DLR-IB-RM-OP 2019-220

Detection of Object Movements in non-static RGBD Camera Streams

Bachelorarbeit

Jonas Herzog

DLR Deutsches Zentrum für Luft- und Raumfahrt

Institut für Robotik und Mechatronik	BJ.: 2019 IB.Nr.: DLR-IB-RM-OP-2019-220

BACHELORARBEIT

DETECTION OF OBJECT MOVEMENTS IN NON-STATIC RGBD CAMERA STREAMS

Freigabe:	Der Bearbeiter:	Unterschriften
	Jonas Herzog	
	Betreuer:	
	Dr.-Ing. Tim Bodenmüller	
	Der Institutsdirektor	
	Prof. Alin Albu-Schäffer	

Dieser Bericht enthält 75 Seiten, 21 Abbildungen und 4 Tabellen

Ort: Oberpfaffenhofen	Datum: 16.09.2019	Bearbeiter: Jonas Herzog	Zeichen:

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

DLR

Thema

Detection of object movements in non-static RGBD camera streams

---

Bachelorarbeit

für die Prüfung zum

Bachelor of Engineering

des Studiengangs Informationstechnik

der Dualen Hochschule Baden-Württemberg Mannheim

von

Jonas Herzog

16. September 2019

Bearbeitungszeitraum:	24.06.2019-16.09.2019
Matrikelnummer, Kurs:	1729061, TINF16ITIN
Abteilung:	Abteilung Perzeption und Kognition
Institut:	Institut für Robotik und Mechatronik
Ausbildungsfirma:	Deutsches Zentrum für Luft- und Raumfahrt e.V.
Betreuer:	Dr.-Ing. Tim Bodenmüller
Unterschrift Betreuer	

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema „Detection of object movements in non-static RGBD camera streams" selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Oberpfaffenhofen, der 1. Oktober 2019

Zusammenfassung

Die Erkennung von Objektbewegungen in Videosequenzen ist eine Notwendigkeit für viele Videoanalyseanwendungen. Für mobile Roboter, die mit RGB-D-Kameras ihre eigene Position bestimmen, ist es erforderlich Objekte, die nicht Teil der statischen Umgebung sind, herauszufiltern. Die Identifizierung dieser Objekte kann über deren Eigenbewegung erfolgen.

Daher ist das Ziel dieser Arbeit die Entwicklung eines Systems, das aus dem Live-Stream einer sich bewegenden RGB-D Kamera dynamische Objekte von statischen Objekten in Echtzeit trennen kann. Die größte Herausforderung für diese Aufgabe ergibt sich aus der Tatsache, dass Verschiebungen in aufeinanderfolgenden Bildern entweder durch eine Kamerabewegung oder durch dynamische Objekte verursacht werden können.

Es wird ein System vorgestellt, das eine binäre Segmentierungsmaske erzeugt, welche für jeden Pixel anzeigt, ob es zu einem statischen oder dynamischen Objekt gehört. Das Verfahren in dieser Arbeit basiert auf Tiefenhintergrundmodellierung, was für Echtzeit-Bewegungserkennung mit einer beweglichen Kamera neuartig ist. Es wird eine Anti-Blurring Technik vorgeschlagen, welche die Schärfe von Kanten im Hintergrundmodell bewahrt.

Die experimentelle Validierung zeigt, dass dynamische Objekte erfolgreich mit einer durchschnittlichen Bildrate von 25 Bildern pro Sekunde segmentiert werden können. Anhand der Analyse verschiedener Setups und Szenarien werden Anpassungsmöglichkeiten vorgestellt. Im Vergleich zu modernen Methoden mit dem gleichen Ziel konnte die Verwendung von Tiefenbildern sowie die Anti-Blurring Technik in vielen Szenarien die Segmentierungsergebnisse verbessern.

Abstract

Detecting object movements in video sequences is essential for various video analysis applications. For mobile robots that use RGB-D cameras to determine their own position, it is necessary to pre-filter interfering objects that are not part of the static environment. These interfering objects can be identified as they are dynamic and exhibit thus an own motion.

Therefore, given a live color and depth video input stream from a moving camera, the objective of this work is to develop a system capable of segmenting dynamic objects from static objects in real-time. The key challenge for this task arises from the fact that displacements in consecutive images could either be caused by a camera movement or by dynamic objects.

A system is proposed which yields a binary segmentation mask showing for every pixel whether it is assumed to belong to a static or dynamic object. The method in this work is based on depth background modeling which is novel for real-time motion detection with a moving camera. An anti-blurring technique is proposed which preserves the acuity of the background model.

Experimental validation demonstrates that dynamic objects are successfully segmented at an average frame rate of 25 frames per second. Adaptation possibilities are presented based on the analysis of different setups and scenarios. Compared to state-of-the-art methods with the same objective, the usage of depth images could enhance the detection accuracy in many scenarios.

Contents

List of Acronyms and Abbreviations

DLR	German Aerospace Center
ICP	Iterative Closest Point
IoU	Intersection over Union
KLT	Kanade-Lucas-Tomasi feature tracker
LSRE	Least Squares Regression Estimation
RANSAC	Random Sample Consensus
RGB-D	Red-Green-Blue-Depth
RMC	Robotics and Mechatronics Center
SLAM	Simultaneous Localization and Mapping
SURF	Speeded up Robust Features

1 Introduction

The detection of object movements in video streams is a fundamental task in computer vision. It is required in many fields such as automatic surveillance [1], object tracking [2], behavior analysis [3] or Simultaneous Localization and Mapping (SLAM) in dynamic environments [4]. Methods for motion detection with a static camera have already been developed in the early history of computer vision. Borst and Engelhaaf [5] examined the principles of visual motion detection, Horn and Schnunck [6] developed a method to determine apparent velocities in an image sequence and Wren *et al.* [2] showed how moving objects can be extracted by modeling the static scene. However, when the camera moves, the problem is more challenging. The apparent motion in an image sequence could not only be caused by moving objects, but also by the ego-motion of the camera. When no reliable information about the camera movement itself is known, it has to be estimated by analyzing the consecutive images. This is done in Visual Odometry systems, where the key objective is the identification of the position where the images have been taken from. A lot of Visual Odometry systems, are, however, designed for the operation in static environments, where the camera is the only object to move in the scene. Thus the emerging challenge for motion detection in dynamic environments is to extract the camera motion among the observed additive apparent motions and detect objects that move differently. Several approaches exist already that tackle this issue. One of the most recurrent issues in these methods is the trade-off between computational efficiency and detection accuracy. Algorithms operable in real-time do often yield substantially poorer results [7]. In this work, a real-time method for the segmentation of moving objects is chosen based on related work, implemented and evaluated.

1.1 Problem Statement

In the Robotics and Mechatronics Center (RMC) of German Aerospace Center (DLR), a humanoid robot called *Rollin' Justin* (Figure 1) was developed as a platform for research in service robotics. *Rollin' Justin* is designated to assist astronauts in space as well as care-dependent people in their households. For the latter, where the robot is deployed in the interiors of a building, the environment in which the robot interacts is partially known. Walls,

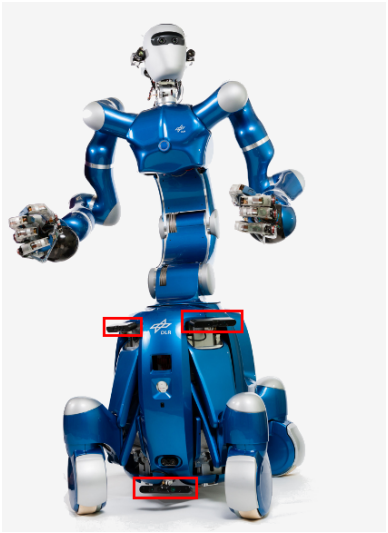

Figure 1: The humanoid robot Rollin' Justin, equipped with RGB-D cameras shown in the red rectangles.

windows, doors and most of the furnitures remain at their places, allowing the robot to use a three-dimensional map of the interiors to interact with the environment. Characteristic interactions are navigation, object grasping and other robot movements. For all of these tasks, it is necessary for the robot to know its own position in the map, otherwise path planning fails, collisions can occur and objects are missed.

Therefore, *Rollin' Justin* determines its own position by comparing the current observations with the three-dimensional map of the environment. This process is also called localization. *Rollin' Justin* is equipped with several Red-Green-Blue-Depth (RGB-D) cameras that are used for the localization. By recognizing walls, furnitures and other static objects in a certain distance and direction, it is possible to derive the location where the images have been taken from, which is, namely, the robot's position.

Dynamic objects, i.e. those that move independently of the robot in space, make the localization challenging so far. If, for example, people carry out activities in the environment and move through the robot's field of view, misinterpretations can occur. As no image segmentation is performed prior to the localization, information gained from measurements on the human body is included in the matching process. Since the human body does not exist in the 3D model of the environment, the matching algorithm cannot associate it correctly, resulting in wrong robot localizations.

1.2 Objective and Contributions

In this work a system is to be developed that is capable of segmenting objects that are not part of the background. As there could be also other robots, object displacements and further disturbing factors in the scene, the approach cannot be restricted to the exclusive detection of humans. Also, no prior knowledge is given, so the segmentation cannot be achieved solely based on the appearance of the objects. Instead, foreground objects are detected by their relative motion to the background. The apparent motion of the background follows a certain pattern, whereas the foreground objects exhibit a different motion. Extracting the foreground objects based on this pivotal idea is subject of this work. The proposed system should process the images from an RGB-D camera and filter out all objects that have exhibited an independent motion. It serves thus as a preprocessing stage before any localization is performed. When only the information of the static background is considered for the localization process, misinterpretations as described above can be avoided.

Given an RGB-D stream from a moving camera as its only input, the system should output a label for every pixel, where the label indicates whether the pixel belongs to a foreground object. Foreground objects are defined as all objects that have actually moved in the scene. These objects are also referred to as moving objects or dynamic objects, however, their own motion may cease and they should still be detected as foreground objects. This requirement is sensible in particular for the robotic use case discussed before because objects could just be displaced once and yet they are no longer useful for the localization process.

Regarding the requirements on its qualitative results, over-filtering, i.e. yielding false foreground positives, is acceptable to a minor degree whereas undetected foreground may still compromise the success of the localization process.

The proposed method is based on local background modeling. The system tries to model the depth of the background in current field of view of the camera. Consequently, all elements in the image, which do not occur in the background model, are probably foreground objects.

Combining and extending existing methods from Yi *et al.* [8] and Yun *et al.* [7], the optimal setup and configuration is sought that achieves the best detection accuracy while still fulfilling the real-time requirement.

The main contributions of this work are

- the usage of depth images for background modeling
- modifications improving the performance for indoor scenes
- an anti-blurring scheme preserving the sharpness of the background model

The key task is thereby the maintenance of an uncontaminated background model. It must provide consistent information about the appearance of the static background in every area of the current frame and must not contain any parts of foreground objects. Moreover, it must be able to cope with errors introduced by inaccurate camera motion estimations.

In contrast to pure motion detection, where often only boundaries of the object are detected, the system must continuously yield a foreground segmentation mask for *every* pixel. The system is designed to operate in real-time on a CPU with the live camera stream as its input, meeting the hardware requirements of *Rollin' Justin*.

1.3 Outline

The following chapters describe the conceptual design, implementation and evaluation of a system for moving object detection with a moving camera. Chapter 2 examines related work. First, fundamental techniques for the detection of object movements with a static camera are explained. Thereafter, it is discussed how the challenges that arise with a moving camera are solved in related work. Chapter 3 focuses on the conceptual design of the proposed system, where a method from related work serves as a base line. Here, the proposed system is broken down in its components and their underlying algorithms. Improvements and extensions are presented that were motivated from the drawbacks of the existing system. How the entire system is implemented in *C++* is shown in chapter 4. Besides the architecture and the overall data flow, the tasks of the functional modules that form the system are explained in detail. In chapter 5, the implemented system is evaluated. Examination of the effects of the proposed improvements from chapter 4 and further optimizations for both run-time and detection accuracy is performed. The obtained results are compared to results achieved by related work. Finally, chapter 6 summarizes the idea and the achievements of this work and features enabled future work.

2 Detection of Object Movements in Related Work

2.1 Fundamental Techniques for Movement Detection with a Static Camera

Basic visual motion detection techniques that assume a static observer can be classified in three categories [9]: Frame difference, background subtraction and optical flow. This section explains the key principles behind each technique as their ideas can be utilized also for motion detection with a moving camera.

2.1.1 Frame Difference

Frame differencing is a common technique when moving objects are to be detected in the static camera case [10]. Two consecutive frames are compared by calculating the difference of each pixel's value from the pixel's value in the other frame. The resulting differencing image describes the change between these two frames. If its value is high for a certain pixel, it is probable that a boundary of a dynamic object is located there. Since the images from both frames show the dynamic object, merely at a slightly displaced position, the entire dynamic object cannot be extracted by differencing. Frame difference on depth images typically only detects depth edges, i.e. the boundaries of moving objects. Frame difference on color images detects both boundaries and the texture of dynamic objects.

