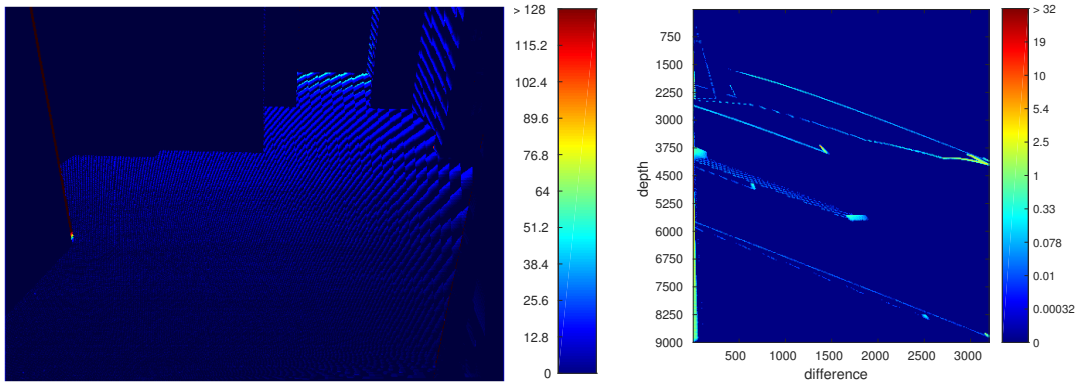
Ghost detection is done in image coordinates. This results in a set of new 2D points that depict the 'real' background. They are added to the 3D background model, whereas the ghost points must be removed from the model. The removal cannot be done in a reasonable time by exhaustive search, thus the newly added points are treated with a higher priority when the background model is projected back to image coordinates:

1. Project all points labeled as ghosts.

2. Project other points. If a projection falls into an occupied pixel do not override it and remove it from the background model.

4.2.3 Difference Maps

Difference Maps are introduced to illustrate the errors in the dynamic motion detector. They are bi-variant histograms that count the differences between the dynamic motion model and the current depth frame depending on the depth. The maps are normalized on the number of frames to be invariant according to the sequence length.



(a) (b) (c)

Figure 4.6: Image (a) shows a dynamic motion model minus a depth frame. The resulting differences are color-coded (b). The difference map is shown in image (c) after panning a sensor 180° in a static scene.

Difference maps contribute to the interpretation of inaccuracies due to real data. However, to avoid distorting this maps there should be no occurrence of real motion. Figure 4.6 shows an example of a difference map. The horizontal axis partitions the depth difference between the model and the current frame. Negative differences are clipped such that, the only points considered, lie in front of the dynamic background model. The vertical axis depicts the depth value of the current frame.

A depth sensor does not measure the Euclidean distance to a point in the scene, but the normal distance to the sensor plane. Thus, a point in the scene has different depth

values depending on the sensor rotation. To be invariant to sensor rotation, each depth point is converted to measure the euclidean distance before creating the difference map.

The density of the maps grows rapidly when looking at small difference values (< 100), since small errors appear more often. Thus, a non-linear density scale is used to visualize the full spectrum of errors.

The dynamic motion detector is evaluated with synthetically generated data that is manipulated to mimic problems with real data. In addition difference maps are used to visualize and analyze the impact of this data.

4.3 Problems with Real Data

Inaccuracies arise when handling real data as if it is collected from the PTU. These inaccuracies contribute to errors in the dynamic motion detector and are therefore analysed in greater detail:

- Pixel discretization.
- Center of rotation.
- Synchronization.
- Lens distortion.
- Depth accuracy and resolution.
- Rolling shutter and motion blur.

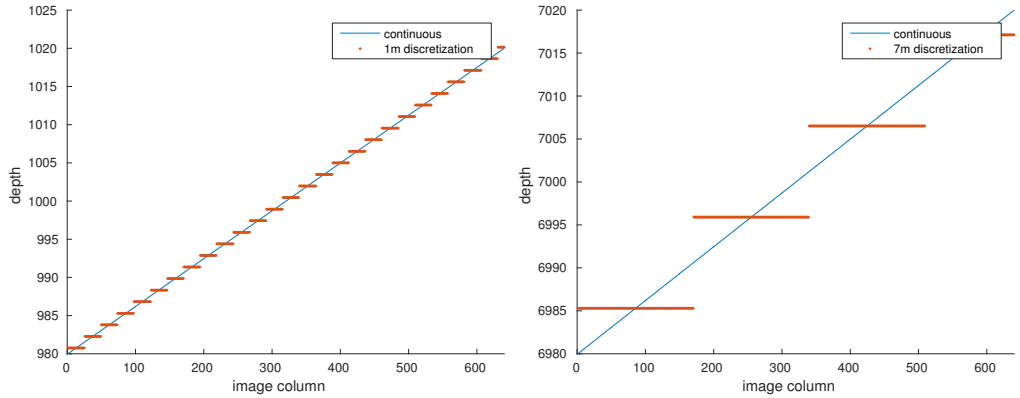
4.3.1 Pixel Discretization

The infrared camera of a structured-light depth sensor as the one used by *fearless* creates a VGA sized depth image. This image contains a constant number of 640x480 depth pixels. The density of pixels falling onto a flat surface is inversely proportional to the squared distance to the sensor [67]. Thus, the quantization error [2] increases at higher distances. The area of one pixel covers about $2.4mm^2$ at 1 meter distance. At 7 meter this area increases to $118.4mm^2$.

Figure 4.7 shows the discretization of a continuous depth signal at 1 meter and 7 meter. The quantization error strongly depends on the input data. Rapid depth changes, such as object edges, will significantly increase the quantization error. This happens because a pixel falling onto an object edge contains a single depth value that represents a wide depth range.

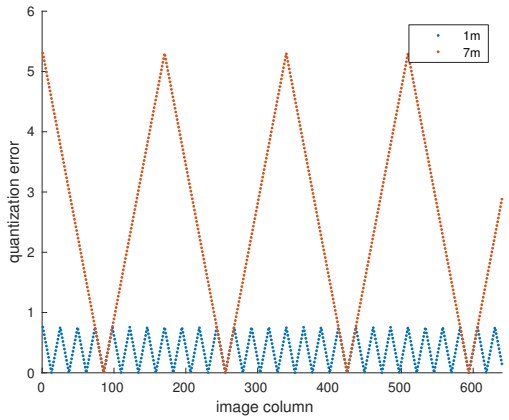
Quantization errors occur when 3D data points are projected from world coordinates into image coordinates. This is done 15 times a second, since this frame rate is required by *fearless*. To keep quantization errors as minimal as possible, points that are already

[2] difference between depth image and continuous depth data



(a) discretization at 1 meter

(b) discretization at 7 meters



(c) quantization error

Figure 4.7: Discretization of continuous depth data along an image row.

in the dynamic background model are not updated. This includes the adaptation of the model to the current frame. The only exception is the ghost detection which is required to remove falsely added objects from the model.

Quantization errors cannot be prevented when the depth data is discretized onto an image. Thus, synthetically created data contains this errors. The other problems do not affect synthetic data except they are particularly implemented. Hence, the results are further used as a baseline for the effect of the other problems.

The quantization error is analyzed by examining the difference between the current frame and the background model of synthetic data. Since no motion occurs in the data, the difference should optimally be zero. However, due to the quantization error the difference must be greater than zero.

4.3.2 Center of Rotation

Optimally, the center of rotation of a movable camera is the location of its *entrance pupil* [68]. This eliminates the parallax error that arises due to a translation of the sensed image. The depth sensor is mounted centrally on the PTU and the sensed image is created from the perspective of the IR camera. This camera is located about $-4.8cm$ off the panning rotation axis. Further, the depth sensor is fixated on a joint that is used for tilting. The joints location is $3.3cm$ beneath the IR sensor. Figure 4.8 illustrates this offsets.

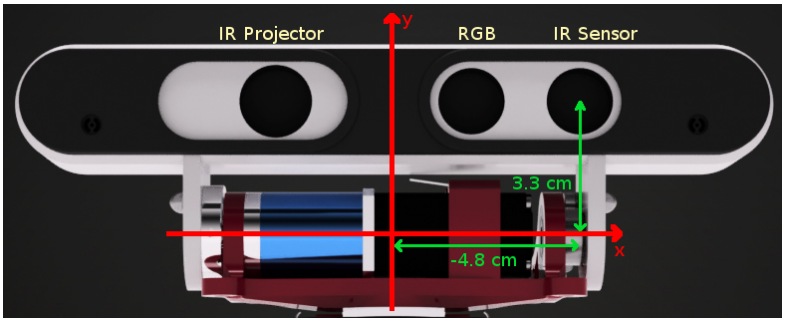


Figure 4.8: Frontal view of the mounted depth sensor on the PTU. The red lines illustrate the rotation axes: y-axis panning, x-axis tilting. The IR sensor does not lie at the center of rotation.

When transforming the depth image from and into world coordinates the center of rotation can be corrected by modifying the extrinsic matrix $\mathbf{E} = \begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}$. This matrix consists of a rotation $\mathbf{R}$ and a translation $\mathbf{t}$, and relates the camera coordinate system with the world coordinate system. The rotation is composed on the PTU pan and tilt and the translation is statically set to the position of the camera $\mathbf{t} = (0, h, 0)^T$, where h is the sensor height. To compensate the shifted rotation center, the offset vector from the rotation center $\mathbf{o} = (-4.8, 3.3, 0)^T$ is rotated and added to the camera position:

$$\mathbf{t} = (0, h, 0)^T + \mathbf{Ro} \tag{4.5}$$

The influence of the parallax error created by this shift is evaluated on synthetic data. The camera is shifted by the offset vector $\mathbf{o}$ from the rotation center to create the same setup as on the PTU. Furthermore, correction is applied to the data (Equation 4.5) and the results are compared to a setup with zero offset from the rotation center.

4.3.3 Synchronization

Images obtained from the depth sensor that is mounted on the PTU arrive after a delay of about 50ms[3]. This delay occurs because the depth images are computed locally on

[3]http://rosindustrial.org/news/2016/1/13/3d-camera-survey Last accessed 02 Dec 2016

the depth sensor. The received depth images are processed using the pan and tilt values from the PTU. Since this values are delivered in the range of microseconds, the latency receiving them can be neglected. Thus, the depth images must be synchronized to the corresponding sensor orientation.

Synchronization can be done by delaying the pan and tilt values from the PTU to match the latency of the depth sensor. Since the latency of the depth sensor is a rough estimate, the impact of small synchronization errors are evaluated. The evaluation is done using equidistant time delays at a fixed panning speed of the unit. This corresponds to a sensor being panned at different speeds with a fixed delay. Furthermore, the correlation between the delay and the sum of errors in the difference maps can be detected.

4.3.4 Lens Distortion

Lens distortion is an optical aberration that makes physically straight lines appear curvy in the image [69]. The distortion is commonly radially symmetric since photographic lenses are symmetric. Two types of radially symmetric distortions are distinguished: (i) barrel distortion and (ii) pincushion distortion. The distortion shape of the former appears as if the image is mapped around a barrel, whereas straight lines in the latter are bowed towards the image center, looking like a pincushion.

The distortion can be modeled by splitting it into a radial $f(r)$ and a tangential $g(r, \alpha)$ part [69]:

$$r^2 = (x - x_p)^2 + (y - y_p)^2 \tag{4.6}$$

$$\hat{r} = f(r)r \tag{4.7}$$

$$\hat{r}\alpha = g(r, \alpha)r\alpha \tag{4.8}$$

x and y refer to a 2D point in pixel coordinates, whereas x_p and y_p refer to the principal point. The radius from a pixel to the principal point is denoted as r and the image must be scaled such that the magnitude of r equals one at the image corners. To compute the corrected radius $\hat{r}$ radial distortion $f(r)$ and tangential distortion $g(r, \alpha)$ must be considered. Experience shows, that the tangential distortion $g(r, \alpha)$ is at least one magnitude lower than the radial distortion $f(r)$. Thus, it is neglected in the context of measuring the impact of lens distortion to the dynamic background model.

The radial distortion function $f(r)$ depends on the lens used and is smaller than 1 for barrel distortion and greater than 1 for pincushion distortion. It is commonly modeled using the polynomial

$$f(r) = 1 + k_1 * r^2 + k_2 * r^4 \tag{4.9}$$

The values of k_1 and k_2 can be found alongside the intrinsic camera parameters using camera calibration. However, instead of performing camera calibration, *fearless* obtains the intrinsic camera parameters from Smisek et al. [59] to be plug and play capable. The accuracy of those values is found sufficient, however, lens distortion is neglected and thus, the conversion into world coordinates suffers increasing inaccuracies at image borders. The influence of this inaccuracies should be estimated by synthetically creating a similar distortion and comparing it to undistorted results.

To obtain the distortion parameters k_1 and k_2 of the depth images, the IR camera is calibrated using a checkerboard pattern on a planar surface in different orientations [70], see Figure 4.9. Since the projected dot pattern disturbs the calibration pattern, the IR projector is covered with a post-it note which blurs the pattern and yields to a illumination effect. Further, the IR image has a size of 1280x1024 pixels, whereas the depth image has a size of 640x480. However, according to Khoshelham et al. [67] a convenient approach to this situation is to estimate the distortion parameters from the reduced IR images.

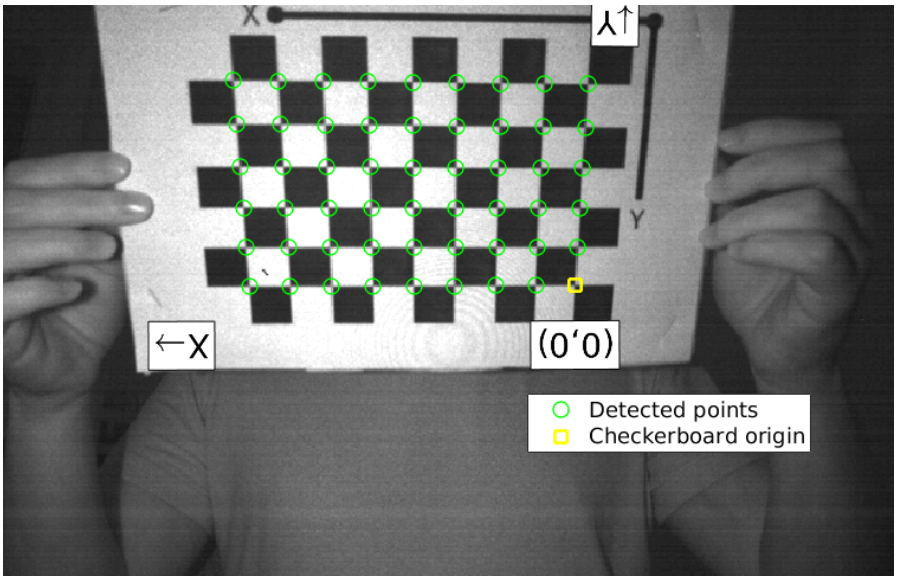


Figure 4.9: Calibrating the IR camera with a checkerboard pattern.

Results on synthetic data that is distorted with $k_1 = -0.026$ and $k_2 = 0.084$ are compared to undistorted data to evaluate the effect of lens distortion. The values for k_1 and k_2 are found as a result of camera calibration.

4.3.5 Depth Accuracy and Resolution

Structured light depth sensors include noise [71]. Khoshelham et al. [67] show that this noise can be modeled with a Gaussian distribution. The standard deviation of this distribution is computed as follows:

$$\sigma_Z = (\frac{m}{fb})Z^2\sigma_{d'} \tag{4.10}$$

The parameters f and b correspond to the focal length and the base line respectively. m is a normalization factor used to convert between the physically correct disparities and the quantized disparities[4]. The disparity measurement error $\sigma_{d'}$ is estimated with $\frac{1}{2}$ pixel.

Since the depth is inversely proportional to the disparity (Equation 3.1), the depth resolution is inversely proportional to the levels of disparity. This resolution can be computed as the difference of two successive levels of disparity d': $\Delta_Z(d') = Z(d') - Z(d'-1)$. Khoshelham et al. [67] show that this equation yields to:

$$\Delta_Z = (\frac{m}{fb})Z^2 \tag{4.11}$$

Figure 4.11 shows the theoretical random error σ_Z and the depth resolution Δ_Z for different depth values. With increasing depth value, the depth resolution and accuracy decreases. The theoretical random error is smaller than the depth resolution, however it should not be understated for higher distances.

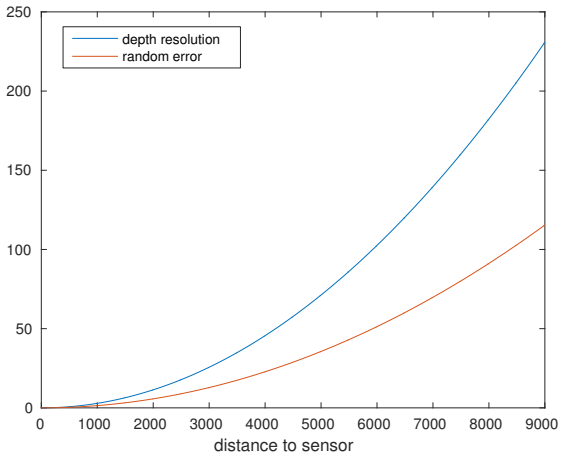


Figure 4.10: The depth resolution Δ_Z (blue) and the theoretical random error σ_Z (red) of a Kinect-like depth sensor.

Figure 4.11 shows a synthetically created scene where gaussian noise with standard deviation σ_Z is added. Further, the quantization of depth values matches the depth resolution Δ_Z. Due to sensor tilt, the quantization layers are slanted. The depth resolution is greater than the random error, however the sensor noise 'pushes' data points into other layers. This can be clearly observed on the flat wall at $9m$. The low accuracy

[4]Kinect-like depth sensors quantize the measured disparities into 1024 levels of disparity.

coupled with the high sensor noise is the reason for *fearless* to stop monitoring regions farther away than $7m$.

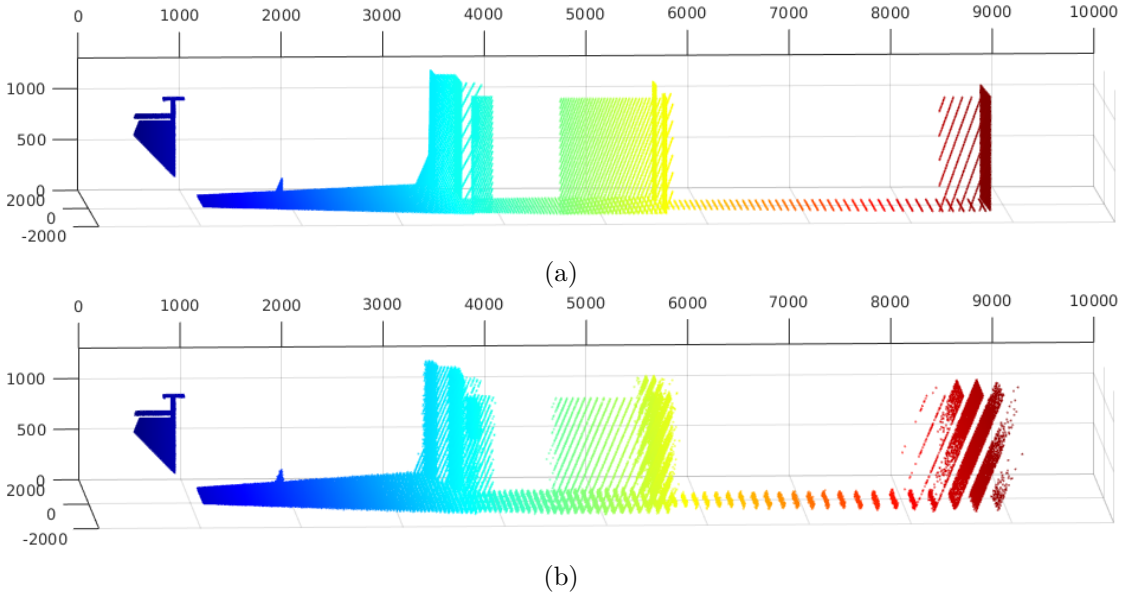


Figure 4.11: A point cloud of a synthetically created scene with (b) and without (a) Kinect-like noise and resolution.

The dynamic motion detector is evaluated on synthetically created input data that is modified as explained previously. The results are compared to the unmodified data to estimate the influence of decreasing depth accuracy and resolution.

4.3.6 Rolling Shutter and Motion Blur

All problems described until now affect static images. However, two effects arise simultaneously when the depth sensor is moved: (i) rolling shutter and (ii) motion blur. Both effects occur if the scene changes during a single exposure, which includes rapidly moving objects. For explanation purpose the two effects are discussed for moving objects, however, the discussion is also valid for sensor location and rotation changes.

Motion Blur

If a sensed object moves during a single exposure of the image sensor, the object appears blurred along the moving direction. This happens since the light of the object is transferred to spatially indistinct pixels. Thus, the blurring effect depends on the exposure time of the image sensor and the moving speed of the object.

A blurred IR image can be corrected using the provided pan and tilt values [72]. However, depth images are created from the blurred IR images already on-chip. Thus, IR images cannot be altered for computation, but correction must be performed on the

depth images. A model for doing so can only be created through experimental results, since the reconstruction process is patented.

Tourani et al. [48] discovered that motion blur causes depth image pixels to take the minimal depth value from those exposed at one time. This leads to an edge-flattening effect causing object boundaries to extend beyond their real boundaries. They use this information to provide a probabilistic labeling of edge-flattened pixels. This labeling can be thresholded resulting in a depth image with a reduced edge-flattening effect.

However, Ringaby et al. [73] show that it is convenient to only consider the rolling shutter effect up to 115°/s. Below this angular velocity the motion blur effect is negligible. Since the PTU moves with a maximal speed of about 75°/s, motion blur is neglected when the dynamic motion detector is tested with synthetic data.

Rolling Shutter

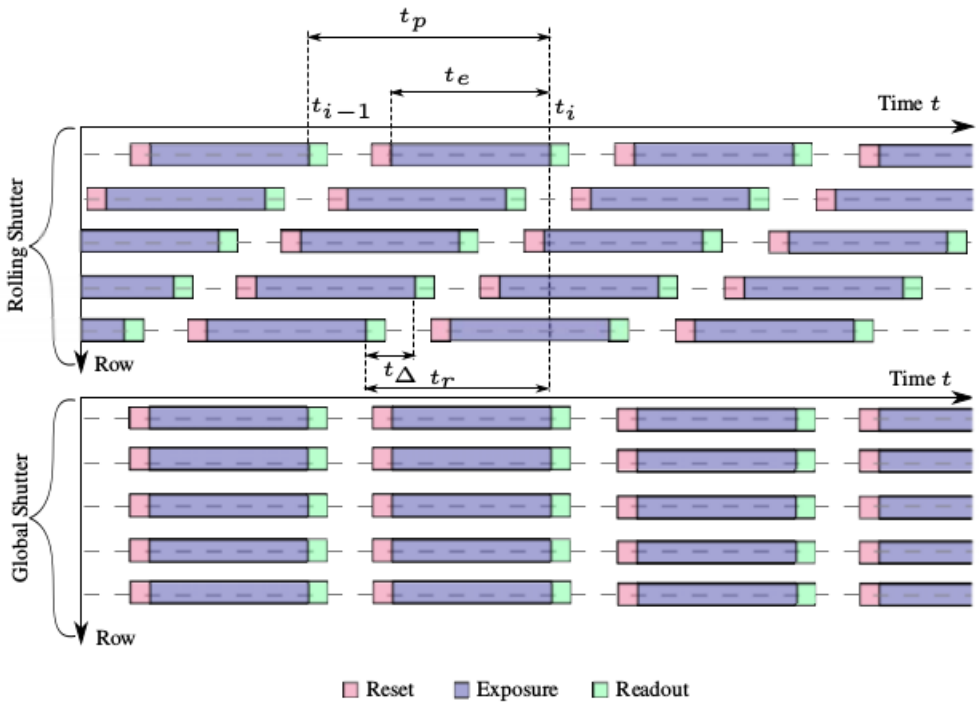


Figure 4.12: Rolling shutter (top) vs global shutter (bottom) from Meilland et al. [72]. The exposure time t_e defines how long an image row is exposed to light. The total readout time t_r measures the delay between the readout of the first and the last image row. The frame period time t_p is the time between two successive readouts of the same row. This defines the framerate of the image sensor.