2.1.2 Background Subtraction

Background subtraction is similar to the frame difference method as there is also a differencing image calculated which serves as an indicator for dynamic objects. Instead of selecting two consecutive frames, one initial frame and another more recent frame is selected. The more recent frame is the frame in which the dynamic objects are to be detected. The image from the initial frame shows the scene before any dynamic objects entered the scene. Hence, the differencing images shows all objects that look different than the static scene behind. In

contrast to frame difference from 2.1.1, the whole shape of the dynamic object is detected as long as the dynamic object differs from the static objects behind. However, when parts of the dynamic object exhibit a visual appearance similar to the static scene, holes emerge in the detection mask. Also, when the background scene does not remain exactly as it was captured in the initial frame, background subtraction yields false detections. To overcome this problem, it is necessary to update the image that is expected to show the static background. This is also referred to as background modeling.

2.1.3 Optical Flow

While the frame difference and background subtraction techniques are based on the pixel value change, the concept of the optical flow attempts to find point correspondences in two consecutive images. These point correspondences form the start and end points of motion vectors that show how objects have apparently moved. The underlying assumption is that objects do not change their visual appearance over time. Horn and Schunck [6] define the optical flow as the distribution of apparent velocities of movement of brightness patterns in an image. At the same time Horn and Schunck provide in [6] a method to determine the optical flow. Treating the optical flow as a motion vector field, two main types of optical flow methods are to be distinguished. *Dense* optical flow methods [6] [11] [12] [13] yield a motion vector for every pixel. On the other hand, the *sparse* optical flow [14] is only defined for a few points in the image. Given the dense optical flow field, the binary foreground mask could be obtained by extracting all points whose motion vectors exceed a certain length.

Determining the optical flow is a more costly operation than background subtraction and frame difference. In contrast to background subtraction, the optical flow and frame difference methods only extract objects that have moved from the previous to the current frame. If the motion of the dynamic object ceases, they do no longer detect the dynamic object.

2.2 Methods for the Operation with a Moving Camera

In the previous section, basic techniques for the detection of motion were introduced for the case of a static camera. However, as the objective of this work is to segment dynamic objects with a moving camera, applying these techniques is not sufficient. All of the above discussed

methods yield false foreground positives when the camera moves even though all objects in the image are static. Yet their underlying principles can be reemployed for the operation with a moving camera.

2.2.1 Camera Motion Estimation

When applying the techniques from sec. 2.1 to a static camera, objects remaining static in the scene are falsely labeled as foreground and objects that move parallel to the camera are falsely classified as background objects. This is because these techniques determine the motion relative to the camera. When the camera itself moves, consecutive images differ even though there is no dynamic object in the scene. Therefore, in order to make the techniques from sec. 2.1 applicable for moving cameras, the motion caused by the camera must be compensated. To this end, several methods for the estimation of the camera motion between two frames exist.

Three-dimensional approach with color and depth images

Some methods that use both color and depth images estimate the motion of the camera by searching for the best alignment of the two image in all three dimensions. Knowing the pinhole model of the camera, it is possible to convert the 2D image with depth and color values to 3D points with color values. Given these 3D points of two consecutive images, it is subsequently possible to estimate a transformation that aligns the points of both images. This transformation is caused by and thereby corresponds to the motion of the camera.

The scene flow algorithms from Jaimez *et al.* [15] [16] [17] apply this technique. They align two images by finding a transformation that minimizes both the geometric and photometric errors in a reprojection. The geometric error is defined as the difference of the depth value and the photometric error is, in their case, simply the intensity value difference. The scene flow is a three-dimensional extension of the optical flow (see sec. 2.1.3) and its motion fields can thereby already serve as a valuable indicator for independent motion from dynamic objects. However, solving the minimization problem in real time requires a high degree of parallelization which could only be achieved on a GPU in [16]. Jaimez *et al.* proposed an accelerated version for CPUs in [17], where clustering was applied in order to reduce computational cost. Running on multiple CPU cores, it still consumed 80ms per frame at

QVGA resolution. In contrast to the objective of this work, where a binary information about foreground or background belonging is sufficient, the scene flow from [17] aims to figure out the direction and speed of objects while providing a visual odometry at the same time. Scona *et al.* [18] also utilize the idea of minimizing the geometric and photometric reprojection error in order to obtain the camera motion. Their work has the objective to make SLAM applicable in dynamic environments. Therefore, they jointly estimate the camera motion and foreground segmentation in a single term which is to be minimized. This is achieved by forcing clusters to be segmented as dynamic when their residuals are high and simultaneously reducing their effect on the camera motion estimation. Thus the system from Scona *et al.* is able to generate a foreground segmentation mask, however, also this algorithm is only real-time operable when running on a GPU.

Other methods try to find matchings in point clouds that were generated from the depth images. A common technique for the registration of point clouds is the Iterative Closest Point (ICP). The ICP aims to find a transformation that minimizes the sum of the squared distances between the two point clouds from the previous and current frame. As it considers the closest point in the target point cloud in each iteration, it performs well only if the point clouds have a small displacement. Variants of the ICP for the estimation of the camera motion are used in [19] and also in SLAM systems like *KinectFusion* [20]. Li and Lee [4] use an intensity assisted variant of the ICP, where they create 3D points with intensity values as mentioned above. Having thus fused color and depth information, in the point matching process they aim to minimize the intensity difference in addition to the Euclidean distance.

Two-dimensional approach for color images

Less time-consuming than most point cloud registration approaches are classical tracking methods based on the optical flow in two dimensions. Given only color or gray images, the camera motion in dynamic environments can be determined by combining a tracking method, such as Kanade-Lucas-Tomasi feature tracker (KLT)[14] and a consensus finder, such as Random Sample Consensus (RANSAC). First, the tracker finds corresponding points in the two consecutive images. Then, the transformation that aligns these point pairs best is determined by finding a large consensus set among the point pairs. This transformation is the inverse of the camera motion since when the camera moves exemplary to the left, the vectors of the apparent motion field in the image point to the right. KLT is often used because

it is a fast method to find corresponding features in two images and works well if the two images have not to much displacement. RANSAC is a suitable choice in particular because it is resistant against motion outliers that are caused by the independently moving dynamic objects. RANSAC works with the found point correspondences of two images as its input and yields a transformation that aligns the point pairs. This is achieved by iteratively selecting four point pairs, computing the exact transformation valid for them, checking for how many other pairs this transformation fits and repeating this process until a transformation is found that is supported by a large ratio of point pairs. Exactly four points are selected because the transformation to determine, with that the relationship between the previous and the current image, is a *homography*. A homography sets any two images of the same planar surface in a relationship. When describing the transformation through a homography, it is assumed that the camera has taken images from the same surface from two different perspectives. The perspective transformation associated with the homography is non-rigid, but preserves straight lines.

The principle of applying KLT and subsequently RANSAC is implemented in the moving object segmentation systems from Xu *et al.* [21], Kim *et al.* [22], Wu and Chiu [23], Sun *et al.* [24] Yi *et al.* [8], Yun *et al.* [7] and Makino *et al.* [9]. They all work with the intensity image and determine the camera motion first and then proceed with the obtained results. Xu *et al.* [21] propose a geometrical approach that extracts the pixels that do not follow the motion induced by the camera. Kim *et al.* [22], Yi *et al.* [8] and Yun *et al.* [7] maintain a model of the background. Wu and Chiu [23] and Makino *et al.* [9] combine the optical flow with a background model for deriving foreground probabilities.

Inaccuracies in camera motion estimation as a primary issue

Once the camera motion has been estimated by one of the aforementioned techniques, attempts can be made to remove the effects caused by the moving camera in order to make detection methods from sec. 2.1 applicable. With an exact transformation that warps every pixel from the previous frame onto the correct location in the current frame, the problem of the moving camera were solved and methods for static cameras could be directly applied. However, small offsets in the projection on the current frame are enough for a significant amount of false detections, especially at edges in the image. Yet inaccuracies in the estimation of the camera movement are barely evitable. This paragraph explains the emergence of these

inaccuracies under different circumstances.

In the two-dimensional case, a homography represents the camera motion. Homographies set, by definition, two observations of the same planar surface in a relationship. Unfortunately, in most indoor environments the observed objects are not located on the same surface. Consensus estimators like RANSAC strive to find a solution that keeps the thus induced error low for most of the pixels, however, a small error might remain through this conjuncture. After applying the homography transformation on the previous frame, the resulting image does therefore almost never coincide with the current image. Less inaccuracies emerge when finding the dense pixel matchings. When correspondences are found for every pixel, no transformation matrix needs to be estimated. Comparison of two images can be directly applied on the found point correspondences. Interpolation is still necessary since there may be remain some unmatched points. In contrast to sparse optical flow methods like KLT, determining such a dense optical flow is more expensive. Methods attempting to obtain accurate results [23] [12] do not achieve real-time performance. For the explanation of sparse and dense, see subsec. 2.1.3.

In the three-dimensional case, where point clouds are to be registered, the occurrence of inaccurately warped points depends on the applied method. The ICP in its conventional form estimates a rigid body transformation. Rigid body transformations cannot represent scalings as they are induced by zooming for instance. A homography can represent any rigid body transformation but not vice versa. Therefore, the errors resulting from the inaccurate transformation when applying ICP are greater than if a homography were estimated. More accurate are methods that utilize the pinhole model of the camera. Instead of aligning point clouds, the approach is here to find a camera position that would explain the difference between two frames. Thus Jaimez *et al.* [15] and Scona *et al.* [18] propose a warping function where a transformation is applied to the 3D points of the current frame, which are subsequently reprojected in the two-dimensional plane according to the camera's pinhole model. It is searched for the transformation that causes the best match in the image plane. Having solved this error minimization problem as it is done in [18] on real time on the GPU, the true motion of the camera has been estimated. In contrast to the other techniques, the transformation does not align the observations and assume the camera motion is the inverse of this transformation. Instead, the transformation of the point where the field of view originates has been determined, which is the camera's location. Thereby, the estimated camera motion

could be exact, however, deviations in depth measurements, depth measuring errors, noise, illumination differences and the need to solve the minimization problem in a coarse-to-fine scheme [18] are sources for new inaccuracies in the alignment.

2.2.2 Moving Object Detection after Camera Motion Estimation

The estimation of the camera motion is motivated by the idea of enabling the comparison of images taken from different spots. When the current frame is aligned by transforming it according to the estimated transformation, methods from sec. 2.1 become applicable again. As the previous section points out, a key issue for motion detection with a moving camera are inaccuracies in the estimation of the camera motion. Therefore, the selected method must be resistant to small spatial deviations. This subsection reconsiders the techniques from sec. 2.1 with respect to the inaccurately estimated camera motion.

Frame Difference

Sun *et al.* try to apply frame difference on the camera motion compensated image in [24]. They use Speeded up Robust Features (SURF) [25] to determine a set of point pairs in both images. Next, RANSAC is applied to eliminate outliers. The homography corresponding to the camera motion is estimated from the inliers as described in subsec. 2.2.1. After aligning the previous and current image with this transformation, they apply interpolation to fill gaps and define a least squares of distances minimization term to refine the transformation. On the thus aligned images they now apply frame difference. This way they detect motion at object boundaries. However, the images in their evaluation show that also edges of static objects are detected. To cope with these false detections and to segment the entire body, particle filter [26] based tracking as well as depth value based clustering is employed in their work.

Frame difference as in subsec. 2.1.1 is not suitable for the objective of this work as it only detects the boundaries of moving objects. Subsequently applying clustering mechanisms to segment the entire body of the moving object as in [24] would be too time-consuming. Moreover, whole surfaces could falsely be detected as foreground when trying to cluster objects based on their boundaries. When two images are aligned with a small offset and frame difference is applied, the effects are similar to an asymmetric sharpening kernel, where edges are extracted. Consequently, edges of actually static objects are then detected as foreground.

Background Modeling and Subtraction

Kim *et al.* [22], Yi *et al.* [8] and Yun *et al.* [7] use the intensity image only and maintain a background model in their systems. The background model is only kept for the current field of view of the camera. Hence, no global coordinate system is used. As a result, to make the existing background model usable when a new frame arrives, it must be transformed according to the camera motion between the previous and the current frame. Having transformed the previous background model, it can be updated with the current observations as described in subsec. 2.1.2. Also for background modeling, the inaccuracy of the transformation hampers the process. False detections similar to those in the frame difference method occur when the transformation is not exact. This is because background subtraction, which is applied to detect objects that are not part of the background, is also an image differencing method, where in this case the image to be subtracted is the background model.

Kim *et al.* [22] tackle the inaccuracy problem by considering also neighboring pixels in both the update and background subtraction process. When the current observation updates the existing background model, the pixel with the best match to the current observation is selected from the neighborhood in the transformed background model of the previous frame. In contrast, the common approach is to update the underlying pixel in the transformed previous background model directly, without looking at the neighborhood. Also the background subtraction is applied on the pixel with the best match in the neighborhood of the model, which prevents false detections caused by the inaccuracy of the transformation. In their system, the background model is a Gaussian model, see also subsec. 2.1.2. In addition to the mean and variance, they introduce an age value that corresponds to the number of frames that have been incorporated in the background model at the respective pixel. The higher the age, the lower is the proportional impact of the current frame. When the camera moves, newly covered regions initially have the age zero and are thus faster updated than regions that have already been updated several times. This way Kim *et al.* try to keep the model receptive in uncertain regions and stable in well known regions.

However, the background model from Kim *et al.* is vulnerable to corruption [23]. When dynamic objects are located in the newly covered areas after a camera shift, they are initially classified as background as no prior information exists. When the dynamic object is observed to move in the following images, some background areas which were occluded by the dynamic object are revealed, others are occluded. Instead of only labeling the newly occluded areas

as foreground, also the revealed areas are falsely classified as foreground. This is because the background model contains the dynamic object. Background subtraction then detects a difference between the current observation and the background model. When the camera keeps moving, this error is dragged along from frame to frame. Suchlike detections of dynamic objects that do not exist anymore at the detected location are also referred to as ghosts.

Yi *et al.* [8] try to solve this issue by introducing a second Gaussian model in addition to the existing one from Kim *et al.* [22]. The second model collects observations that do not match with the primary background model. This has mainly two effects: Firstly, the primary background model is not contaminated with single false detections. Secondly, when observations in subsequent frames support the conjecture of the second model and it becomes thus more probable that the background actually looks like it is modeled in there, focus can be switched on the second model. In the work of Yi *et al.* [8] this idea is realized by comparing the ages of the Gaussian model. When the age of the second model exceeds the age of the first model, they are switched. Dynamic objects in newly covered areas are thereby less problematic. Yi *et al.* also estimate a homography based on the intensity images, but tackle the inaccuracy problem with a different approach than Kim *et al.* They reduce the resolution of the background model drastically by dividing the image in equal grid cells. Instead of transforming every pixel with the homography, the cell centers are transformed and the new grid is obtained by averaging the overlapping transformed cells. By not matching pixels directly this way, inaccuracies in the homography have less impact. Moreover, computational cost is reduced by maintaining only a low resolution model. Yi *et al.* thus claim their algorithm can run on real time on a standard mobile phone from 2013.

Yun *et al.* [7] try to improve the robustness of the system of Yi *et al.* by the assumption that dynamic objects move smoothly. They further reduce computational cost by restricting the considered pixels to a sample set which is created based on the foreground probability in the previous frame. When operations are performed only on a few pixels, the time consumed for processing one frame decreases.

The methods of Yi *et al.* [8] and Yun *et al.* are delineated in more detail in chapter 3.

The work of Wu and Chiu [23] as well as the *IViBe* [27] use multiple samples instead of a Gaussian model to represent the background. Each time new observations arrive, they replace an existing sample through the current observation with a certain probability. As against Wu and Chiu's system, the IViBe is real-time capable and also models the depth of

the background. However, drawbacks are that no scheme for coping with motion inaccuracies is proposed, newly covered regions are often detected as foreground and objects that cover small proportions of the image are falsely incorporated in the background model [10].

There are also methods that model the background three-dimensionally. In contrast to the above explained methods, where only a small size background model is maintained, Scona *et al.* [18] model the background globally and in three dimensions. This has the advantage that no information is discarded in regions that slide out of the image and may reenter later on. Their representation of the background is a set of 3D disks, also called surfels. Each disk has an associated position, normal vector, RGB color, viability value, timestamps and an age value. These values are updated for concerned disks with every incoming frame. Rendering from this 3D model is performed to predict the image the camera would see if the scene consisted of static objects only. As subsec. 2.2.1 already elucidates, the set of dynamic pixels is jointly obtained with the camera motion estimate through a single minimization term. However, real time performance with [18] can only be achieved on a GPU.

Optical Flow

Detecting moving objects after the estimation of the camera motion is also possible by analyzing the optical flow field. Independently moving objects exhibit a divergent motion and thus remain in the optical flow field after removing the effects caused by the camera. If optical flow methods like the KLT (sparse) or *EpicFlow* [12] (dense) have already been used for estimating the camera motion, their results can be reutilized. This is in particular advantageous with respect to the runtime since no additional computations like frame difference or background subtraction have to be performed.

Huang *et al.* [28] estimate two different types of optical flow. First, they employ *FlowNet2.0* [29], a fast neural network based method, to obtain the current optical flow field. Next, they try to calculate the optical flow of the background. To this end, they consider the background optical flow field as a quadratic function of the point coordinates. For estimating this quadratic function, they sparsely select pixels distributed over the entire image and find a solution with Least Squares Regression Estimation (LSRE) and RANSAC. Points for which the background optical flow differs strongly from the actual optical flow field are assumed to have a high foreground probability. Therefore, the foreground mask is obtained by thresholding the

difference of the two optical flow fields. Wu and Chiu [23] also calculate two optical flow fields. For the actual dense optical flow, they use the method of Liu [30]. The background optical flow is generated by applying the estimated homography corresponding to the camera motion to every point and measuring the displacements. Wu and Chiu now calculate an initial foreground guess by comparing the two optical flows. As against Huang *et al.* [28], they combine this with background modeling and are not able to achieve real-time performance.

While Huang *et al.* and Wu and Chiu separately calculate the camera motion and the dense optical flow, Sun *et al.* [31] merge these processes by deriving the camera motion from the dense optical flow. The homography corresponding to the camera motion is estimated by finding the most common motion in the dense optical flow field yielded by [12]. After that, generating the background flow and determining the foreground probability based on the difference of the vector fields is performed as in Wu und Chiu's work. Furthermore, Sun *et al.* memorize how foreground objects look like by storing their RGBD values once they have been detected. This principle is also referred to as foreground modeling. Since the dense optical flow method they use [12] takes about seven seconds to process one frame, their method is not real-time operable.

Makino *et al.* [9] also try to use discrepancies in the optical flow field as an indicator for dynamic objects. Unlike the methods above, they are able to go without the estimation of a dense optical flow. This is beneficial as the calculation of the dense optical flow is the major time consumer in the other methods. To this end, they apply conventional KLT and obtain thus a sparse optical flow, from which again the homography is estimated with RANSAC (see subsec. 2.2.1). The background flow is generated from the homography as in the previously discussed methods. Equally, it is subtracted from the actual optical flow, which is in this case, however, the sparse one resulting from KLT. Therefore, subsequent interpolation is required to obtain the differencing vectors for every pixel. To make the foreground decision, Makino *et al.* consider the angle of the residual flow vectors after subtraction instead of focusing on their Euclidean length. This is based on the assumption that moving objects do not often change their moving direction. Consequently, when the angle of the residual flow vector remains similar over time, the corresponding pixel is assumed to have a high foreground probability. Merging the so determined foreground probability with another, background modeling based foreground indicator from [8], their system is able to achieve a higher accuracy than [8] and still runs at 45fps at a similar resolution [9].

2.2.3 Summary

The prevailing approach in related work is to estimate first the camera motion (subsec. 2.2.1) so that the current image can be transformed in a coordinate system in which knowledge from previous frames is available. This can be achieved either two-dimensionally, which is faster, or, considering also the depth image, three-dimensionally, which is more accurate. Being thus able to compare the current image with prior knowledge, different indicators are utilized in order to distinguish between static and dynamic objects. The most common indicators are a large difference between consecutive frames, the deviation from an incrementally built background model and anomalies in the optical flow (2.1). When working with a moving camera, frame difference seems to be the least suitable technique as inaccuracies in the camera motion estimation are difficult to handle. Background modeling and subtraction has the advantage of accumulating information over time. Detection results can thus become more stable as if the current image is only compared with the previous one. Background modeling can be either performed in 2D or 3D, where again the above mentioned trade-off between speed and accuracy is pivotal. Methods utilizing anomalies in the optical flow often calculate the flow field for every pixel. This is necessary when seeking the dense foreground segmentation mask without interpolation, however, it comes with high computational cost and is hardly feasible in real-time.

3 Proposed Method: Background Subtraction with Two Local Background Models

Through its robustness and efficiency, the background modeling technique from *MCD5.8* [8] proved to be a suitable base line for this work. Therefore, the method proposed in this work is based on *MCD5.8* [8] and the system of Yun *et al.* [7], who attempt to enhance both speed and accuracy of *MCD5.8* by considering fewer pixels and by remembering the position of dynamic objects. Section 3.1 to 3.4 cover the base method from Yi *et al.* and show adaptations and improvements to it. Section 3.5 focuses on the extensions introduced by Yun *et al.* , which are also slightly modified in the proposed system.

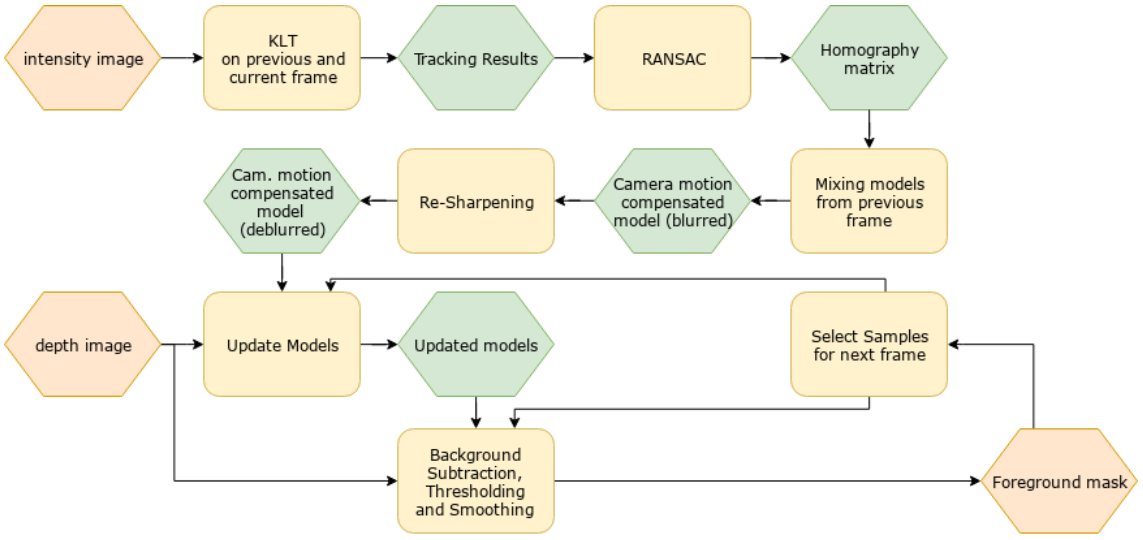

Figure 2: Concept of the proposed system visualized as event driven process chain. Hexagons represent information which are the input or output of processing stages (yellow rectangles). Based on the intensity image and the depth image, a binary foreground mask is calculated (orange hexagons).

Figure 2 provides an overview of the proposed system. The system's procedure can be outlined as follows: First, the camera motion is estimated using intensity images. By applying this transformation, the background model from the previous frame is transformed so it can

be used in the current frame (section 3.2). Next, the background model is updated with the current depth image (section 3.1). Subsequently, background subtraction is applied in order to find foreground objects (section 3.4).

One of the most relevant adaptations is the usage of depth images for the background model. Also in contrast to *MCD5.8*, the update rate of the model, the variance and the plausibility are treated in a decoupled manner (section 3.1). Apart from that, one of the main contributions of this work is an anti-blurring scheme that keeps the background model sharp (section 3.3).

3.1 Two Local Gaussian Background Models with Age in a Low Resolution

Similar to an image, a background model maps spatial and visual information. For a certain region, a background model should provide information about the appearance of the static environment. As depth images are used to model to the background, the appearance of the static environment is expressed in terms of the distance of objects to the camera. Gaussian models are employed to represent the background depth. A Gaussian model consists of a mean and a variance, where the mean value represents the average depth in the covered region.

Background modeling is performed grid-wise, locally and dually. This strategy has been introduced in *MCD5.8* [8].

Grid-wise background modeling means that it is not performed for every pixel. Instead, the image is divided in a coarse grid with equal quadratic cells and all pixels belonging to a certain cell share the same background model. This lowers the resolution of the background model and reduces computational cost. Additionally, inaccuracies in the camera motion estimation have less impact on the foreground decision (see subsection 2.2.2).

The above mentioned local property refers to the fact that the background model is only maintained for the current field of view of the camera.