The depth camera mounted on the PTU uses a CMOS light sensor. The main difference of its counterpart, is namely, the charge-coupled device (CCD), which refers to

the way the analog light signal is read out and digitized. CCD sensors expose all of their pixels at the same time, which is called *global shutter*. Using global shutter mode, the signal of all pixels after exposure must be serially converted to a single analog-digital converter (ADC). Transferring the pixels' signal to the converter is done by shifting them onto a serial bus. This requires a high amount of power and is a bottleneck, since the frame rate of the sensor depends on how fast the signal can be transferred to the ADC.

CMOS sensor use electronic rolling shutters (ERS), where this bottleneck is eliminated since each pixel row is exposed and read-out individually [72]. This requires an ADC for each column, which increases the noise, due to micro variations of the ADCs preamplifier. However, CMOS draw less power than CCD sensors, since the pixels can be read in parallel and must not be shifted onto a serial bus.

An exposure can be started after the previous readout. The whole pixel array is converted one row at a time, which creates a small delay between the readouts of two rows. CMOS sensors optimize their framerate by exposing the $(n+1)$th row after the nth row was converted into a digital signal. Figure 4.12 illustrates this process in contrast to global shutter mode.

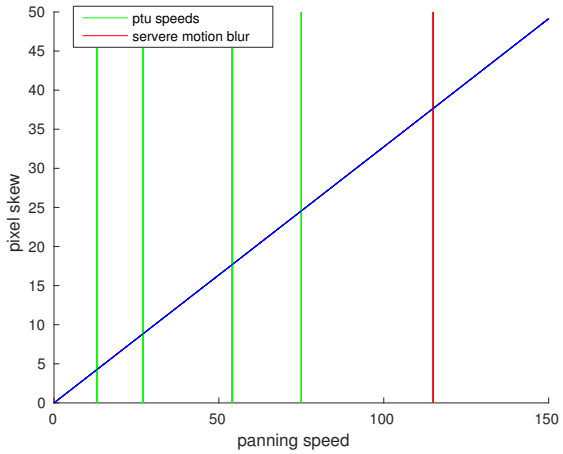


Figure 4.13: Skew of the image in pixels at different panning speeds considering rolling shutter. Severe motion blur starts at 115°/s.

The delay between the readout of the first row and the last row is called *total readout time* t_r and equals to $32.51ms$ for the IR sensor of the PTU [74]. The skew of a moving image is dependent on t_r and can be computed using the relationship that connects the sensor width w, the focal length f and the horizontal field of view $\theta = h_{FoV}$ [75]:

$$\frac{w}{2f} = tan(\frac{\theta}{2}) \tag{4.12}$$

Solving this equation for w and employing the angular velocity $\omega = \frac{\theta}{t_r}$ yields to the following identity:

$$s = 2f * tan(\frac{\omega t_r}{2}) \tag{4.13}$$

, where s is the skew that defines the horizontal shift between the first and last image row. If the focal length f is inserted with pixel units, the skew is in pixel units. Figure 4.13 shows the skew in dependence of the angular sensor velocity and Table 4.2 gives a precise summary of the image skews depending on the angular velocity of the PTU. The Figure highlights that the maximal angular velocity PTU is far off the point where severe motion blur occurs.

steps	speed pan	skew
1+2	75.5967°/s	24.7537px
4	54.0378°/s	17.6930px
8	27.0189°/s	8.8460px
16	13.5095°/s	4.4229px

Table 4.2: Image skew of the depth images with different angular velocities from the PTU.

The dynamic motion detector is evaluated using synthetic data, where the camera is rotated with the same angular velocities as the PTU. The exposure and readout time match the ones found in this section to provide a sophisticated evaluation of the rolling shutter effect. Motion blur, however, is not evaluated, since the effect does not have a severe implications on the data with the operational speeds of the PTU.

4.4 Updating the Region of Interest

fearless computes occupancy grids and occlusion grids for a given ROI that is defined in world coordinates. For efficiency reasons, the ROI should be as small as possible but span the area the sensor can observe. For static sensors, a ROI can be defined that does not change anymore. It is restricted to the observed boundaries by transforming all pixels to world coordinates and fitting a bounding box. The vertical boundaries are fixated to $[-200, 2000]$ to include the full height of a standing person that could enter the scene afterwards. (Due to an imprecise calibration, points may fall 'under' the floor, hence the negative lower boundary.)

This approach is not possible for a moving sensor. Either the ROI has to adjust or span the whole area the sensor can observe. Regarding future work where translations should be supported, the first method is implemented, moving the ROI based on the current sensor position/rotation. A ROI is dynamically constructed by projecting four points into the scene with different distances to the sensor: two at 0.5 meters and two at 6.0 meters. The two farther points are moved along the line connecting them, such that their distance matches 6 meters. This values are all configurable and control the size of the ROI, which has a major influence on the performance. A bounding box is

spanned around these four points to obtain the final ROI. Figure 4.14 shows the ROI of two sensor positions with the projected points. The farther points are connected to the origin and illustrate the alignment of the sensor.

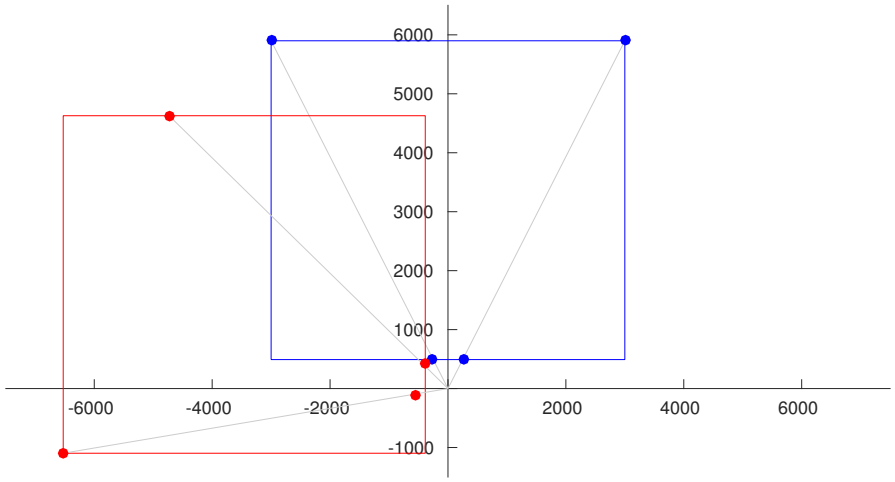


Figure 4.14: Adjustment of the ROI before (blue) and after (red) sensor movement. horizontal: x-axis, vertical: z-axis

Detected persons are represented as a subregion in a given occupancy grid. The offset of the subregion is stored to reconstruct the absolute position in this grid. Moving the ROI implies that these offsets become invalid and must be corrected. For instance, if a stationary person is located at the center of the ROI, panning the sensor to the left results in the person moving to the right. Thus, the offset of the person subregion must be updated accordingly to match the correct position in the new ROI.

Occupancy grids are created in each frame and are thus unaffected by sensor changes. However, creating occlusion grids is computationally expensive, thus they are incrementally updated in *fearless*. When changing the sensor rotation, however, occlusion grids must be partially created. Voxels in the occlusion map where the old and the new ROI overlap can be reused. Yet, this approach only works on rotations, not for translations. Considering future work where this method should work with translations will have to fully recreate the occlusion map in each iteration or find a better solution.

4.4.1 Tracking

All this changes enable fall detection while panning and tilting the sensor. However, the PTU must follow a person to keep it inside its FOV. This is done by projecting the center of the detected person to the two dimensional image plane. The difference of the projected point to the image center is used to control the PTU. The higher this difference is the faster the unit is rotating to the person. It is highlighted that the persons moving direction is not taken into account, but future work will focus on that to minimize the units tracking latency.

The fall detector is evaluated using the PTU following a person that walks along different trajectories and falls at different distances to the sensor.

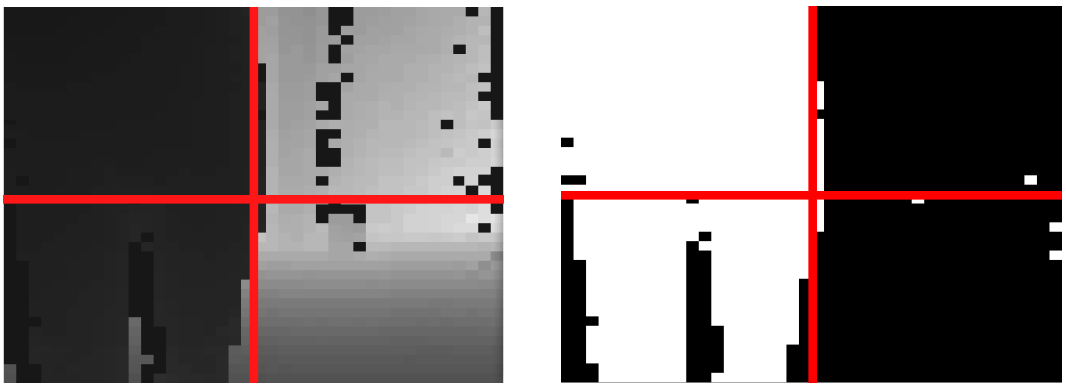
4.5 Detection of Uncontrolled Sensor Movements

Uncontrolled sensor location changes (e.g. sensor tampering) can occur due to persons, intentionally or unintentionally moving the sensor. This changes yield to an abnormal state in the fall detection system and must therefore be detected. Therefore, the following method is proposed since *fearless* lacks this detection. It can be used in a static setup, using only *fearless*, or in a setup with moving sensor support.

When the sensor is tampered with, the background model will become invalid. However, this results in significant changes of the absolute difference between the model and the current frame. Thus, this changes can be detected by monitoring the amount of motion over time. The motion detector used is based on that, which is used by *fearless*.

The *fearless* motion detector only detects motion, if a data point is significantly closer to the sensor as the model. While, if it is significantly farther away the pixel in the background model is classified as a ghost and no motion is detected. Ghosts are removed periodically, however, this behavior is not beneficial, since a tampered sensor should optimally increase the motion of the whole image, independently from data points situated in front or behind the background model. Thus, the *fearless* motion detector is adapted to classify motion neglecting the sign of the difference between background model and current frame. Further, the ghost detection is removed.

If the motion exceeds a threshold, the system goes into a «tampering» state. To exit this state the sensor must not be tampered with for a period of time. Since the background model is invalid after sensor movement, frame differencing is instead used for motion detection during the «tampering» state.



(a) Depth Frame (b) Motion Mask

Figure 4.15: Depth frame and corresponding motion mask of a person occupying two quadrants. The red lines illustrate the four quadrants.

This approach works well if the sensor is mounted higher than 2 meters, but problems arise if persons move closer to the sensor. For example, a close person would result in a motion peak that might be detected as tampering. To increase the robustness, the motion image is split equally into four quadrants. At least three quadrants must exceed a shared threshold to enter the «tampering» state. Figure 4.15 illustrates this approach.

Furthermore, close points cannot be measured by the sensor and result in a black pixel on the image. This knowledge is applied for tampering detection: if at least two quadrants are fully occupied by black pixels, which indicates that the sensor points towards a close object, tampering is detected.

The tampering detection approach can be used in conjunction with the proposed fall detection system. This protects the fall detector from processing faulty data that results after the depth sensor is intentionally or unintentionally manipulated. However, the fall detection system suffers inaccuracies that arise from real depth data. Inaccuracies can be corrected to reduce the errors, but they are the reason falls can only be detected to a limited sensor distance.

CHAPTER 5

Implementation

All contributions made to *fearless* are divided into several loosely coupled *components*. This facilitates its maintenance and allows for it to be used in different programming languages. Therefore, this chapter describes how each component is implemented and how it interacts with other components. The first section gives an overview of how the fall detection system is distributed in general. Also, an overview of all the components is given and explanation of their communication process is provided. The following sections describe the purpose and implementation of each component in detail.For clarity, each component's name is typed in bold letters.

5.1 System Distribution

Figure 5.1 shows a component diagram illustrating those components and their relationships. The *fearless* implementation with pan/tilt support, is summarized into one component, namely **Pan Tilt Fearless**. This component is the core component of the fall detection system and generates events that are forwarded to the **Event Manager** component. The event manager sends the events to a variable number of **Adapters** that share the same interface. Each adapter implements an interface to a third-party platform that should receive events from the fall detection system. Those platforms can take the form of an email or short message service (SMS) server, but also alarm monitoring platforms.

The input data is obtained from the **Depth Provider** component. The depth provider supplies the **Tampering Detection** component with the same input data. This component performs the detection of uncontrolled sensor location changes and will pause fall detection while tampering is detected. The **Pan Tilt Fearless** component act therefore as a server and provides a start/stop interface. The **Remote Control** component uses this interface to enable start/stop functionality for the user.