Lastly, *dual* signifies the idea of maintaining two background models for every cell in the grid, but using only the more reliable model for the detection of moving objects. In the following, the more reliable background model is called the *apparent* model, whereas the

other model is referred to as the *candidate* model. While the apparent model can provide the appearance which is believed to belong to the background, the candidate model collects possible foreground observations. In systems with only one Gaussian model, the information of the latter could be discarded. When using two models per cell, initial false classifications can be rectified by simply interchanging the apparent and candidate models. This happens in *MCD5.8* when the number of observations which support the second model is higher than the number of observations that support the first. This number of observations is called the age, and the age value is stored alongside every Gaussian model. It is capped at an empirically chosen α_{max} in order to keep the model receptive.

In the following, the background model as it is used in this work is defined. Affiliation of a function or variable to the depth value is expressed by the D superscript. The input depth image $L_t^D(p)$ provides the depth value d_p for the pixel p with the coordinates (x, y) at time t through

$$L_t^D(p) = d_p \ , \quad p = (x, y) \in \Omega_p \ . \tag{1}$$

The pixel domain Ω_p contains all pixels of the input image with height n_{rows} and width n_{cols}:

$$\Omega_p = \Big\{(x, y) \in \mathbb{N}^2 \mid (0 < x < n_{rows}, 0 < y < n_{cols})\Big\} \tag{2}$$

Instead of holding a background model for every pixel p, it is only maintained for quadratic cells. Being arranged in a two-dimensional image, the cells also have an index tuple $c = (i, j)$ with i and j as the row and column index of the cell. A cell C_c at position c is a set of pixels, so that $C_c \subset \Omega_p$.

All cells are, by definition, quadratic and have an equal number of elements, therefore they share the edge length of $N \in \mathbb{N}$ pixels as well as the number of contained pixels, which is $N \cdot N$:

$$\forall C_c : |C_c| = N \cdot N \ . \tag{3}$$

The Gaussian model that is associated with every cell is denoted as the tuple

$$(\mu, \sigma, \alpha) \in Y \subset \mathbb{R}_+^3 \ , \tag{4}$$

where μ is the mean depth value, σ the variance and α the age of the model. The image $L^Y : \Omega_c \to Y$ maps each cell onto a Gaussian model using the cell's index c. The grid

domain Ω_c contains all pairs of row and column indices of the grid. For a point in time t, the background model image L_t^Y is defined as

$$L_t^Y(c) := (\mu_{t,c}, \sigma_{t,c}, \alpha_{t,c}) \ . \tag{5}$$

Up to here, the general structure of cells and Gaussian models as well as their mutual relationship have been considered. In this work, there are two images L_t^Y stored, one holding the *apparent* models, the other the *candidate* models. Both images are defined on a grid of cells that is obtained by dividing the input image into $N \times N$ squares. The grid holding the background model is constructed in a way cells do not overlap each other and fully cover the input image. The grid domain Ω_c is thus a partition of the pixel domain Ω_p. In other words, every pixel is located in exactly one cell. Partitioning in squares with edge length N is only possible if N is a factor of both the image height n_{rows} and width n_{cols}. The grid domain can then be defined as

$$\Omega_c = \left\{(i,j) \in \mathbb{N}^2 \mid 0 < i < \frac{n_{rows}}{N}, 0 < j < \frac{n_{cols}}{N}\right\} \tag{6}$$

The set of all pixels that are located in C_c after dividing the input image $L_t^D(p)$ into $N \times N$ squares assembles C_c as

$$C_c = \left\{p \in \Omega_p \mid \left\lfloor \frac{p}{N} \right\rfloor = c\right\} \ , \tag{7}$$

which allows the mapping from pixels to cells.

For each incoming frame, the components of every model $L_t^Y(c)$ are updated as:

$$\mu_{t,c} = \lambda \cdot m_{t,c} + (1-\lambda) \cdot \mu_{t-1,c} \ , \quad \mu_{0,c} = m_{0,c} \tag{8}$$

$$\sigma_{t,c} = \lambda \cdot v_{t,c} + (1-\lambda) \cdot \sigma_{t-1,c} \ , \quad \sigma_{0,c} = v_{0,c} \tag{9}$$

$$\alpha_{t,c} = \alpha_{t-1,c} + 1 \ , \quad \alpha_{0,c} = 0 \ . \tag{10}$$

In the update process described in eq. (8)(9)(10) the update rate λ determines the influence of the current observation $m_{t,c}$ with its variance $v_{t,c}$ on the existing model. A higher λ implies faster model adaption. In contrast to *MCD5.8*, in this work the update rate is kept at a fixed value independent of the model's age. The main reason for this is that depth values naturally change when the camera moves whereas the intensity, used in *MCD5.8*, remains constant.

Updating the model slower the higher the age is would hamper keeping the model up-to-date. Also, through mixing and averaging models for every frame after the camera moved, the background model tends to get blurry especially in cells where lots of observations have been incorporated. A slower update in older regions as in *MCD5.8* intensifies this blur. The reasons and consequences of this blur as well as further counteracting technique is depicted in section 3.3.

The values $m_{t,c}$ and $v_{t,c}$ from eq. (8)(9) are obtained from the pixel values in the current image. The cell's mean value $m_{t,c}$ is determined by simple arithmetic averaging of pixels which are located in the cell with index c, i.e. by

$$m_{t,c} = \frac{1}{|C_c|} \sum_{p \in C_c} L_t^D(p), \tag{11}$$

where $L_t^D(p)$ is the value in the current depth image L_t^D at pixel p. The corresponding variance $v_{t,c}$ is obtained from the pixel that shows the largest depth difference of the model's mean:

$$v_{t,c} = \max_{p \in C_c} (\mu_{t,c} - L_t^D(p))^2. \tag{12}$$

The formulation of $v_{t,c}$ is adopted from *MCD5.8* and contrasts to the standard definition of a variance in statistics.

For each input frame and for each cell, either the *apparent* or *candidate* model is updated according to equations (8)(9)(10) and the other one remains untouched. If the current observations in a cell match with the *apparent* model, the *apparent* model is updated. A match is given when two conditions are fulfilled. First, the difference between the mean in the current observation and the mean in the Gaussian model must be less than the variance of the model:

$$(m_{t,c} - \mu_{t,c})^2 < \sigma_{t,c} \ . \tag{13}$$

Second, the variance in the current observation may not exceed the variance of the Gaussian model by more than factor k_{var}:

$$v_{t,c} < k_{var} \cdot \sigma_{t,c} \ , \tag{14}$$

If the conjunction of equation 13 and 14 is false for the *apparent* model, but true for the

candidate model, the *candidate* model is updated. In case the conjunction is false for both models, the *candidate* model is reinitialized with the current observation and the age is reset to one.

The condition from equation 13 ensures that the *apparent* background rejects depth values from objects that are probably dynamic. Instead of discarding the information in this case, it is absorbed by the *candidate* model, which is either updated or reinitialized. The additional condition from equation 14 is introduced in this work to tackle issues arising at boundaries of dynamic objects. A depth edge in the current image passing through a cell causes a high variance $v_{t,c}$. If only a small ratio of the sample pixels belongs to the dynamic object, the mean $m_{t,c}$ is similar to the background model's mean $\mu_{t,c}$, so equation 13 would be fulfilled. The variance of the *apparent* model would then be increased according to the update process in equation 9. Since the dynamic object should not influence the *apparent* model, this is undesirable behavior. The high variance allows equation 13 to become true in the following frames, leading to a false incorporation of dynamic object edges. Therefore, equation 14 ensures that a model is only selected for the update process if the variance of the current observation is similar or lower than the model's variance.

After updating, the two background models are switched if the age of the *candidate* model exceeds the age of the *apparent* model multiplied with factor k_{switch} . Thus the *apparent* model holds information which is believed to belong to the static background, whereas the *candidate* model incorporates information that do not fit in the *apparent* model. These deviating information are probably caused by foreground objects or noise, however, if further observations support the conjecture that rather the candidate model holds information of the static background, models can be switched.

3.2 Camera Motion Estimation and Compensation by Transforming and Mixing the Background Model

For the most part, the camera motion estimation and compensation system is adopted from *MCD5.8*. Point pairs indicating correspondences in the previous and current frame are determined with conventional KLT on the intensity images. RANSAC is applied to find a homography that is supported by the most point pairs (see subsection 2.2.1). The thus

estimated homography is presumed to be the inverse of the camera motion. Applying the homography matrix on any pixel coordinate in the previous frame, its coordinate in the current frame can be determined. For the reasons expounded in subsection 2.2.1 this matching is not exact for every pixel. As a robust method to deal with these inaccuracies, Yi *et al.* and this work therefore choose to maintain the background model cell-wise instead of pixel-wise. The estimated homography matrix is then only applied on every cell center. Thus the cells valid for the coordinate system of the previous frame can be transformed onto the current grid. However, as most probably the matrix does not transform every cell center by a multiple of the cell's edge length, cells in the transformed grid from the previous frame intersect cells in the current grid. Making the previous background model valid for the current grid despite this issue is achieved by mixing overlapping cells. For each cell in the current grid, the background model containing the information of the previous frame is obtained by calculating the mean $\tilde{\mu}_{t-1,c}$, variance $\tilde{\sigma}_{t-1,c}$ and age value $\tilde{\alpha}_{t-1,c}$ of $L^Y_{t-1}(c)$ through interpolating intersecting cells:

$$\tilde{\mu}_{t-1,c} = \sum_{z \in \Omega_z} w_z \cdot \mu_{t-1,z} \tag{15}$$

$$\tilde{\sigma}_{t-1,c} = \sum_{z \in \Omega_z} w_z \cdot \Big(\sigma_{t-1,z} + \Big(\big(\mu_{t-1,z}\big)^2 - \big(\tilde{\mu}_{t-1,c}\big)^2\Big)\Big) \tag{16}$$

$$\tilde{\alpha}_{t-1,c} = \sum_{z \in \Omega_z} w_z \cdot \alpha_{t-1,z} \ . \tag{17}$$

The cell specific mixture weights w_z are proportional to the intersection area of the squares $\mathcal{R}_c$ and $\mathcal{R}_z$ which are defined according to figure 3:

$$w_z \propto Area\{\mathcal{R}_c \cap \mathcal{R}_z\}, \quad \sum_z w_z = 1, \quad w_z \in [0,1]\,. \tag{18}$$

In equation 15-18, z indicates the value refers to a cell from the previous grid cell domain Ω_z. Therefore, it is found on the right-hand side of the equations only. The index c signifies affiliation to the current grid, for which the background model is generated by eq. 15-18. Lastly, the tilde above the mean, variance and age value signifies that the model has been transformed and the values are thus valid in current frame. The process is executed for both the apparent and the candidate models.

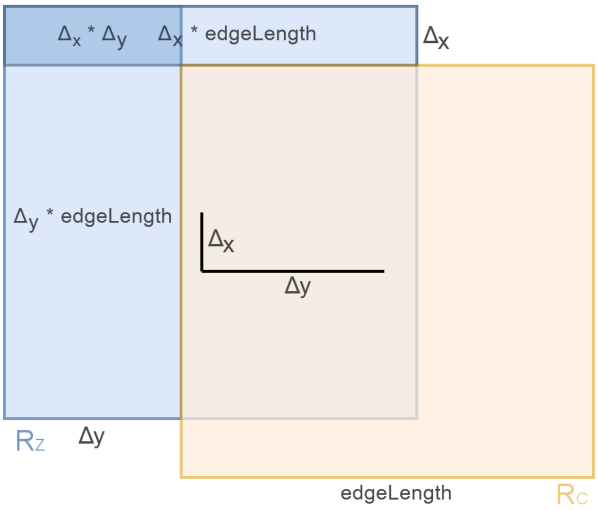

Figure 3: Calculating the overlapping area of the two equal sized squares $\mathcal{R}_z$ (blue) and $\mathcal{R}_c$ (orange) based on the distances Δ_x and Δ_y between their center coordinates. From the figure it can be easily derived that the overlapping area of the blue and orange cell equals to the entire blue square area minus $\Delta_x \cdot edgeLength$ minus $\Delta_y \cdot edgeLength$ and, to add what has been subtracted twice in the top left, plus $\Delta_x \cdot \Delta_y$.

To sum up, frame-to-frame camera motion compensation of the background model is achieved by estimating a transformation matrix with intensity images and subsequently applying it on every cell center of the grid of the previous frame to find the cells' positions in the grid of the current frame. Each cell in the current grid is influenced by the transformed previous cells proportional to their joint overlapping ratios as shown in figure 3. Given the camera motion compensated background model, current observations can be incorporated as per section 3.1. To this end, in the update functions of equations 15, 16 and 17, $\mu_{t-1,c}, \sigma_{t-1,c}$ and $\alpha_{t-1,c}$ are replaced through their camera motion compensated versions $\tilde{\mu}_{t-1,c}, \tilde{\sigma}_{t-1,c}$ and $\tilde{\alpha}_{t-1,c}$.

3.3 Anti-Blurring Technique to Preserve Acuity of Background Model

A prevalent issue that comes with the camera motion compensation technique in *MCD5.8* and section 3.2 is blurring. Through mixing adjacent background models after every frame

when the camera has moved, the background model gradually loses its sharpness. When the same region is observed for a certain time and the camera meanwhile makes small movements, edges are no longer visible in the background model. This effect is shown in Figure 4. Objects in the background model change their appearance as nearby observations, that are actually not part of the object, incorporate into the model at the position of the object. Edge regions around actually static objects are thus incorrectly classified as foreground. This is because the depth values in the current image differ from the value in the background model. Therefore, with a blurred background model, motion detection by background subtraction becomes erroneous.

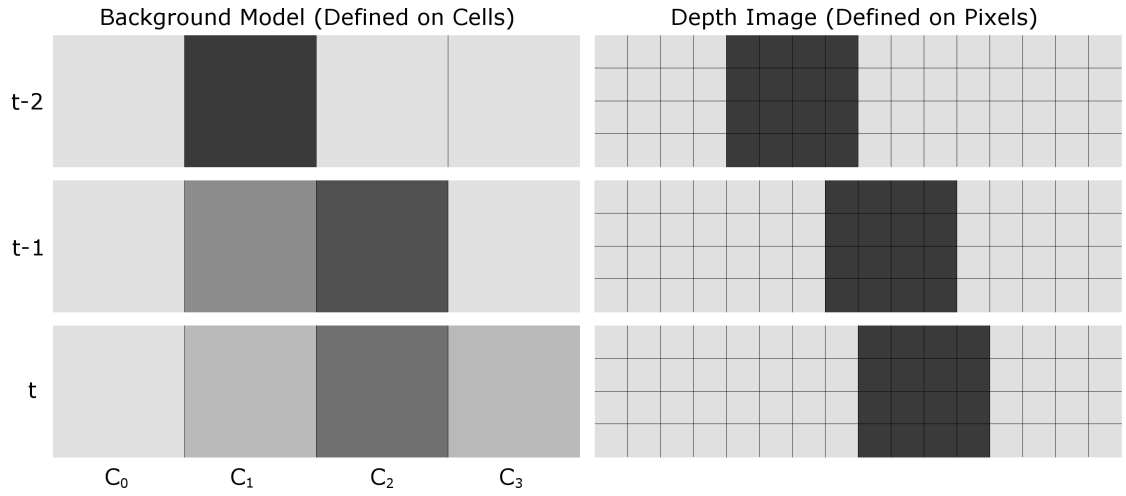

Figure 4: Schematic visualization of the blur effect. A section covering four cells of the background model is selected to show the emergence of blurs while mixing cells in a minimal example. The measured distance of objects increases with the brightness. At time $t-2$, the artificially created scene can be thought of showing a wall (light gray) and a pillar in front of it, where the depth value of the pillar is significantly lower (dark gray). The camera moves to the left from time $t-2$ to $t-1$ as well as from $t-1$ to t. As a consequence, the captured images apparently move to the right. Adding the motion over both frames, the pillar has moved in the depth image by 4 pixels, corresponding to one cell edge length. Hence, at time t the background model is expected to show the pillar in cell C_2 and the wall in all other cells. However, the bottom row shows that the dark gray of the pillar has diffused and affected cell C_1 and C_3.

This work proposes a technique to counteract the blurring visualized in Figure 4. The main idea is to measure and maintain the degree of blur for every cell and consequently prevent

unblurred cells from being contaminated with blurred cells. To this end, a blur compensation value is calculated for each cell. After the camera motion has been compensated as in section 3.2, the blur compensation value is added to the mean value of the respective apparent model. Although the candidate model also experiences the blur effects when mixing cells, blur compensation is not applied on the candidate model in order to save computation time. Omitting the candidate model is reasonable since the apparent model is always the model with the higher age and blur effects thus concentrate on the apparent model.

3.3.1 Derivation of the Blur Effect in Exemplary Scenario

Suppose a situation as depicted in Figure 4. No dynamic objects interfere the scene. The observed scene contains a wall in distance d_{wall} and a pillar in front of it with distance d_{pillar} to the camera. To simplify the modeled scenario, the grid consists of (1×4) cells only. The vertical pillar covers the cell C_1 in the second column with coordinates $c_1 = (0, 1)$, so that $\mu_{t-2,c_1} = d_{pillar}$. All other mean values in the grid belong to the wall and hence equal d_{wall}.

Three frames are envisaged to understand the blur effect. From the first to the second frame, camera motion estimation yields that the entire image has moved by n_{t-2}^{t-1} pixels to the right. From the second to the third frame, the more recent image appears to be shifted by n_{t-1}^{t} pixels to the right. A scenario is envisaged where both displacements add up to the cell's edge length (see equation 3):

$$n_{t-2}^{t-1} + n_{t-1}^{t} = N. \tag{19}$$

Comparing the first and the third frame, the pillar should consequently have moved by one cell to the right, so that $\mu_{t-2,c_2} = d_{pillar}$. All other cells, including C_1, should exhibit d_{wall} as their mean values. However, this is not the case. In the following, the reason for that is examined.

The mean value of a model for time $t-1$ is assembled from the mean values of the models in the transformed grid from time $t-2$. Weights are calculated corresponding to the overlapping ratio. Because the same right shift transformation has been applied on every cell, for each cell C_c at $t-1$ the weight of its left-handed cell from $t-2$ is equal and corresponds to

$$w_{l,t-2} = \frac{n_{t-2}^{t-1}}{N}\ , \tag{20}$$

where the coordinate l left of c is given by $l = c - (0, 1)$. The weight w_c of each cell with the same indices in the frame from $t - 2$ is

$$w_{c,t-2} = \frac{N - n_{t-2}^{t-1}}{N} = 1 - w_{l,t-2}. \tag{21}$$

For the cell C_0 at position $(0, 0)$, C_1 at $(0, 1)$ and C_2 at $(0, 2)$, the mean values are obtained by

$$\tilde{\mu}_{t-2,c_1} = w_{l,t-2} \cdot \mu_{t-2,c_2} + (1 - w_{l,t-2})\mu_{t-2,c_1}, \tag{22}$$

$$\tilde{\mu}_{t-2,c_2} = w_{l,t-2} \cdot \mu_{t-2,c_1} + (1 - w_{l,t-2})\mu_{t-2,c_2} \ . \tag{23}$$

Expressed in terms of the depth values of the wall and the pillar, this is

$$\tilde{\mu}_{t-2,c_1} = w_{l,t-2} \cdot d_{wall} + (1 - w_{l,t-2})d_{pillar} \tag{24}$$

$$\tilde{\mu}_{t-2,c_2} = w_{l,t-2} \cdot d_{pillar} + (1 - w_{l,t-2})d_{wall}. \tag{25}$$

Through the right shift of the images due to the camera motion, each cell at time $t - 1$ incorporates the ratio $w_{l,t-2}$ of its left-handed cell from time $t - 2$. Up to here the cell states are as expected. The scene is static, therefore the current observations do not change the background model, so that $\forall c : \mu_{t-1,c} = \tilde{\mu}_{t-2,c}$. Otherwise equation 8 would be used to obtain : $\mu_{t-1,c}$ from $\tilde{\mu}_{t-2,c}$.

Next, from $t - 1$ to t, the image appears to have moved to the right by n_{t-1}^{t} pixels. Under the scenario constraint of equation 19, the weights $w_{l,t-1}$ and $w_{c,t-1}$ can be expressed through

$$w_{l,t-1} = 1 - w_{l,t-2} \tag{26}$$

$$w_{m,t-1} = 1 - w_{m,t-2} = w_{l,t-1}. \tag{27}$$

Mixing models according to equation 15, the mean values for time t are thus

$$\tilde{\mu}_{t-1,c_1} = (1 - w_{l,t-2})\mu_{t-1,c_0} + w_{l,t-2} \cdot \mu_{t-1,c_1}, \tag{28}$$

$$\tilde{\mu}_{t-1,c_2} = (1 - w_{l,t-2})\mu_{t-1,c_1} + w_{l,t-2} \cdot \mu_{t-1,c_2}. \tag{29}$$

Substituting expressions using equations (24)(25), the means valid for the frame at time t

are given by

$$\tilde{\mu}_{t-1,c_1} = (1 - w_{l,t-2})d_{wall} + w_{l,t-2}(w_{l,t-2} \cdot d_{wall} + (1 - w_{l,t-2}) \cdot d_{pillar}), \tag{30}$$

$$\tilde{\mu}_{t-1,c_2} = (1-w_{l,t-2})(w_{l,t-2}\cdot d_{wall}+(1-w_{l,t-2})d_{pillar})+w_{l,t-2}(w_{l,t-2}\cdot d_{pillar}+(1-w_{l,t-2})d_{wall}). \tag{31}$$

Reducing (30)(31), it turns out that

$$\tilde{\mu}_{t-1,c_1} = (1 - (w_{l,t-1})(w_{l,t-2}))d_{wall} + (w_{l,t-1})(w_{l,t-2})d_{pillar} \tag{32}$$

and

$$\tilde{\mu}_{t-1,c_2} = (1 - 2(w_{l,t-1})(w_{l,t-2}))d_{pillar} + 2(w_{l,t-1})(w_{l,t-2})d_{wall}. \tag{33}$$

However, since all objects in the image have actually moved by exactly one cell, the value for $\tilde{\mu}_{t-1,c_2}$ should be $1.0 \cdot d_{pillar}$, completely containing the pillar. Equally, $\tilde{\mu}_{t-1,c_1}$ should actually attain $1.0 \cdot d_{wall}$ because the current image shows wall in the entire cell region. Even the cell in the fifth column is unequal to d_{wall} because it absorbs the ratio $w_{l,t-1}$ of μ_{t-1,c_2} even though the pillar has never been there. The difference between the target and actual mean value of C_3 and C_1 (equation 30) can be understood as

$$\Delta\mu_{t,c_1} = \Delta\mu_{t,c_3} = d_{wall} - \tilde{\mu}_{t-1,c_1} \cdot w_{l,t-1} \tag{34}$$

and for the forth column in equation 31 as

$$\Delta\mu_{t,c_2} = d_{pillar} - \tilde{\mu}_{t-1,c_2} \cdot w_{l,t-1} \tag{35}$$

so that

$$\Delta\mu_{t,c_1} = \Delta\mu_{t,c_3} = (w_{l,t-1})(w_{l,t-2}) \cdot (d_{wall} - d_{pillar}), \tag{36}$$

and

$$\Delta\mu_{t,c_2} = -2((w_{l,t-1})(w_{l,t-2}) \cdot (d_{wall} - d_{pillar})) \tag{37}$$

Note that the signs of $\Delta\mu_{t,c_1}$ and $\Delta\mu_{t,c_2}$ differ and the sum of the target-actual difference over all cells is zero, implying it changes only the local appearance of the background model,

which is typical for blurs [32]:

$$\Delta\mu_{t,c_2} = -(\Delta\mu_{t,c_1} + \Delta\mu_{t,c_3}) \ , \quad \sum_c \Delta\mu_{t,c} = 0 \tag{38}$$

Up to here, it was shown how after two frames the actual value deviates from the desired target value. The origin of the blur effect in the mixture process becomes apparent. Cells partially comprise the original state of other cells. Considering this effect on the whole image, the background model becomes blurry over time.

3.3.2 Generalization of the Example Using an Intermediate Blur Value

A general formula is to be determined so that the blur effects can be quantized beyond the scenario above. Equations 36 and 37 show that the blur effect depends on weights in earlier mixture processes as well as on the original mean values of cells. Although the weights that formulate $\Delta\mu_{t,c_1}$ and $\Delta\mu_{t,c_2}$ are the same, the expected results differ in both magnitude and sign.

In the following, the target-actual difference is represented by the sum of two functions. The primary function, $L_t^{\Delta\mu,prim}(c)$, is able to represent $\Delta\mu_{t,c_1}$ and $\Delta\mu_{t,c_3}$. It occurs on every depth edge when the image is transformed by an amount unequal to any multiple of the cell's edge length N. The secondary function, $L_t^{\Delta\mu,sec}(c)$, approximates the effect leading to $\Delta\mu_{t,c_2}$. It is observed that $L_t^{\Delta\mu,sec}(c)$ is only necessary when a single-row or single-column depth line is involved. The reason for that can be comprehended when considering the composition of $\tilde{\mu}_{t-1,c_2}$ in equation 29.

Given only the Gaussian models from the previous frame as it is the case in the standard mixture process, the blur target-actual difference cannot be deduced. This becomes clear through equations 36 and 37, where weights and mean values from time $t-2$ are involved. A blur measure is introduced whose formulation is based on the composition of the mean value through mixing cells from time $t-2$. In the scenario examined above, the mean value is influenced by two different observations from time $t-2$ with the respective weights $w_{l,t-2}$ and $w_{m,t-2}$. The resulting mean value for $t-1$ is thus some value in between. This is already a blur, however, since the resolution of the grid is fixed, it is not avoidable at this point.