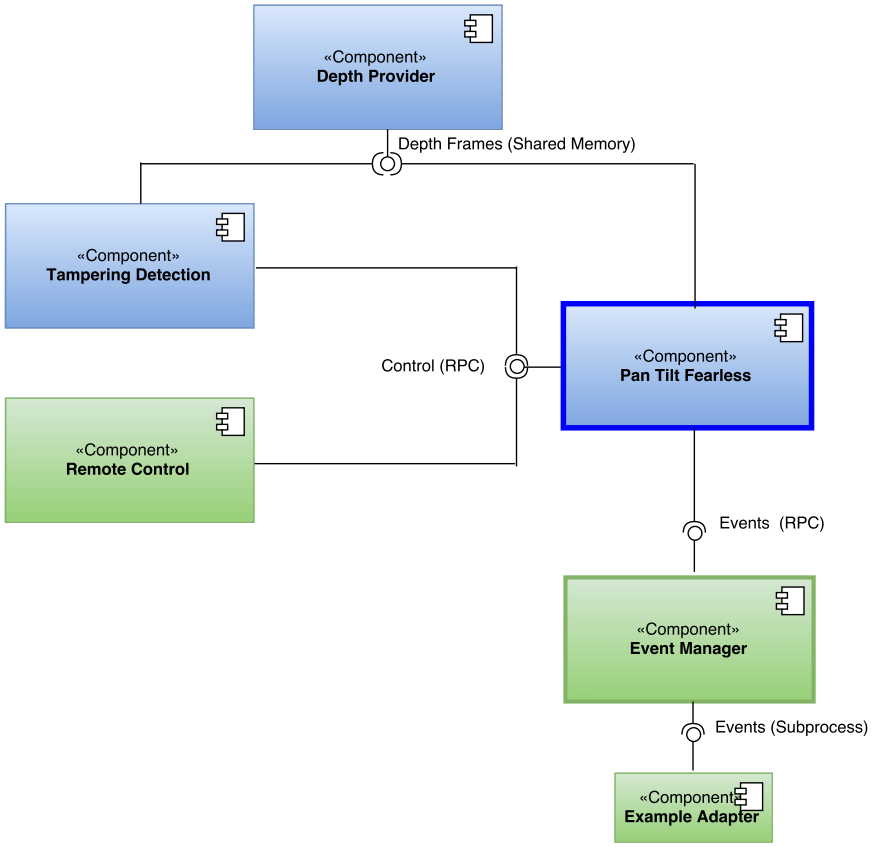


Figure 5.1: System distribution of *fearless* with pan/tilt support. Blue components are written in C++, green components are written in Python. **Pan Tilt Fearless** is the core component of the fall detection system that generates fall events.

5.1.1 Apache Thrift

Apache Thrift[1], in short *thrift*, is a remote procedure call (RPC) framework that is used to communicate between the components, see Figure 5.1. Its implementation is described in a technical paper published by Facebook in 2007 [76] and is now hosted by Apache.

A service interface is defined by a `.thrift` file. This file can be used to generate code for different programming languages with the thrift compiler. The generated files are imported into components to act as a client or a server. The code overhead is minimal allowing a fast setup, which is the reason that thrift is chosen as a communication framework.

Thrift servers must define a host and port where the RPC service is bound to. Since all components of the fall detection system run on the same system, the host is always the local host. However, in theory, components could be moved to other hosts which

[1] https://thrift.apache.org/ Last accessed 02 Dec 2016

advises to secure the connection with secure sockets layer (SSL) encryption, which is supported by thrift.

5.2 Depth Provider

The **Depth Provider** writes the depth images into shared memory, allowing an arbitrary number of components to read them. This communication is denoted as inter process communication (IPC) and implemented using *Boost Interprocess*[2]. The **Tampering Detection** and **Pan Tilt Fearless** components use the images from the **Depth Provider** as their input data. Not only the depth images are provided, but also meta data:

- **id**. The frame id.
- **timestamp**. The frame timestamp.
- **width, height**. The frame width and height.
- **framerate**. The framerate at which depth images are read.
- **running**. Whether frames are currently read.
- **repeat**. Whether frames are repeated sequentially.
- **number of frames**. The number of frames in case of a recorded sequence.

The meta data is important since clients know how much frames to expect. Also, the read frequency is defined by the meta data and whether the sensor stopped providing frames due to errors. If the **Depth Provider** gets its data from a recorded file the frames are limited. Thus, the *number of frames* is set to a finite number and the *repeat* flag can be optionally set. Those parameters are redundant when reading from a live depth stream.

The **Depth Provider** reads the depth data using OpenNI2[3], which is a open source framework that provides an application programming interface (API) to communicate with depth sensors. OpenNI2 supports the ASUS Xtion sensor mounted on the PTU, but also the Microsoft Kinect and the Orbbec Astra.

5.3 Event Manager

The **Event Manager** acts as a server for **Pan Tilt Fearless** and receives and manages events from this component. The following list gives an overview of the supported events:

- **status changed**. The internal status of **Pan Tilt Fearless** changed: stopped, init, running, errors.

[2]http://www.boost.org/doc/libs/develop/doc/html/interprocess.html Last accessed 02 Dec 2016
[3]http://structure.io/openni Last accessed 02 Dec 2016

- **init started**. Initialization started. Includes a visualization of an unprocessed depth image.
- **init completed**. Initialization completed. Includes a visualization of the scene.
- **person event**. Fall or recovery. Includes a visualization and confidences.
- **person detected**. Person is detected. Includes the track id.
- **person lost**. Person is lost. Includes the track id.

It is highlighted that the communication between **Pan Tilt Fearless** and the **Event Manager** happens through RPC even though the components use different programming languages. The former is implemented in C++ and the latter is implemented in Python.

5.3.1 Queing

Events are sent to an arbitrary number of adapters, which are responsible to forward them to third-party platforms. However, if a platform is not reachable (i.e. network problems) the adapter raises an error. Therefore, a queuing system is implemented in the **Event Manager**.

A queued event has a maximum lifetime of 3 days. This ensures that the queue size does not exceed local hardware resources. Also, experience shows that network connections can drop for multiple days. This makes it important to spare the network bandwidth by processing the queue every 60 seconds.

Another important feature of the queue is being persistent. This is done by serializing it into a `.pickle` file. Thus, the queue is not cleared if the fall detection system restarts, the whole system reboots, even if the **Event Manager** is uninstalled.

5.3.2 User Feedback

Besides sending events to adapters the events are used to control the LED of the PTU. This gives the user instant feedback of the system state. Status updates could be for example the fall detection system being stopped, starting, running or going in an error state. The LED blinks if a fall or recovery event is received. Such events have a timeout that defines how long the LED is blinking until it falls back to its last color. The communication to the PTU is accomplished with the pyserial[4] library.

5.4 Adapters

An **Adapter** implements an interface to a third-party platform. Those platforms can be email servers, SMS servers, monitoring platforms, et cetera. Each adapter acts as a server to the **Event Manager** and shares the same interface. If the interface changes,

[4]https://pythonhosted.org/pyserial/ Last accessed 02 Dec 2016

every adapter implemented must be updated accordingly. However, the advantage of using a shared interface is that new **Adapters** can easily be 'plugged' into the **Event Manager**. This is realized by collecting all active adapters in a configuration file. A new event spawns a subprocess for each adapter in this file. The subprocesses are called with the same arguments, since they share the same interface.

5.5 Pan Tilt Fearless

Pan Tilt Fearless depicts the core component of the fall detection system. It comprises the implementation of *fearless* including the contributions to enable fall detection during pan tilt. The depth data is obtained from the **Depth Provider** component and events are sent to the **Event Manager** component. Additionally, a thrift service is provided to enable and disable the detection of events. This is used by the **Tampering Detection** component to stop the fall detection during sensor movement. Also, the **Remote Control** component gives the user the option to manually disable fall detection using this service.

5.5.1 Fearless

The implementation is split into 5 libraries that implement the algorithmic pipeline of *fearless*. Table 5.1 gives an overview with the corresponding dependencies. All libraries use OpenCV 3[5] for image processing. Further, TBB 4[6] is used (except in *depthio*) to enable processing in multiple CPU cores.

Library	Description	TBB 4	OpenCV 3	Other
depthio	Depth data input/output		x	
autocalib	Automatic calibration	x	x	
dmd	Depth data motion detection	x	x	
c3d	3D processing	x	x	
fearless3-core	Fall detection logic	x	x	x

Table 5.1: Dependency overview of the *fearless* implementation.

The *depthio* library had to be updated to enable reading from the **Depth Provider** using IPC. *fearless3-core* uses a number of other dependencies, since it takes care of high-level logic and uses functionality from other libraries:

- **Boost 1.54**[7]. Reading/parsing configuration files and runtime arguments, as well as manipulating files, directories and paths that identify them.
- **RapidJSON**[8]. Reading and parsing classifier models.

[5]http://opencv.org/ Last accessed 02 Dec 2016
[6]https://www.threadingbuildingblocks.org/ Last accessed 02 Dec 2016
[7]http://www.boost.org/ Last accessed 02 Dec 2016
[8]http://rapidjson.org/ Last accessed 02 Dec 2016

- **SpdLog**[9]. Fast logging library.
- *autocalib*. Filtering object normals to detect walls used to restrict static objects.
- *c3d*. Occupancy and occlusion grid operations for fall detection.

All libraries are built as shared libraries (`.so` files on Linux) using the standard CMake[10] installation pipeline. The libraries are dynamically linked when building a fall detection application, hence, this type of library is applied in this implementation.

5.5.2 Pan Tilt Support

A fall detection application is developed with pan/tilt support that uses the libraries from *fearless*. Dividing *fearless* into multiple libraries allows for the modification of intermediate results. The implementation of the dynamic motion detector, for example, reuses the *dmd* library at a per-frame basis to compute the motion mask that is forwarded to the fall detection logic in *fearless3-core*.

Dynamic Motion Detector

The dynamic motion detector is implemented with a threshold starting at $0.3m$ and increasing linearly with the distance to the sensor. However, an individual in a lying position is about $0.3m$ above the ground, making fall detection impossible, since the motion mask would not include a person. Thus, a second, more sensitive dynamic motion detector is utilized for floor regions. This motion detector is more error-prone due to its lower threshold, however, it is tuned such that a persons in a lying position can be robustly detected up to $5m$.

fearless only converts points into world coordinates that are classified as motion. The dynamic motion detector, however, must convert the whole depth image. Thus, it forms a bottleneck of the whole system making it infeasible to run on a single-board computer. To overcome this limitation, the input images are scaled down with nearest-neighbor interpolation [77] to half of their size. This reduces the number of points based on VGA resolution from 307200 to 76800 (one fourth).

The conversion from and to world coordinates is done using the pan and tilt values from the PTU. However, these values are relative to an origin that varies depending on the initialization of the PTU. For this reason, software calibration is performed using the *autocalib* library to obtain the initial groundplane and consequently the pan and tilt values.

[9] https://github.com/gabime/spdlog Last accessed 02 Dec 2016
[10] https://cmake.org/ Last accessed 02 Dec 2016

5.6 Tampering Detection

The implementation of detecting uncontrolled sensor movements is summarized in the **Tampering Detection** component. As an input it receives the same depth frames as **Pan Tilt Fearless** via IPC from the **Depth Provider**. Instead of detecting persons the motion of the whole scene is used as a basis for the tampering detection. Thus, the input images are scaled down to 40x40 pixels. This yields to a lightweight system that runs parallel to **Pan Tilt Fearless** without consuming much hardware resources.

When tampering is detected, the thrift service of **Pan Tilt Fearless** is used to stop fall detection. After sensor tampering the fall detection application is reinitialized and started.

5.7 Remote Control

The proposed system can be started or stopped using a remote control such as the *IR Remote Controller* from Hardkernel[11] (Figure 5.2). The decoding of the IR signals is accomplished using the linux infrared remote control (LIRC)[12]. However, an IR receiver must be available on the local device to receive the IR signals (e.g. ODROID-C1+). The following buttons on the remote control initiate an action on the fall detection system when pushed:

- **power** toggle
- **vol+** start
- **vol-** stop

Figure 5.2: Remote used to control the fall detection system.