Instead, the blur can be quantized in a value denoted as $\beta_{t-1,c}$ by determining the value to which the result differs from the original component with the highest weight:

$$\beta_{t-1,c} = \mu_{t-2,z_{max}} - \tilde{\mu}_{t-2,c}, \quad z_{max} = \underset{z \in \Omega_z}{\arg\max}\, w_z, \tag{39}$$

where the notation is adopted from equation 15. In words, $\beta_{t-1,c}$ represents the value that must be added to $\mu_{t-1,c}$ to obtain the original mean of the cell that had the largest influence in the mixing procedure.

With $\beta_{t-1,c}$ from the previous mixture process, the difference between target and actual values in $\Delta\mu_{t,c_1}$ and $\Delta\mu_{t,c_3}$ can be explained. In equation 28, cell C_0 is still unblurred because in the previous frames it has not incorporated any different mean value. Its blur value β_{t-1,c_0} is therefore zero. In contrast, cell c_1 has been composed out of different means and therefore exhibits a blur value β_{t-1,c_1} according to equation 39:

$$\beta_{t-1,c_1} = \begin{cases} d_{pillar} - \tilde{\mu}_{t-2,c_1} = (w_{l,t-2})(d_{pillar} - d_{wall}) \text{ if } z_{max} = c_1 \\ d_{wall} - \tilde{\mu}_{t-2,c_1} = (1 - w_{l,t-2})(d_{wall} - d_{pillar}) \text{ if } z_{max} = c_0 \end{cases} \tag{40}$$

Target-actual difference when mixing blurred and unblurred cells

The primary target-actual difference function $L_t^{\Delta\mu,prim}$ is based on the inequality of mixed cells with respect to blurriness.

To find blurred and unblurred cells in general, the influencing cells Ω_z are split in two equal sized subsets. If $|\Omega_z|$ is uneven, one subset contains one element more than the other. The split of Ω_z is performed by the blurriness $|\beta_{t-1,z}|$, resulting in a subset containing all lower blurred cells and a subset containing the higher blurred cells. The values and weights of the models in each subset are averaged in order to obtain the two artificial cells u and b, where u contains the less blurred model and b the stronger blurred model.

Then, the difference of the absolute blur values can be calculated by

$$\delta_{\beta_t,c} = |\beta_{t-1,b}| - |\beta_{t-1,u}|. \tag{41}$$

As $\beta_{t-1,b}$ contains the cells with the stronger blurred cells, $\delta_{\beta_t,c}$ is always positive. For further

calculations, also the mean difference between blurred and unblurred cell is determined:

$$\delta^{ub}_{\mu_{t,c}} = \mu_{t-1,b} - \mu_{t-1,u} \ . \tag{42}$$

The primary blur effect $L_t^{\Delta\mu,prim}(c)$ is now to be expressed in terms of the blur difference $\delta_{\beta_t,c}$. Exemplary for C_1, δ_{β_t,c_1} (equations 40,41) and $\Delta\mu_{t,c_1}$ (equation 36) differ by a factor only:

$$\frac{\Delta\mu_{t,c_1}}{\delta_{\beta_t,c}} = \begin{cases} 1 - w_{l,t-2} & \text{if } z_{max} = c_1 \\ w_{l,t-2} & \text{if } z_{max} = c_0 \end{cases} \tag{43}$$

The generalized formulation for this missing factor is derived in the following. The scenario from subsection 3.3.1 showed that a deviation from the expected mean value occurs when a certain cell influences another cell twice. A gap between the actual value and the target value could be observed that depends on both the weights from the second last and the previous frame. The blur value $\beta_{t-1,c}$ implicitly includes a weight from the second last frame. Equation 39 shows that the blur value calculation is based on the cell from $t-2$ with the highest influence, which is $w_{z_{max},t-2}$. Because of the cell partition in two artificial cells, the cell with index z_{min} complements z_{max} so that $\Omega_z = \{z_{min}, z_{max}\}$. By replacing $\tilde{\mu}_{t-2,c}$ by its composition according to equations 22 and 23, it can be shown that the weight which is included in $\beta_{t-1,c}$ is the weight of the cell with the lower influence, namely $w_{z_{min},t-2} = (1 - w_{z_{max},t-2})$:

$$\beta_{t-1,c} = \mu_{t-2,z_{max}} - \tilde{\mu}_{t-2,c} = (1 - w_{z_{max},t-2})\delta^{minmax}_{\mu_{t-1,c}} \ . \tag{44}$$

In equation 44, $\delta^{minmax}_{\mu_{t-1,c}}$ is the difference of the influencing means for the composition of cell C_c at time $t-1$:

$$\delta^{minmax}_{\mu_{t-1,c}} = \mu_{t-2,z_{max}} - \mu_{t-2,z_{min}} \ . \tag{45}$$

Equation 44 showed that $\beta_{t-1,c}$ is proportional to $w_{z_{min},t-2}$. The cell from time $t-2$ with the lower influence for C_c is likely to have a larger overlap with another cell and thus propagates its mean to time $t-1$. As aforementioned, when this cell again influences the already blurred cell in the mixture process at time t, the target-actual difference emerges. Subsection 3.3.1 and equation 36 yield that the target-actual difference is a factor of both the weights from $t-2$ and $t-1$. Given $\beta_{t-1,c}$, the weight that is missing is the weight from $t-1$. The

influencing cell holding the mean that had less influence in the previous mixture process is found by checking if the sign of the blur value of the stronger blurred cell contradicts the sign of the mean difference:

$$w_{z_{compl},t-1} = \begin{cases} w_{b,t-1} & \text{if } sgn(\beta_{t,b}) = -sgn(\delta^{ub}_{\mu_{t,c}}) \\ w_{u,t-1} & \text{otherwise .} \end{cases} \tag{46}$$

The blur difference $\delta_{\beta_t,c}$ is based on the stronger blur $\beta_{t-1,b}$, containing a weight from time $t-2$. When $\delta_{\beta_t,c}$ becomes proportional to $w_{z_{compl},t-1}$, it can formulate $\Delta\mu_{t,c_1}$ and $\Delta\mu_{t,c_3}$ from equation 36. The primary blur effect can thus be explained through

$$L_t^{\Delta\mu,prim}(c) = -sgn(\delta^{ub}_{\mu_t,c}) \cdot w_{z_{compl},t-1} \cdot \delta_{\beta_t,c} \tag{47}$$

with the sign function $sgn(\cdots)$ choosing the sign so that $L_t^{\Delta\mu,prim}(c)$ favors the mean of the unblurred cell.

Being able to determine the difference between target and actual value through $L_t^{\Delta\mu,prim}(c)$, the system can counteract the blur effect for cells C_1 and C_3 in the example.

Target-actual difference when mixing two blurred cells

However, for cell C_2, there is another logic necessary to derive $\Delta\mu_{t,c_2}$ from equation 31. Equation 29 shows that μ_{t-1,c_3} and μ_{t-1,c_4} are the influencing means. For cell C_2, the blur value β_{t-1,c_2} has been calculated according to equation 39 as

$$\beta_{t-1,c_2} = d_{wall} - \tilde{\mu}_{t-2,c_2} = 0.3(d_{wall} - d_{pillar}). \tag{48}$$

Considering β_{t-1,c_1} from equation 40, the influencing cells have thus the same absolute blur value. Hence, the blur difference δ_{β_t,c_2} is zero. The value of $\Delta\mu_{t,prim}(c_2)$ can therefore not explain $\Delta\mu_{t,c_2}$ or any other target-actual difference when both cells are blurred. However, the effect of $\Delta\mu_{t,c_2}$ only emerges when the pillar covers a single column. If the pillar was larger than one column by also covering c_2, all mixture components in equation 28 would exhibit d_{pillar} as their mean values. No blur emerges thus, therefore β_{t-1,c_1} would be zero. The blur value for C_2, β_{t-1,c_2}, remains unchanged under this changed conditions. This way there is only cell C_2 blurred at time $t-1$, and the technique from equation 47 can be applied to

explain $\Delta\mu_{t,c_2}$.

Becoming capable of modeling the blur induced by single row or single column objects requires the determination of the common blur of both cells. The common blur is given by the less blurred cell u:

$$\beta_{t,c}^{com} = 2 \cdot |\beta_{t-1,c_u}|. \tag{49}$$

Analogous to equation 47, the absolute value of the wanted $\Delta\mu_{t,sec}(c)$ is obtained by multiplying $\beta_{t,c}^{com}$ with the weight of the unblurred cell:

$$L_t^{\Delta\mu,sec}(c) \approx s^{sec} \cdot w_u \cdot \beta_{t,c}^{com} \tag{50}$$

As the approximate sign indicates, as against equation 47 this term is not exact in general. It is only based on the idea to strengthen the edge when two blurred cells are mixed, which is typical when single column or single row objects occur. The sign of $\Delta\mu_{t,sec}(c)$ is chosen so that the zero sum property from equation 38 is preserved. To this end, neighboring $\Delta\mu_{t,prim}(c)$ are considered that have been determined yet. If the sum of $\Delta\mu_{t,prim}(c)$ values in all cells with directly adjacent coordinates $\mathcal{S}_{adj}^c \subset \Omega_c$ is positive, $\Delta\mu_{t,sec}(c)$ is assigned a negative sign and vice versa, so that

$$s^{sec} = -sgn(\sum_{c_{adj} \in \mathcal{S}_{adj}^c} L_t^{\Delta\mu,prim}(c_{adj})) \ . \tag{51}$$

Combining the two concepts from equation 47 and 50, the target-actual differences can be generalized in the blur compensation image $L_t^{\Delta\mu}$:

$$L_t^{\Delta\mu} \approx L_t^{\Delta\mu,prim} + L_t^{\Delta\mu,sec} \ . \tag{52}$$

From frame to frame, the blur values are inherited analogously to equation 15. They are increased using equation 39 and decreased when the blur compensation based on $L_t^{\Delta\mu}(c)$ is applied, which is focused in subsection 3.3.3.

This section showed how the blur effects can be formulated in a general manner. When mixing unblurred and blurred cells, a closed-form derivation of the compensation value is possible (equation 47). With equation 52, the blur can be approximated when both cells have a non-zero blur value.

3.3.3 Modification of the Model Mixture Process with a Blur Neutralization Scheme

Seizing the findings from the previous section, this work includes a neutralization scheme which counteracts the discovered gradual blur. After mixing models and before updating them, the mixed means $\tilde{\mu}_{t-1,c}$ can be modified so that blur effects are neutralized.

First, the background model from equation 4 is extended to

$$Y = \{\mu, \sigma, \alpha, \beta\}\,. \tag{53}$$

Adding the blur value β to each cell's model allows recording blurs when they are inevitably induced and reacting at a later instance of time.

The mixing rule corresponds to the averaging of the mean in equation 15:

$$\tilde{\beta}_{t-1,c} = \sum_{z \in \Omega_z} w_z \cdot \beta_{t-1,z}\ , \quad \beta_{0,c} = 0 \tag{54}$$

Given the set of influencing cells Ω_z and their corresponding weights, it is checked if blurred cells exist that would further spread through the mixture process. As this causes the image to blur increasingly, the anti-blurring technique is applied here.

With equation 52, the blur effects can be explained for the scenario where two cells are mixed and the current observations do not change the model, so that $\tilde{\mu}_{t-2,c} = \mu_{t-1,c}$ and $\tilde{\mu}_{t-1,c} = \mu_{t,c}$. However, in real scenarios, both presumptions are not given. In most cases the model of a cell is assembled out of four overlapping models. When there is a zoom, even nine overlapping cells are possible. To make the idea yet applicable, the influencing cells are separated by their blur value as described in subsection 3.3.2. Equation 52 and its derivation can now be understood with respect to this two artificially created models. The primary blur effect which is to be compensated is given by equation 47. The secondary effect is approximated in equation 50. Equation 52 adds them up, multiplies this sum with the weight of the unblurred cell and yields thus the blur compensation image $L_t^{\Delta\mu}(c)$.

This value of $L_t^{\Delta\mu}(c)$ is calculated for the entire grid domain.