The component **Remote Control** is a Python application that implements the IR communication and offers callbacks to start, stop and toggle **Pan Tilt Fearless**. It is highlighted that only the core component and therefore the internal status of the system

[11] http://www.hardkernel.com/main/products/prdt_info.php?g_code=G141637537661 Last accessed 02 Dec 2016

[12] http://www.lirc.org/ Last accessed 02 Dec 2016

is stopped and started. This has the advantage of being able to restart the fall detection system faster.

The **Remote Control** is one example of an extension to the fall detection system. Other extensions can be added by connecting to one of the components. Thus, the implementation is a scaffold that can be easily extended.

CHAPTER 6

Results and Discussion

This chapter contains the experiments and evaluation results of (i) the dynamic motion detector (Section 6.1), (ii) the fall detection system with pan and tilt support (Section 6.2) and (iii) the detection of uncontrolled sensor movements (Section 6.3).

The dynamic motion detector is evaluated using synthetically created data that is artificially distorted to mimic problems that arise with real data. Reference results are obtained using non-distorted synthetic data and are compared to the distorted data to evaluate the influence of those problems. At the end of the section the synthetic data is distorted with a combination of all problems which is then compared to the real data.

The fall detection system with pan and tilt support is evaluated with real data that is obtained from the PTU. The evaluation contains quantitative results from two different sensor heights and falls with distances of 2, 3 and 4 meters from the sensor. Additionally, the movements of the PTU are determined by three trajectories of the test subject which classify the amount of sensor movement into three categories: **heavy**, **middle** and **low**. In addition a fourth class, namely **no movement**, is introduced to compare the categories against a static setup. This enables the evaluation of the amount of sensor movement to the fall detection performance.

The detection of uncontrolled sensor movements is a contribution to a static *fearless* setup and thus, evaluated with static sequences. This is quantitatively done using a dataset that comprises 1815 sequences.

6.1 Dynamic Motion Detection

All results in this section are generated based on data recorded in the same scene, see Figure 6.1. All sequences are recorded using the same camera movements and pan/tilt speeds. The only differences between the sequences are the added distortions to mimic problems with real sensors.

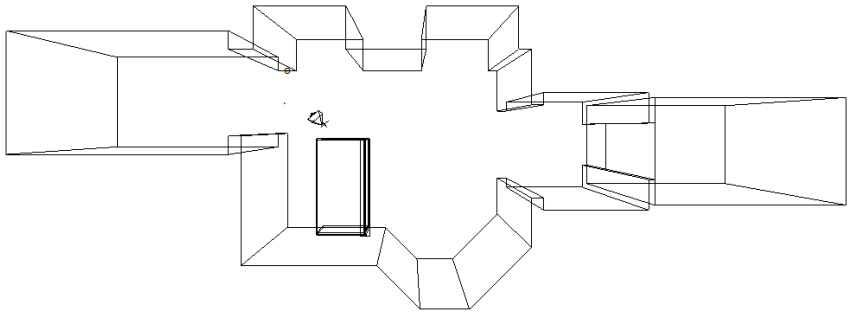
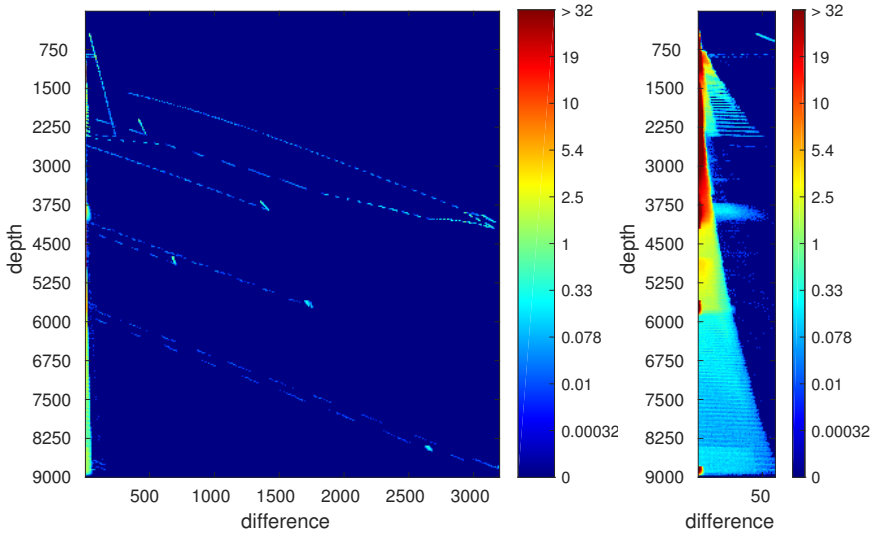


Figure 6.1: Top view of the synthetically created scene. The camera is positioned near a desk and pans 180° in 200 frames.

6.1.1 Pixel Discretization

Synthetic data without modification is used to generate results that are used as a reference for all experiments. The only inaccuracies of this data arise from quantization errors when projecting the depth signal into image coordinates and discretizing it into pixels.



(a) (b)

Figure 6.2: Difference maps of a synthetically generated sequence. Image (a) shows all differences, where image (b) focuses on small differences.

Figure 6.2 shows two difference maps for a synthetically generated sequence. High differences ($> 0.5m$) occur on object edges. This edges can be seen in the left difference map as stripes, further denoted as *edge-stripes*. They start at zero difference, since the object edge meets the ground plane. Moving upwards, along an edge, the difference increases until the edge occludes a wall that is perpendicular to the sensor view direction.

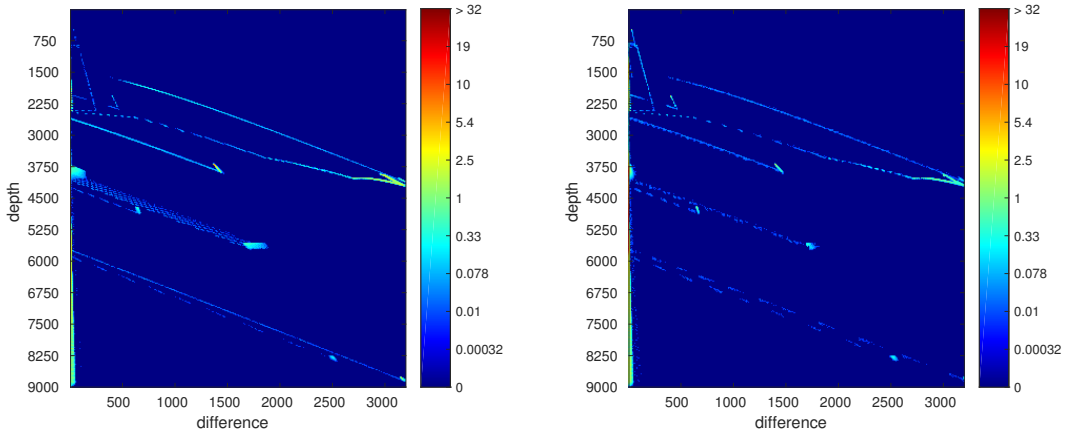
At the wall, differences remain constant and leave a dense area in the map. Since the depth values of the points are measured in camera coordinates a flat wall that faces the depth sensor appears tilted. Thus, the edge-stripes are skewed.

Differences below $50mm$ are illustrated in the right image of Figure 6.2. The quantization error of the ground plane can be clearly seen in form of a cone starting from $0.5m$ and ending at $8.5m$. The spacing between the depth layers of the cone occurs since the area occupied by one pixel increases with the depth. However, another important factor which should be noted is the slope of the surface. The synthetic scene contains a desktop surface that is positioned $1m$ above the ground. Thus, the depth sensor looks in a shallow angle onto the surface yielding to rapid depth changes in the depth image. As a result the quantization error increases significantly. It can therefore be concluded that a high sensor position is beneficial for quantization errors on the ground plane (and other parallel planes).

The two dense areas around $5.7m$ and $8.8m$ with differences below $3mm$ depict walls that are parallel to the image plane. The angle the of the camera ray falling on the surface is 90°, where quantization errors reach a minimum.

Around $3.8m$ a wall is located that is orthogonal to the image plane and additionally close to the sensors x-position. This, again creates a shallow angle onto the surfaces resulting in a high quantization error, which appears as a cloud-like structure in the difference map.

6.1.2 Center of Rotation



(a) Uncorrected (b) Corrected

Figure 6.3: Difference maps of a synthetic sequence with the camera not lying on the center of rotation. Image (a) shows the consequences of this shift, where image (b) is generated with regards to the shift in the dynamic motion detector.

To mimic the depth sensor mounted on the PTU, the camera of the synthetic scene is

shifted from the rotation center. A difference map is generated using this modified input data. Further, another difference map is obtained correcting the shift in the dynamic motion detector.

Figure 6.3 shows the two difference maps with and without correction. The uncorrected difference map displays higher errors on object edges. Also, the edge-stripe starting at $3.8m$ increases in size. This edge-stripe corresponds to a wall where its surface normal is perpendicular to the viewing direction of the sensor. The difference map suggests that errors grow rapidly on such surfaces.

The difference map of the corrected rotation center depicts a significant loss in errors. However, a comparison with the reference map is difficult, since they look identical. Thus, Figure 6.4 shows the sum of differences of the uncorrected, corrected and reference difference map. It can be observed that the correction of the rotation center is a great benefit to the algorithm. The errors are still greater than using reference data by a factor of 1.5. This can be explained by the occlusions that arise if the depth sensor is shifted, similar to the IR sensor measuring the projected pattern.

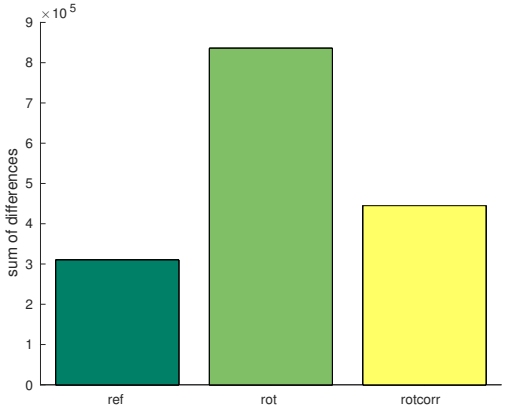
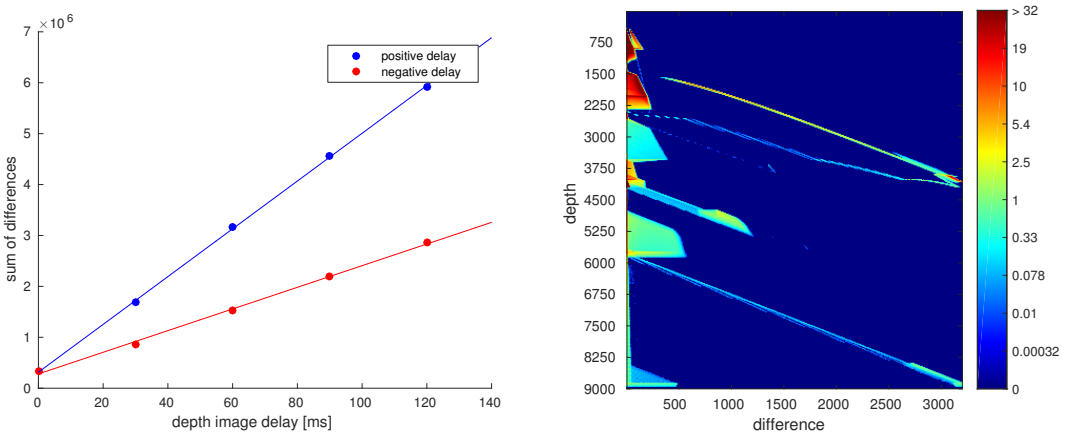


Figure 6.4: Counted errors of the difference maps using reference data (ref) and data with a shifted rotation center (rot) that is corrected (rotcorr) by the dynamic motion detector.

6.1.3 Synchronization

The delay of the depth images obtained from the PTU is approximately $133ms$. The camera is set with a panning speed of 15° per second which corresponds to the panning speed of the PTU using a step division of 16. Figure 6.5 shows the difference map of synthetic data using this delay. The edge-stripes illustrate that the frequency of errors on object edges increases significantly. Also, walls that are aligned orthogonally to the image plane draw huge error-areas into the difference map. The reason for this areas is a error-prone conversion from depth points into world coordinates, since the extrinsic matrix is generated from pan and tilt values that are not synchronized to the depth images. The distance of each data point in world coordinates in comparison to its true

location is constant. However, since a difference map measures the distance to the sensor, the direction where a 3D point is translated is crucial. Translations approximating the sensor, create higher errors because of translations along the perimeter of the sensor. Thus, walls that are orthogonal to the image plane are more vulnerable to inaccurate synchronization than walls that are parallel to the image plane. Moreover, the difference map emphasizes that the extend of such errors is proportional to the depth.



(a) (b)

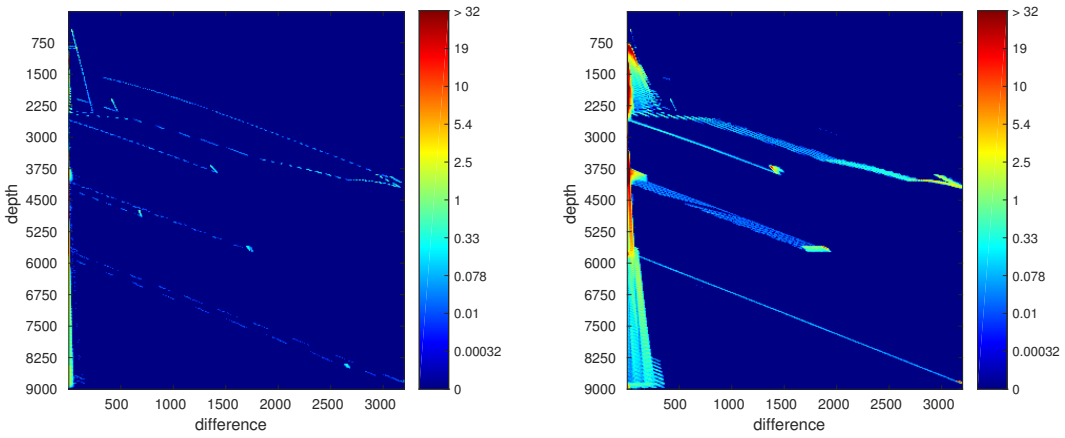
Figure 6.5: Image (a) shows the correlation between the depth image delay and the sum of differences obtained from the difference maps. The difference map of unsynchronized data from the PTU has a delay of $133ms$ and is displayed in Image (b).

A positive delay corresponds to the depth images being delayed to their pan/tilt values, where a negative delay means that depth images are in advance to their pan/tilt values. The difference map only includes positive differences. Thus, it is decisive whether the delay is positive or negative, since the sign of depth differences of a wall surface flips with the sign of the delay. The left image in Figure 6.5 shows that a negative delay would be beneficial for this specific sequence. However, in general no assumption can be made whether a positive or negative delay produces smaller errors.

The plot illustrates the correlation between the depth image delay and the sum of differences. A linear function is fitted using least squares regression [78] which starts at zero delay and corresponds to the reference results. For the sequence used the error is doubled at a delay of $13.2ms$, respectively $-6.7ms$. This highlights the importance of a precise synchronization.

6.1.4 Lens Distortion

To synthetically mimic a real lens, the distortion effect of such is estimated using camera calibration. The distortion parameters are applied to the depth images as a post processing effect with *Blender* before using them as input data to the dynamic motion detector.



(a) (b)

Figure 6.6: Comparison of the reference difference map (a) to a difference map with lens distortion (b).

Figure 6.6 shows a comparison of the resulting difference map to the reference map. The increase in errors can clearly be observed. Especially object edges are vulnerable to such inaccuracies, but also slanted surfaces like the ground plane yield to errors up to $0.5m$.

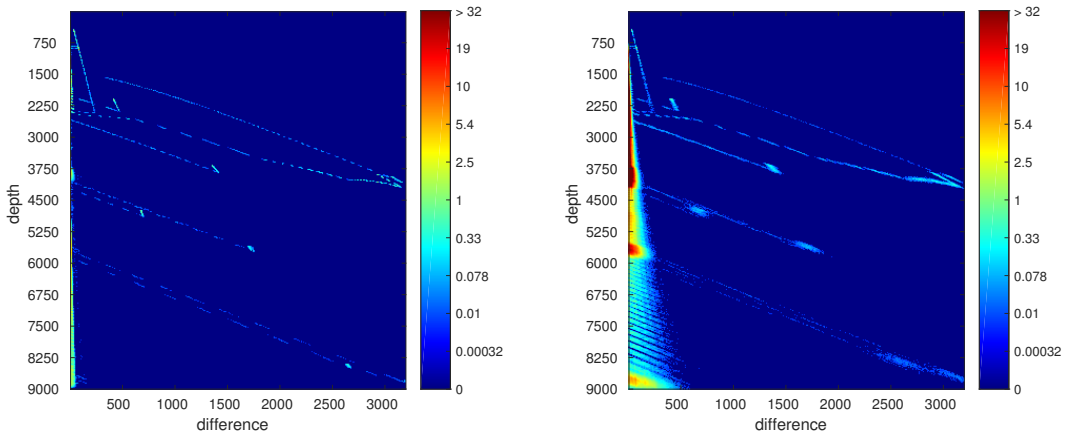
The sum of all differences with lens distortion is **9** times higher than the reference results. This highlights the sensitivity to an error-prone conversion of depth images into world coordinates.

6.1.5 Depth Accuracy and Resolution

Since real depth sensors suffer decreasing accuracy with increasing depth, Gaussian noise is added to the synthetic depth images. Additionally, to match realistic depth resolution, the depth values are quantized into depth bins with size proportional to the depth value.

Figure 6.7 illustrates the difference map created from the generated image sequence. The sum of differences is **4.7** times higher as the reference results. This indicates less sensitivity of the dynamic motion detector as compared to lens distortion. However, depth accuracy and resolution cannot be corrected and depend on the type of depth sensor used whereas lens distortion can be corrected using camera calibration.

When comparing the difference maps, the added noise can be clearly observed in form of dispersed shapes. The noise yields to a massive increase in errors (up to $0.5m$ at $9m$ distance) at distances. The ground plane forms the shape of a cone and spans nearly the whole spectrum of depth values $[0.75m, 9m]$, illustrating the increase in inaccuracies with distance. Flat walls that are parallel to the image plane create significant errors, due to the noise. Without noise, they would only create differences < 3. This error areas appear as smeared ellipses i.e. at $5.5m$ and $8.75m$.



(a) (b)

Figure 6.7: Comparison of the reference difference map (a) to a difference map with realistic depth accuracy and resolution (b).

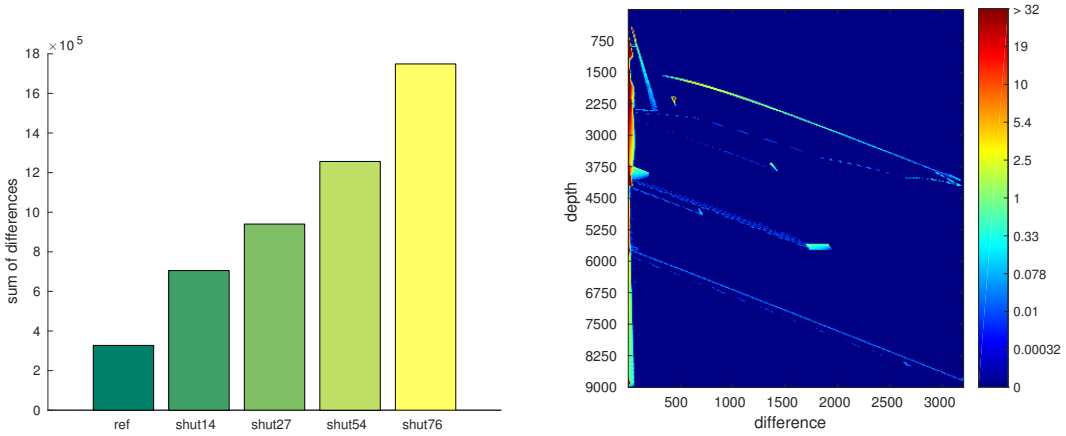
The decreased depth resolution can be seen at the ground plane cone. Starting at $6m$ the cone appears layered. In between those layers no depth points are located due to the quantized data. This error adds up with the added noise and depends on how far off the depth point is from the resolution bin center. At $9m$ the points are about $240mm$ apart and thus the maximal error is $120mm$ if a point falls in between two depth bins. This error contributes to the noise that is added to the depth points.

6.1.6 Rolling Shutter

The results of adding the rolling shutter effect to the synthetically created data is shown in Figure 6.8. The overall error already doubles at the slowest angular velocity 14°/s. However, the difference map is created at 76°/s and suggests that only errors at low distances show a significant increase, even though the overall error is **5.3** times higher at this speed. This is owing to the rolling shutter effect which creates a horizontal shift that linearly increases with the image row. Thus, the error is zero at the first row and maximal (i.e. 24.75px at 76°/s, see Table 4.2) at the last image row. Since the depth sensor is tilted downwards, pixels that are located in the upper part of the image correspond to points that have a higher distance to the sensor. Therefore, those distances do not suffer a high rolling shutter error, whereas pixels of the lower part of the image suffer increasing errors.

Further, the rolling shutter induced skew only changes if the sensor accelerates/decelerates. Thus, a depth sequence with constant speed and no tilt changes results in reduced rolling shutter errors, since the depth points are shifted equally.

When comparing the difference map with the rolling shutter effect (Figure 6.8) to the reference map (Figure 6.7) it can be seen that the rolling shutter effect does not add

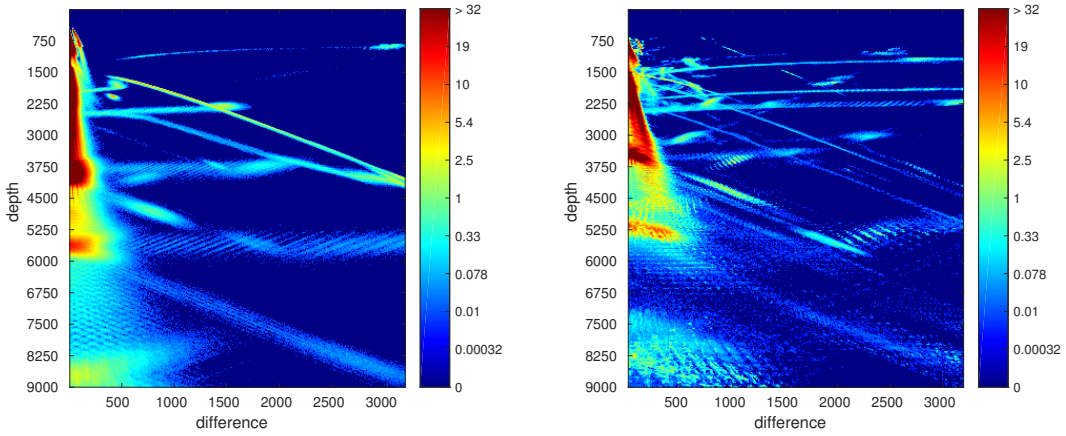


(a) (b)

Figure 6.8: The influence of the rolling shutter effect to synthetically created data. The bar chart (a) shows the overall error at angular velocities of 14°/s (shut14), 27°/s (shut27), 54°/s (shut54) and 76°/s (shut76), which correspond to the operating speeds of the PTU. The reference bar (ref) shows the errors with no rolling shutter effect. The difference map created at an angular velocity of 76°/s can be seen in image (b).

significant errors to points above 4.5m distance. Thus, the sensor tilt is highly beneficial for the dynamic motion detector, since errors have higher influence on points at high distance. When performing fall detection, a tilt downwards the floor is common, since the depth sensor has to monitor the floor.

6.1.7 Real Data



(a) (b)

Figure 6.9: Synthetic data (a) vs real data (b) obtained from the PTU.

Real data is recorded using the setup that is modeled synthetically, see Figure 6.1. The synthetic data combines all problems from this section. This yields to two difference maps that are depicted in Figure 6.9. Both maps are similar but lack matching details, since the synthetic data is only a coarse model.

Lens distortion has the highest impact on errors in the dynamic motion model. Assuming a perfect synchronization between depth images and pan/tilt values, inaccurate depth values in conjunction with decreasing depth resolution and rolling shutter comes second.

It should be highlighted that the theoretical random error σ_Z has to be multiplied with a factor of three to obtain similar inaccuracies. Another error model is tested [79] to legitimize the error model used from Khoshelham et al. [67]. However, similar results are obtained. Thus, it can be concluded that an unknown error source exists, which adds significant errors to the dynamic motion detector. This makes inaccurate depth values the leading cause of errors. Future work will focus on finding this error source and, if possible, correcting it.