Mixing models is executed according to equation 54 standardly and without consideration of $L_t^{\Delta\mu}(c)$. After mixing models, the blur value is updated by adding the blur caused by the

most recent mixing. Adopting k_{max} and the logic from equation 39, the blur value is updated by

$$\beta_{t,c} = \tilde{\beta}_{t-1,c} + (\mu_{t-1,z_{max}} - \tilde{\mu}_{t-1,c}). \tag{55}$$

Finally, when all cells for the current frame have been assembled from the overlapping cells of the previous frame, the blur compensation image $L_t^{\Delta\mu}(c)$ is used. It was calculated beforehand following equation 52. The means determined as of yet by mixing are overwritten as per

$$\tilde{\mu}'_{t-1,c} = \tilde{\mu}_{t-1,c} + L_t^{\Delta\mu}(c). \tag{56}$$

At the same time, the blur of the respective cell is reduced by the amount of the impact of the blur compensation on the model's mean:

$$\beta'_{t,c} = \beta_{t,c} - sgn(\beta_{t,c}) \cdot L_t^{\Delta\mu}(c). \tag{57}$$

As $\beta_{t,c}$ is signed, the sign function ensures that $|\beta'_{t,c}| < |\beta_{t,c}|$, which means after blur compensation, the cell should be less blurred than before. The blur compensated mean value $\tilde{\mu}'_{t-1,c}$ becomes $\mu_{t,c}$ through updating with the current observations as per equation 15.

For efficiency reasons, the blur compensation technique is applied on the apparent model only. However, it is also sensible to apply it on the candidate model.

3.4 Obtaining the Foreground Mask by Applying Background Subtraction

The next step in each frame after camera motion compensation, deblurring and model updating is background subtraction. The basic logic for background subtraction is borrowed from *MCD5.8* [8] and adapted to the needs of depth images. Using the apparent models which are believed to contain uncontaminated background information, the difference between the current depth image and the current background model is calculated. Since the resolution of the input depth image is greater than the resolution of the grid containing the Gaussian models, the background model is extended to the image domain. Every pixel p in the newly

constructed image $L_t^{Y_{ext}}(p)$ is assigned the *apparent* model of the cell containing p:

$$\forall p \in C_c : L_t^{Y_{ext}}(p) = L_t^Y(c) \tag{58}$$

Background subtraction can now be carried out and the differencing image $L_t^{\Delta D}$ is given by

$$\forall p \in \Omega_p : L_t^{\Delta D}(p) = L_t^D(p) - \mu_{t,p} \ , \tag{59}$$

where $\mu_{t,p}$ is the mean of $L_t^{Y_{ext}}(p)$. For each pixel, a foreground probability can be calculated relative to the variance of the underlying cell. If the variance of the background model is high, the same $L_t^{\Delta D}(p)$ corresponds to a lower foreground probability because the uncertainty is larger. Seizing this idea, the foreground probability is calculated by

$$L_t^{fgprob}(p) = \frac{(L_t^{\Delta D}(p))^2}{\sigma_{t,p}} \cdot \Theta(-L_t^{\Delta D}(p)), \tag{60}$$

where $\Theta(\cdots)$ is the Heaviside step function which ensures the foreground probability is zero if the depth difference is positive. Only pixels which are in the current frame closer to the camera than in the background model are thus detected. $L_t^{fgprob}(p)$ is not scaled from zero to one. However, the greater $L_t^{fgprob}(p)$, the higher is the probability that p belongs to a dynamic object. All values less than one indicate the current depth observation fits in the background model according to equation 13 and therefore the pixel likely belongs to a static object. To obtain a binary mask from L_t^{fgprob}, simple thresholding can be applied:

$$L_t^{fgbin}(p) = L_t^{fgprob}(p) > \theta_{fg}, \tag{61}$$

where the detection threshold θ_{fg} regulates over- or underdetection of foreground objects.

3.5 Efficiency Enhancement through Foreground Probability based Sampling

Instead of performing the model update and background subtraction with every pixel in the current image, only a few pixels are selected. This lowers computational cost. The strategy to choose these pixels is similar to the sampling map from Yun *et al.* [7]. Based

on the binary foreground mask output L_t^{fgbin} from the previous frame, an initial foreground probability image $L_t^{P_{fg}}(p)$ for the current frame is generated. When pixels have a high initial foreground probability, they are selected as samples. The samples, denoted as $\mathcal{S}_{smpl}^p$, replace C_c when updating the model (equations 11 and 12) as well as Ω_p when applying background subtraction (equation 59).

The sample set $\mathcal{S}_{smpl}^p$ is obtained by extracting pixels with a high initial foreground probability $L_t^{P_{fg}}$ as

$$\mathcal{S}_{smpl}^p = \left\{ p \in \Omega_p \mid L_t^{P_{fg}}(p) > \theta_{smpl} \right\} \cup \mathcal{S}_{rnd}^p \ , \tag{62}$$

with Ω_p as the set of all pixels. In order to detect newly appeared dynamic objects, $\mathcal{S}_{smpl}^p$ does also contain $\mathcal{S}_{rnd}^p$, which is a set obtained by randomly selecting 5% [7] of the pixels in Ω_p.

From $L_t^{fgbin}(p)$, the initial foreground probability $L_t^{P_{fg}}(p)$ is derived based on the assumption that the dynamic objects move smoothly and can thus be found at time t near the position where they have been at time $t-1$. This expresses the presumed spatio-temporal property of dynamic objects. Following [7], the spatial property image is calculated by

$$L_t^{spat}(p) = (1-\lambda_{smpl}) L_t^{spat}(p) + \lambda_{smpl} \frac{1}{|neigh(p)|} \sum_{j \in neigh(p)} L_t^{fgbin}(p), \tag{63}$$

where λ_{smpl} is the update rate and the function $neigh(p)$ yields all pixels inside a 5×5 window with p as its center. The temporal property is obtained by

$$L_t^{tmp}(p) = (1-\lambda_{smpl}) L_t^{tmp}(p) + \lambda_{smpl} \cdot L_t^{fgbin}(p). \tag{64}$$

A high temporal property $L_t^{tmp}(p)$ signifies that a dynamic objects has been frequently detected at pixel p over the last frames. When the spatial property $L_t^{spat}(p)$ exhibits a high value, many pixels in the neighborhood of p were classified as foreground in the previous frames. The initial foreground probability is calculated based on these two properties:

$$L_t^{P_{fg}}(p) = L_t^{tmp}(p) \cdot L_t^{spat}(p) \tag{65}$$

According to equation 62, the sample set $\mathcal{S}_{smpl}^p$ is assembled by selecting all pixels with an initial foreground probability $L_t^{P_{fg}}(p)$ higher than θ_{smpl}.

The initial foreground probability $L_t^{P_{fg}}(p)$ exhibits a higher value in regions with previous foreground detections, so that samples are selected densely in these regions. On the contrary, in regions without prior foreground detections, samples are only sparsely selected based on the randomly chosen pixels $\mathcal{S}_{rnd}^p$.

For all pixels that are no samples, the result of the background subtraction is defined as zero. Consequently, the pixel is classified as background. The advantageous effect thereof is a reduction of false detections in regions where no dynamic objects have been located in the previous frame. On the other hand, when the dynamic object covers image regions newly, only those pixels are detected which are contained in the randomly chosen set $\mathcal{S}_{rnd}^p$.

Therefore, to recover the dense detection of these areas, this work proposes post-processing of the foreground mask. After the samples for the next frame have been determined, a box-filter with kernel size $Ksize$ is applied on the binary foreground mask L_t^{fgbin} as it was obtained in equation 61. Blurring with a box filter removes the binary characteristic of the image. Hence, the final output mask is generated by cutting the box-filtered mask to a low threshold of $5\% \cdot |Ksize|$. For example, if $Ksize = 7 \times 7$, the threshold is $5\% \cdot 49$ assuming the binary mask L_t^{fgbin} is encoded using 1 for true and 0 for false.

4 Implementation using *C++* and *OpenCV*

The proposed method from chapter 3 is implemented using the programming language *C++*. The computer vision library *OpenCV 3.3.1* provides algorithms and data structures that reduce the amount of custom-written algorithms necessary.

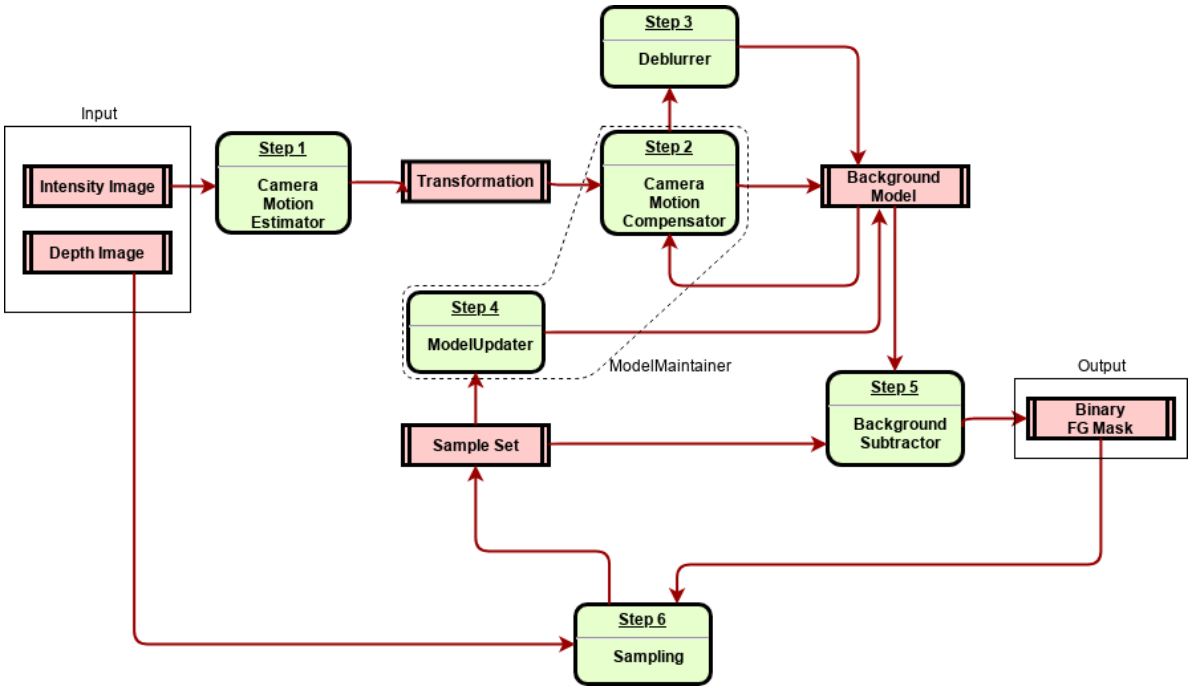

Figure 5: Data flow diagram showing the functional modules in green rounded rectangles and data structures in red double-edged rectangles.

Key components for the elaborated system from chapter 3 are identified. The strategy for the extraction of modules primarily follows the separation of concerns principle. Figure 5 shows the data flow between those modules resulting in the binary foreground segmentation mask. *C++* classes and the overall software architecture are chosen referring to Figure 5.

For every frame, a coordinator function (see subsection 4.2.8) prompts the respective modules to execute the action they are responsible for. The coordinator provides the required input for the module and collects the result. Given the result from one module, the next module in

the procedure order is called. Inputs and outputs as well as the procedure order are visualized in Figure 5.

In section 4.1, the data structures being used are presented. The functional modules performing operations on these data structures are inspected in section 4.2.

4.1 Data Containers

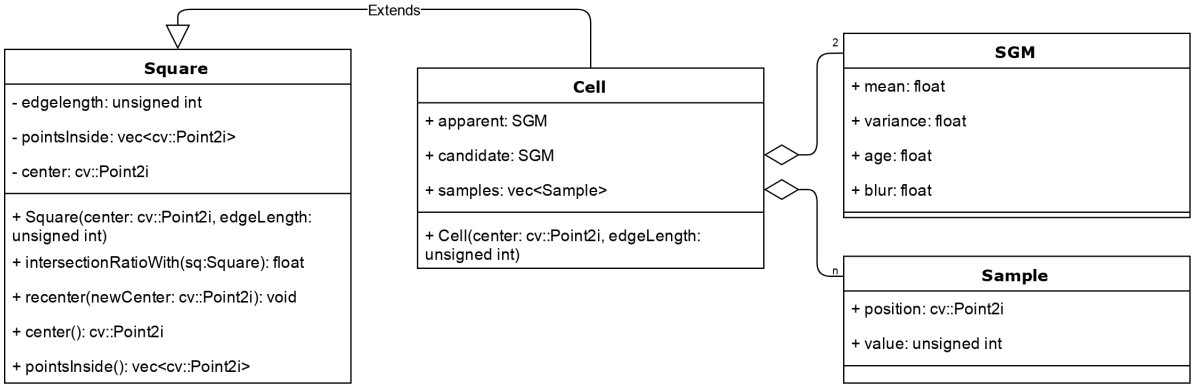

Figure 6: The cell class and its members' types that are used for cell-wise background modeling. The geometric configuration of cells in the grid is prescribed by the *Square* class. A Gaussian model is represented by the class *SGM*. The cell also holds a set of samples, which are formed from their pixel coordinates and the value at this pixel

Besides the core functional classes, data container classes are created to express the data structures that contain the system's information. First of all, a custom class *Image2D* for two-dimensional images is defined. It is created as a template class and can take objects of any type as its pixel values. This allows representing the grid containing the background model as an image of cells, where an object of type *Cell* is defined as shown in Figure 6. Consequently, the cell contains the apparent and candidate Gaussian models and a sample set. Moreover, it inherits from *Square*. The *Square* class defines a geometric square by a given center point and its desired edge length. The coordinates of the center point as well as the edge length are integers. Thereby, the shape of the square outlines a set of points with integer coordinates, which are stored in *pointsInside* in Figure 6. The member function *intersectionRatioWith(Square)* is relevant in the mixture process when the overlapping ratios are to be determined according to equation 18.