6.2 Fall Detection

The fall detection system with pan/tilt support is evaluated using the PTU at 27°/s. This speed is chosen since it is found to be sufficient for the unit to follow an elderly person. The falls are simulated at sensor distances of $2m$, $3m$ and $4m$ at different trajectories, see Figure 6.10. Each trajectory refers to a constant amount of sensor movement, whereas 180°, 50° and 20° are denoted as **heavy**, **middle** and **low movement**, respectively. **No sensor movement** is analyzed to obtain reference results, which correspond to an unmodified version of the *fearless* fall detection system. Furthermore, the influence of the sensor height is evaluated by placing the PTU at $1.15m$ and $2.15m$ above the ground.

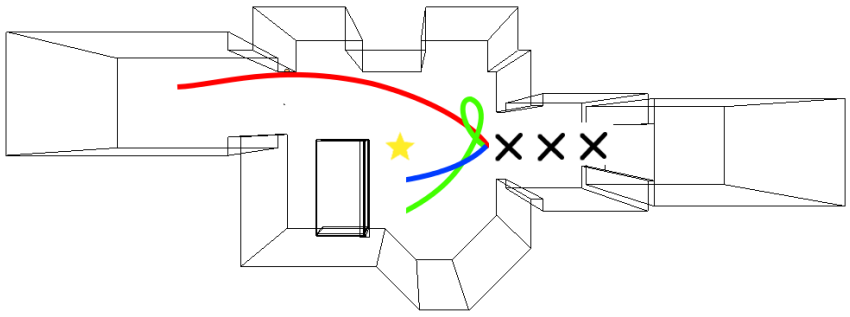


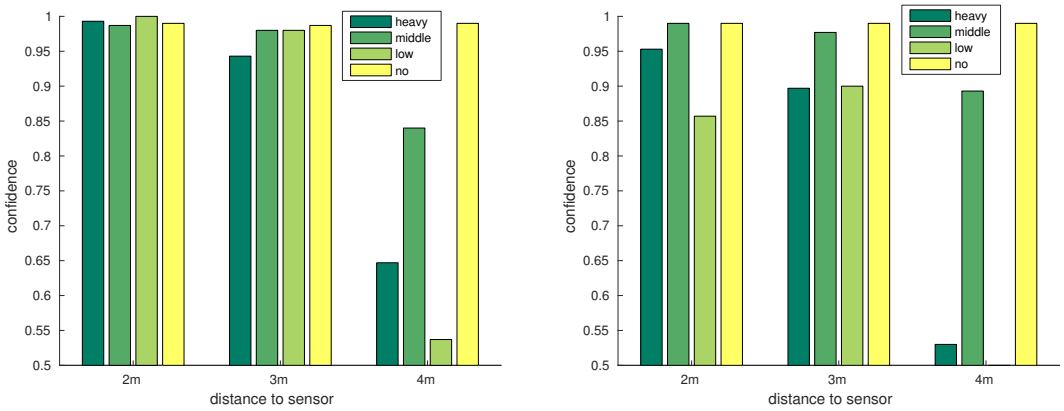
Figure 6.10: Setup used to evaluate the fall detection system. The depth sensor is illustrated with a yellow star and the black crosses mark fall positions at $2m$, $3m$ and $4m$. A subject moved along three different trajectories to trigger heavy sensor movement (red), middle sensor movement (green) and low sensor movement (blue).

Table 6.1 shows the results of the pan/tilt fall detection system. The values are obtained by performing three falls per scenario and averaging their confidences. Thus, if a fall is not detected the value is lower than 0.667.

	heavy movement		middle movement		low movement		no movement	
	low	high	low	high	low	high	low	high
2m	0.993	0.953	0.987	0.990	1.000	0.857	0.990	0.990
3m	0.943	0.897	0.980	0.977	0.980	0.900	0.987	0.990
4m	0.647	0.530	0.840	0.893	0.537	0.207	0.990	0.990

Table 6.1: Confidences of falls at pan/tilt sensor changes. The falls are performed at sensor distances of $2m$, $3m$ and $4m$ and sensor heights of 115cm (low) and 215 cm (high). The subject walked different trajectories yielding to no, low, middle and heavy sensor movement.

For better interpretation the results are illustrated in Figure 6.11. It can be seen that increasing the height of the sensor has a negative effect on the performance of the fall detection system. This happens since increasing the height increases the distance of objects to the sensor and therefore the noise in the scene. The same statement is valid for the three distances to the sensor where the falls are performed. At $4m$ falls show the lowest confidences, whereas the best confidences are achieved at $2m$.



(a) Low sensor height.

(b) High sensor height.

Figure 6.11: Illustrated fall confidences from Table 6.1. The colors correspond to the amount of sensor movement.

The amount of sensor movement has a crucial impact on the fall detection system. When using a static sensor, falls at all distances are detected without problems, see Table 6.1. Middle sensor movements perform better than heavy sensor movements, which is expected. However, low sensor movements performance is surprisingly the worst of the three classes. This is owing to the dynamic motion detector which is initialized with a complete depth image. If the sensor starts moving the dynamic background model is

successively updated with points at the border of the depth image. Taking lens distortion into account the depth error remains constant at the image border and newly added data is distorted with the same error. Thus, it is less beneficial to have a newly initialized dynamic background model, since this error is not constant in the model.

Figure 6.12 shows the recall margin between the pan/tilt and the static fall detection system. It can clearly be seen that the static version performs perfectly with the given data. The precision of both systems is 1.0, since no false alarms are produced. This is evaluated using 32 sequences recorded with the PTU that do not include a fall event, but persons walking by and get tracked by the system. Also, with this additional data no false alarm is generated. A supportive factor for this is that the person classifier of *fearless* is rather restrictive and thus, does not detect errors of the dynamic motion detector as persons.

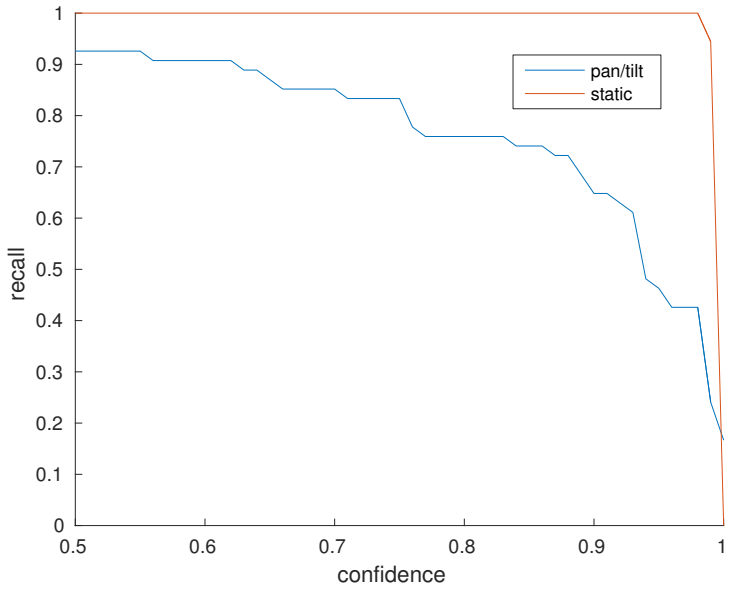


Figure 6.12: Recall of the fall detection system during pan/tilt.

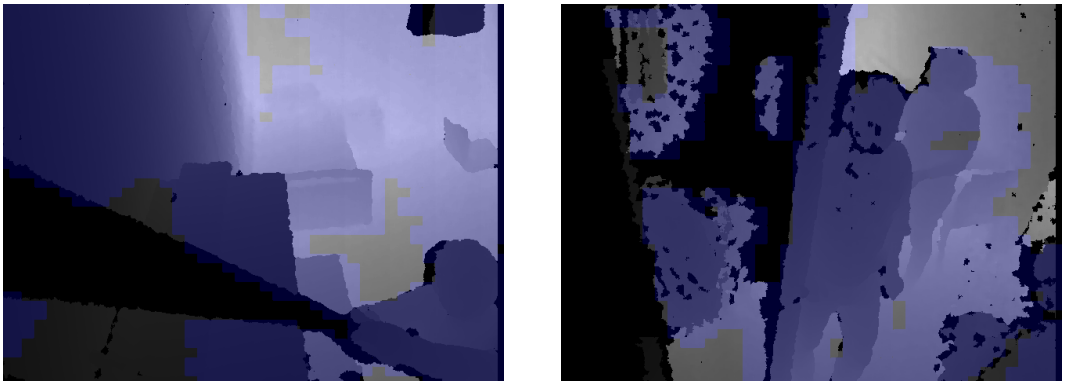
In conclusion the developed fall detection system works best at low sensor heights and after initial movement. Falls can be detected up to $4m$, but the confidences drop significantly from thereon. The precision of the system is equal to *fearless*, since incorrectly detected motion is not classified as a person and can therefore not trigger fall events.

6.3 Detection of Uncontrolled Sensor Movements

The performance of the method for detecting sensor location changes is tested on two datasets. The first dataset is the same as used by Pramerdorfer et al. [31]. It comprises 1815 sequences including recorded scenes under real and experimental conditions. As this dataset does not contain sequences in which the sensor location changes, it is used

to assess the false positive rate of the method. In this context, a false positive occurs if sensor tampering is detected by the system even though no tampering occurred.

The method reported a single false positive, which is visualized in Figure 6.13. The corresponding test sequence is particularly challenging because it depicts multiple persons, a moving door, and a ghost due to a motion detection error.



(a) True positive (b) False positive

Figure 6.13: Cases of detected sensor tampering. Foreground is shown in blue.

The second dataset comprises of 10 sequences to assess the false negative rate of the method. In these sequences, the sensor is located at a height of 3 meters and dislocated with a mop. In two recordings, the mop touches the sensor lens resulting in a black image. The other recordings displace the sensor without occluding the sensor lens. All sensor tampering incidents are detected correctly.

The results in this chapter show that the fall detection system is able to detect falls up to 4m with the support for moveable sensors . A disadvantage to the static fall detection version is the lower performance. However, the FOV is extended by a factor of 3 which is a significant improvement. The sensor should be mounted at 1.15m to get the best performance, but at this height, it is easily reachable by individuals, hence, the need for the tampering detection, which makes the fall detection system effective for practical use.

CHAPTER 7

Conclusion

In this work an existing fall detection system was extended to support depth sensors that are moved by a pan/tilt unit. The results clearly illustrated that falls can be robustly detected up to $4m$ after heavy sensor movements. Moreover, better results are obtained at a low sensor height of about $1.15m$ due to decreased distance from objects and therefore increased sensor accuracy and resolution. The effect of different artifacts that arise when using structured light depth sensor is measured by using synthetically created data. Lens distortion yields to the highest errors in the motion model however, this could be reduced by using camera calibration. The fall detection system which was used, renounces a manual setup to remain plug and playable. This feature remains within this work. Other artifacts such as rolling shutter arise when the depth sensor is moved, and its effect should not be underestimated, since the error increases with the angular velocity of the camera.

Future work will focus on correcting such errors. Decreasing depth accuracy in conjunction with increasing depth resolution is a source of errors and cannot be corrected. The only possible solution is to use other depth sensors with better properties. Another important task is the synchronization between depth frames and the corresponding pan/tilt values. Experiments have shown that a delay of $6.7ms$ can double the errors in the motion model.

All contributions were implemented in a system of loosely coupled components which allow maintenance and extension. A remote control component offers the user the option to turn the fall detection system on and off, whereas the user gets LED feedback on the system status.

Since the sensor should be mounted at heights that are accessible to humans to get the best performance, the detection of uncontrolled sensor movements was developed to detect whether persons manipulate the depth sensor. Results show that this detection

works in all tested cases with one false positive out of 1815 static test scenarios. This makes the proposed fall detection system ready for practical use.

List of Figures

List of Tables

Bibliography

[1] A. Clegg, J. Young, S. Iliffe, M. O. Rikkert, and K. Rockwood, "Frailty in elderly people," *The Lancet*, vol. 381, no. 9868, pp. 752–762, 2013.

[2] E. Kwan, S. Straus, and J. Holroyd-Leduc, "Risk factors for falls in the elderly," in *Medication-Related Falls in Older People*, pp. 91–101, Springer, 2016.

[3] L. Z. Rubenstein, "Falls in older people: epidemiology, risk factors and strategies for prevention," *Age and ageing*, vol. 35, no. suppl 2, pp. ii37–ii41, 2006.

[4] M. H. Beers, R. Berkow, *et al.*, *The Merck manual of diagnosis and therapy.* No. Ed. 17, Merck and Co. Inc., 1999.

[5] C. L. Arfken, H. W. Lach, S. J. Birge, and J. P. Miller, "The prevalence and correlates of fear of falling in elderly persons living in the community.," *American journal of public health*, vol. 84, no. 4, pp. 565–570, 1994.

[6] I. D. Cameron, G. R. Murray, L. D. Gillespie, R. G. Cumming, M. C. Robertson, K. D. Hill, and N. Kerse, "Interventions for preventing falls in older people in residential care facilities and hospitals," *The Cochrane Library*, 2005.

[7] S. Cummings, M. Nevitt, and S. Kidd, "Forgetting falls: The limited accuracy of recall of falls in the elderly," *J Am Geriatr Soc.*, vol. 36, pp. 613–616, 1988.

[8] R. Igual, C. Medrano, and I. Plaza, "Challenges, issues and trends in fall detection systems," *Biomedical engineering online*, vol. 12, no. 1, p. 1, 2013.

[9] S. Chaudhuri, H. Thompson, and G. Demiris, "Fall detection devices and their use with older adults: a systematic review," *Journal of geriatric physical therapy (2001)*, vol. 37, no. 4, p. 178, 2014.

[10] Y. S. Delahoz and M. A. Labrador, "Survey on fall detection and fall prevention using wearable and external sensors," *Sensors*, vol. 14, no. 10, pp. 19806–19842, 2014.

[11] G. Ward, N. Holliday, S. Fielden, and S. Williams, "Fall detectors: a review of the literature," *Journal of Assistive Technologies*, vol. 6, no. 3, pp. 202–215, 2012.

[12] N. Pannurat, S. Thiemjarus, and E. Nantajeewarawat, "Automatic fall monitoring: a review," *Sensors*, vol. 14, no. 7, pp. 12900–12936, 2014.

[13] S. Brownsell and M. S. Hawley, "Automatic fall detectors and the fear of falling," *Journal of telemedicine and telecare*, vol. 10, no. 5, pp. 262–266, 2004.

[14] M. A. Habib, M. S. Mohktar, S. B. Kamaruzzaman, K. S. Lim, T. M. Pin, and F. Ibrahim, "Smartphone-based solutions for fall detection and prevention: challenges and open issues," *Sensors*, vol. 14, no. 4, pp. 7181–7208, 2014.

[15] M. Alwan, P. J. Rajendran, S. Kell, D. Mack, S. Dalal, M. Wolfe, and R. Felder, "A smart and passive floor-vibration based fall detector for elderly," in *2006 2nd International Conference on Information & Communication Technologies*, vol. 1, pp. 1003–1007, IEEE, 2006.

[16] D. Litvak, Y. Zigel, and I. Gannot, "Fall detection of elderly through floor vibrations and sound," in *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 4632–4635, IEEE, 2008.

[17] N. Noury, A. Fleury, P. Rumeau, A. Bourke, G. Laighin, V. Rialle, and J. Lundy, "Fall detection-principles and methods," in *2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 1663–1666, IEEE, 2007.

[18] A. Mihailidis, B. Carmichael, and J. Boger, "The use of computer vision in an intelligent environment to support aging-in-place, safety, and independence in the home," *IEEE Transactions on information technology in biomedicine*, vol. 8, no. 3, pp. 238–247, 2004.

[19] A. A. Chaaraoui, P. Climent-Pérez, and F. Flórez-Revuelta, "A review on vision techniques applied to human behaviour analysis for ambient-assisted living," *Expert Systems with Applications*, vol. 39, no. 12, pp. 10873–10888, 2012.

[20] B. Ni, C. D. Nguyen, and P. Moulin, "Rgbd-camera based get-up event detection for hospital fall prevention," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1405–1408, IEEE, 2012.

[21] H. Nait-Charif and S. J. McKenna, "Activity summarisation and fall detection in a supportive home environment," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 4, pp. 323–326, IEEE, 2004.

[22] C. Zhang, Y. Tian, and E. Capezuti, "Privacy preserving automatic fall detection for elderly using rgbd cameras," Springer, 2012.

[23] E. E. Stone and M. Skubic, "Fall detection in homes of older adults using the microsoft kinect," *IEEE journal of biomedical and health informatics*, vol. 19, no. 1, pp. 290–301, 2015.

[24] C. Rougier, E. Auvinet, J. Rousseau, M. Mignotte, and J. Meunier, "Fall detection from depth map video sequences," in *International Conference on Smart Homes and Health Telematics*, pp. 121–128, Springer, 2011.

[25] R. A. Burckle and V. WinSystems, "The evolution of single board computers," *WinSystems Inc*, 2006.

[26] R. Planinc and M. Kampel, "Emergency system for elderly – a computer vision based approach," in *Proceedings of the 3rd International Workshop on Ambient Assisted Living*, pp. 79–83, 06 2011.

[27] R. Planinc, M. Kampel, and S. Zambanini, "Emergency system for elderly – an overview of the fearless project," in *Proceedings of the 9th International Conference on Smart Homes and Health Telematics*, pp. 225–229, 06 2011.

[28] C. Pramerdorfer, "Depth data analysis for fall detection," Master's thesis, Vienna University of Technology, 2013.

[29] "fearless 2 – final project report," tech. rep., CogVis Software und Consulting GmbH, 2015.

[30] "fearless 3 – final project report," tech. rep., CogVis Software und Consulting GmbH, 2016.

[31] C. Pramerdorfer, R. Planinc, M. Van Loock, D. Fankhauser, M. Kampel, and M. Brandstötter, *Fall Detection Based on Depth-Data in Practice*, pp. 195–208. Springer International Publishing, 2016.

[32] Z. A. Mundher and J. Zhong, "A real-time fall detection system in elderly care using mobile robot and kinect sensor," *International Journal of Materials, Mechanics and Manufacturing*, vol. 2, no. 2, pp. 133–138, 2014.

[33] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pp. 127–136, IEEE, 2011.

[34] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 573–580, IEEE, 2012.

[35] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

[36] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer Science & Business Media, 2008.

[37] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, "Real-time human pose recognition in parts from single depth images," *Communications of the ACM*, vol. 56, no. 1, pp. 116–124, 2013.

[38] F. Bagalà, C. Becker, A. Cappello, L. Chiari, K. Aminian, J. M. Hausdorff, W. Zijlstra, and J. Klenk, "Evaluation of accelerometer-based fall detection algorithms on real-world falls," *PloS one*, vol. 7, no. 5, p. e37062, 2012.

[39] H. Durrant-Whyte and T. Bailey, "Simultaneous localisation and mapping (slam): Part i the essential algorithms," 2006.

[40] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-d mapping with an rgb-d camera," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177–187, 2014.

[41] J. Biswas and M. Veloso, "Depth camera based indoor mobile robot localization and navigation," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1697–1702, IEEE, 2012.

[42] P. Benavidez and M. Jamshidi, "Mobile robot navigation and target tracking system," in *System of Systems Engineering (SoSE), 2011 6th International Conference on*, pp. 299–304, IEEE, 2011.

[43] D. S. O. Correa, D. F. Sciotti, M. G. Prado, D. O. Sales, D. F. Wolf, and F. S. Osorio, "Mobile robots navigation in indoor environments using kinect sensor," in *Critical Embedded Systems (CBSEC), 2012 Second Brazilian Conference on*, pp. 36–41, IEEE, 2012.

[44] D. Maier, A. Hornung, and M. Bennewitz, "Real-time navigation in 3d environments based on depth camera data," in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pp. 692–697, IEEE, 2012.

[45] J. Cunha, E. Pedrosa, C. Cruz, A. J. Neves, and N. Lau, "Using a depth camera for indoor robot localization and navigation," *RGB-D RSS workshop*, Los Angeles, California, 2011.

[46] H. Moradi, J. Choi, E. Kim, and S. Lee, "A real-time wall detection method for indoor environments," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4551–4557, IEEE, 2006.

[47] G. Doisy, A. Jevtic, E. Lucet, and Y. Edan, "Adaptive person-following algorithm based on depth images and mapping," in *Proc. of the IROS Workshop on Robot Motion Planning*, 2012.

[48] T. Siddharth, M. Sudhanshu, N. Akhil, C. Visesh, and K. Madhava, "Rolling shutter and motion blur removal for depth cameras," *Robotics and Automation (ICRA)*, no. 47, pp. 777–780, 2016.

[49] A. Bevilacqua and P. Azzari, "High-quality real time motion detection using ptz cameras," in *2006 IEEE International Conference on Video and Signal Based Surveillance*, pp. 23–23, IEEE, 2006.

[50] K. S. Bhat, M. Saptharishi, and P. K. Khosla, "Motion detection and segmentation using image mosaics," in *Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on*, vol. 3, pp. 1577–1580, IEEE, 2000.

[51] D. Murray and A. Basu, "Motion tracking with an active camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 449–459, 1994.

[52] M. Piccardi, "Background subtraction techniques: a review," in *Systems, man and cybernetics, 2004 IEEE international conference on*, vol. 4, pp. 3099–3104, IEEE, 2004.

[53] E. Hayman and J.-O. Eklundh, "Statistical background subtraction for a mobile observer," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pp. 67–74, IEEE, 2003.

[54] H. Sarbolandi, D. Lefloch, and A. Kolb, "Kinect range sensing: Structured-light versus time-of-flight kinect," *Computer Vision and Image Understanding*, vol. 139, pp. 1–20, 2015.

[55] J. Posdamer and M. Altschuler, "Surface measurement by space-encoded projected beam systems," *Computer Graphics and Image Processing*, vol. 18, no. 1, pp. 1–17, 1982.

[56] G. Guidi, S. Gonizzi, and L. Micoli, "3d capturing performances of low-cost active range sensors for mass-market applications," *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLI-B5, pp. 33–40, 2016.

[57] M. Harville, "Stereo person tracking with adaptive plan-view statistical templates," *Image and Vision Computing*, vol. 22, no. 2, pp. 127–142, 2004.

[58] Z. Zhang, *Camera Calibration*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

[59] J. Smisek, M. Jancosek, and T. Pajdla, *3D with Kinect*. Springer London, 2013.

[60] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Sixth International Conference on Computer Vision*, pp. 839–846, IEEE, 1998.

[61] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[62] J. Mandel, "Use of the singular value decomposition in regression analysis," *The American Statistician*, vol. 36, no. 1, pp. 15–24, 1982.

[63] S. Satoshi and A. Keiichi, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985.

[64] D. Das and S. Saharia, "Implementation and performance evaluation of background subtraction algorithms," *International Journal on Computational Sciences & Applications (IJCSA)*, vol. 4, no. 2, pp. 49–55, 2014.

[65] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[66] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[67] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–1443, 2012.

[68] A. Cosentino, "A practical guide to panoramic multispectral imaging," *e-conservation Magazine*, vol. 25, pp. 64–73, 2013.

[69] W. Hugemann, "Correcting lens distortions in digital photographs;," *Ingenierbüro Morawski +Hugemann: Leverkusen, Germany*, 2010.

[70] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, pp. 1330–1334, Nov. 2000.

[71] D. A. Forsyth and J. Ponce, *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference, 2002.

[72] M. Meilland, T. Drummond, and A. Comport, "A unified rolling shutter and motion blur model for 3d visual registration," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2016–2023, 2013.

[73] E. Ringaby and P.-E. Forssen, "Scan rectification for structured light range sensors with rolling shutters," in *2011 International Conference on Computer Vision*, pp. 1575–1582, IEEE, 2011.

[74] ON Semiconductor, *1/2-Inch Megapixel CMOS Digital Image Sensor*, 4 2015. Rev. M.

[75] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521623049, 2000.

[76] M. Slee, A. Agarwal, and M. Kwiatkowski, "Thrift: Scalable cross-language services implementation," *Facebook White Paper*, vol. 5, no. 8, 2007.

[77] O. Rukundo and H. Cao, "Nearest neighbor value interpolation," *Advanced Computer Science and Applications*, vol. 3, no. 4, pp. 25–30, 2012.

[78] G. S. Watson, "Linear least squares regression," *The Annals of Mathematical Statistics*, vol. 38, no. 6, pp. 1679–1699, 1967.

[79] B. Choo, M. Landau, M. DeVore, and P. A. Beling, "Statistical analysis-based error models for the microsoft kinecttm depth sensor," *Sensors*, vol. 14, no. 9, pp. 17430–17450, 2014.