The background model grid can now be represented by an image of type *Image2D<Cell>*, i.e. with *Cell* objects as its pixel values.

4.2 Functional Modules

In this section, implementation details of the functional components are shown. Each subsection focuses on an individual component, which is also a *C++* class each. Their role in the entire system can be comprehended best from Figure 5. Most of them are usable independently, which facilitates unit testing.

4.2.1 Model Maintainer

The model maintainer is responsible for creating and maintaining the grid containing the background model. Its constructor takes the desired edge length of the cells as an argument. Since the size of the input depth image is known globally, the number of required cells can be derived and the grid can be created as an *Image2D* of *Cell* objects. Width and height of the grid are chosen in a way they equal to the input depth image width and height when being multiplied by the cells' edge length. Hence the prerequisite for the edge length is that it is a common factor of both the width and height of the input depth image.

The model maintainer also conducts the camera motion compensation by mixing the grid as well as the model update by incorporating current observations. However, these two tasks are delegated to distinct classes explicated in subsections 4.2.5.

Figure 7 illustrates the relationship between *ModelMaintainer*, *CameraMotionCompensator* and *ModelUpdater*. The coordinator, in which the main loop for processing each frame runs, calls the *update* function of the *ModelMaintainer* passing the homography transformation estimated beforehand and the current depth image. Then, the *CameraMotionCompensator* is prompted to perform its task based on the homography. After that, the *ModelUpdater* is triggered to update the compensated model with the samples of the current depth image.

The functionality of the model maintainer can be summarized by its inputs and outputs:

Inputs Previous background model, Homography, Current depth samples

Outputs Current background model

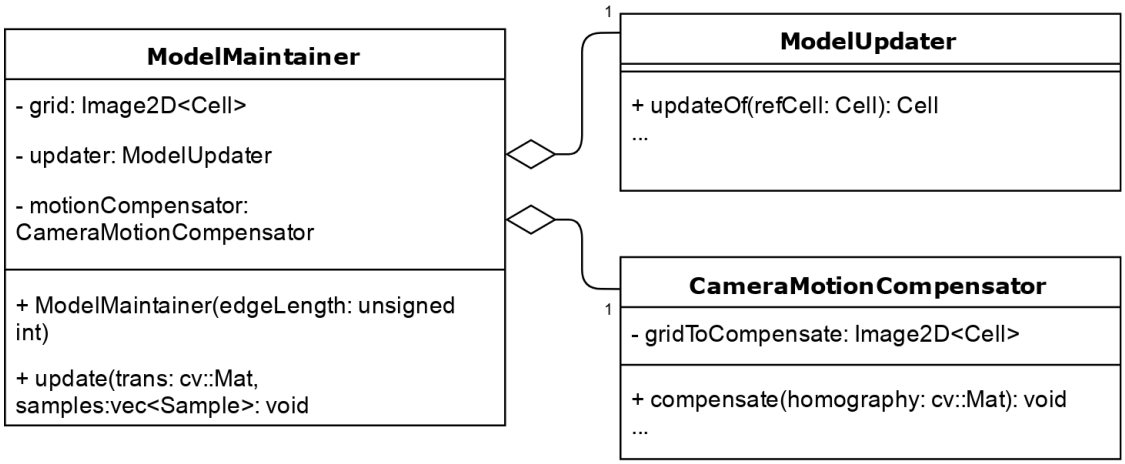

Figure 7: Class diagram showing the ownership relation between the components that change the background model. With the creation of a *ModelMaintainer* object, also a *CameraMotionCompensator* and a *ModelUpdater* object is created. When the function *update* is called, the model maintainer initiates the motion compensation and update of the model.

4.2.2 Camera Motion Estimator

Goal of the camera motion estimator class is to estimate the transformation that aligns the previous frame and the current frame. Since the transformation is calculated from two 2D images, it is a homography (see subsection 2.2.1). In contrast the background model, which is based on depth images, the camera motion estimator uses intensity images. Its inputs and outputs are therefore

Inputs Previous intensity image, Current intensity image

Outputs Homography transformation matrix.

As Figure 8 shows, the only externally accessible member function is *calcTrans*, whose arguments and return value correspond to the in- and outputs listed above. Internally, the transformation is estimated in two stages. In the first stage, covered by the function *findCorrespondingPointsWithKLT*, the KLT implemented in OpenCV is used. Before applying the actual feature tracker, points are extracted in the previous intensity image which are believed to be good features to track. OpenCV's function *cv::goodFeaturesToTrack(...)* determines these points under consideration of custom-definable constraints. Having selected features to track, they are passed to *cv::calcOpticalFlowPyrLK(...)*, which finds the corresponding points

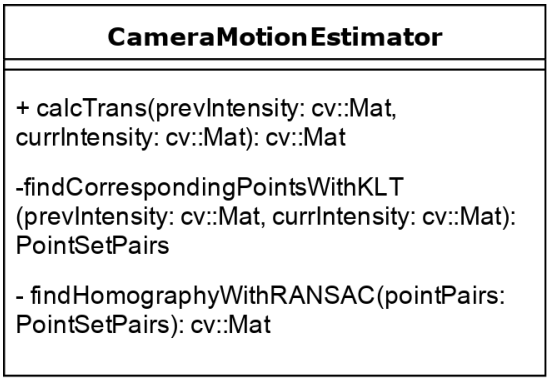

Figure 8: The camera motion estimation is internally performed in two steps: First, corresponding points are found with t KLT, then the homography is estimated based on these points with RANSAC. The type *PointSetPairs* is a pair of vectors.

in the current intensity image using the Lucas-Kanade method in pyramid-levels [33]. From an ordered set of points from the previous intensity image yielded from *goodFeaturesToTrack*, *calcOpticalFlowPyrLK* determines for each element the position where the observation can be found in current image. Thus there are two ordered sets of points, which are returned by the function *findCorrespondingPointsWithKLT* and represent at the same time the input argument for the function *findHomographyWithRANSAC*. This function uses *cv::findHomography*, which estimates a homography that transforms the previous point set onto the current point set with a minimal error over all points. Among the parameters of *cv::findHomography*, RANSAC can be choosen as the method to use. The obtained homogeneous matrix is believed to represent the inverse camera motion and is therefore returned as the result of the *calcTrans* function.

4.2.3 Camera Motion Compensator

The *CameraMotionCompensator* class is responsible for generating a model valid for the current grid containing the information of the previous grid. To this end, the previous grid is warped with the transformation determined in section 4.2.2. Warped cells having an overlap with a cell in the current grid influence the respective cell proportional to the overlap ratio. If one looks at the *CameraMotionCompensator* module from the exterior, data flows in and out as follows:

Inputs Previous background model, Homography

Output Information from the previous background model placed in the current grid

CameraMotionCompensator
- toCompensate: Image2D<Cell> - warpResiduals: cv::Mat
+ compensate(homography: cv::Mat): void - determineWeightsOfPrevCells (prevCells: Image2D<Cell>, transformedCell: Cell, startSearchHere: cv::Point2i): map<Cell, float> - transformPoint2WithMat3(toTransform: cv::Point2i, homography: cv::Mat): cv::Point2i - recenterAll(cellImage: Image2D<Cell>, homography: cv::Mat): void - normalizeCellWeightsSoTheirSumIsOne (weightSumSoFar: float, weights: map<Cell, float): void

Figure 9: Fields and methods of the *CameraMotionCompensator* class. From outside, the *compensate* function is accessible, which modifies the grid *toCompensate* based on the passed transformation matrix.

Figure 9 shows that the function *compensate* acts as the module's only interface to the outside. When *compensate* is called, first of all *recenterAll* transforms every cell in the grid according to the passed matrix. The function *transformPoint2WithMat3* applies the homogeneous transformation matrix to the cell's center point to obtain the new center point's coordinates. The transformed cell is constructed around the determined center point. Since the point coordinates are integers, rounding is necessary. To avoid the accumulation of manipulations through rounding over several frames, the residuals of the rounding operation are accumulated in *warpResiduals*. For future rounding operations, the warp residuals are considered, i.e. added before rounding.

Next, a local object of type *Deblurrer* is created whose life time ends with exiting the function. It is used in the following cell mixture process. A loop iterates over all cells in the grid to be motion compensated. Therein, the transformed cells from the previous frame are determined that overlap with the current cell. When using the term transformed cells it is referred to cells whose centers were transformed as explained above. Checking each transformed cell for an overlap for each cell in the current grid would be too costly since it entailed a complexity of n_c^2, where n_c is the number of cells in the grid, typically > 2000. Instead, a pre-calculated point is additionally passed to the function *determineWeightsOfPrevCells*. This point has

been calculated during cell recentering and indicates where to start the search. The cell is determined where this start point is located. A 3×3 block with this cell in the center is selected for the overlap check. The complexity can thus be reduced to $9 \cdot n_c$. Since all overlapping cells can be found inside the 3×3 block as long as there is no extreme zoom, this technique does not affect the results. Suchlike zooms are not possible with the speed of *Rollin' Justin*.

Iterating over the cells in the 3×3 block, the overlapping ratio is obtained by calling the *intersectionRatioWith* function on the transformed cell, passing the cell from the current grid. The function *intersectionRatioWith* is based on the geometric properties of the cell only and is therefore defined in the inherited *Square* class. It is based on the x- and y-coordinate displacements between the centers of the cell and can be understood best from the illustration in Figure 3. The cell's weights are equated to their overlap ratio. Following equation 18 (page 23), the weights in the 3×3 block should add up to one. With the strategy of equating weights to the overlap ratio, however, this is not always the case. Due to the homography transformation, it is not guaranteed that the transformed grid fully covers the current grid. The sum of the weights over all influencing cells can thereby be less than one. Weight sums greater than one are also possible when the camera moved closer to objects, thus causing a zoom effect. To fulfill equation 18, the weights must add up to one while the proportionality among them is preserved. Changing the weights accordingly is the responsibility of function *normalizeCellWeights*.

Having determined the weights for the cell in the current grid, its Gaussian model can be assembled by averaging the influencing cells proportional to their weights according to equations 15 - 17 (page 15) and equation 54 (page 54). The locally created *Deblurrer* is notified to process the influencing cells. Influencing cells with a zero age value are skipped as a zero age indicates no measurement is incorporated in this cell. This can happen in newly covered regions where only depth error measures have been captured so far.

After all Gaussian models in the current grid have been obtained, the *Deblurrer* is called to adapt the mean values of the apparent Gaussian models.

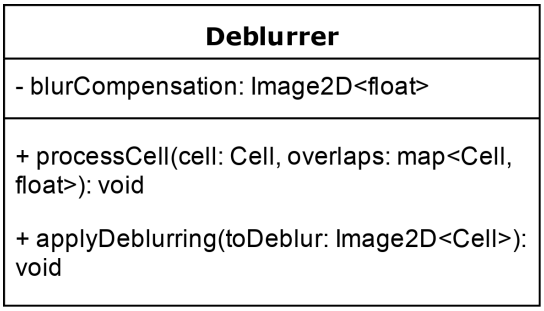

Figure 10: The method *processCell* is to be called for each cell during the cell mixture processes, passing the weights of the influencing cells. The *Deblurrer* then checks if a blur compensation is necessary and stores the compensation value in its member *blurCompensation*, which is an image with grid resolution. After all cells have been processed, *applyDeblurring* can be called to add each value in the compensation image to the respective apparent model's mean.

4.2.4 Deblurrer

The *Deblurrer* class undertakes the tasks necessary to realize the concept elaborated in section 3.3. Its inputs and outputs are therefore:

Inputs Influencing cells from the previous grid for each cell in the current grid,
Camera motion compensated background model

Output Camera motion and blur compensated background model

Following the concept from subsection 3.3.3, the function *processCell* analyzes the influencing cells of each cell and determines a blur compensation value. First, the influencing cells are bisected in two halves. From these halves, the blur difference and all other values required for equation 47 and 50 are calculated so that the blur compensation value $\Delta\mu_{t,}(c)$ can be determined as per equation 52. The resulting $\Delta\mu_{t,}(c)$ is written in the image *blurCompensation* at the coordinates of the center of cell c. Thereby for each cell in the current grid there is an entry in the image *blurCompensation* that states what must be added to the apparent's mean value in order to compensate the blur. This blur compensation is performed on the grid passed on the invocation of *applyDeblurring*.

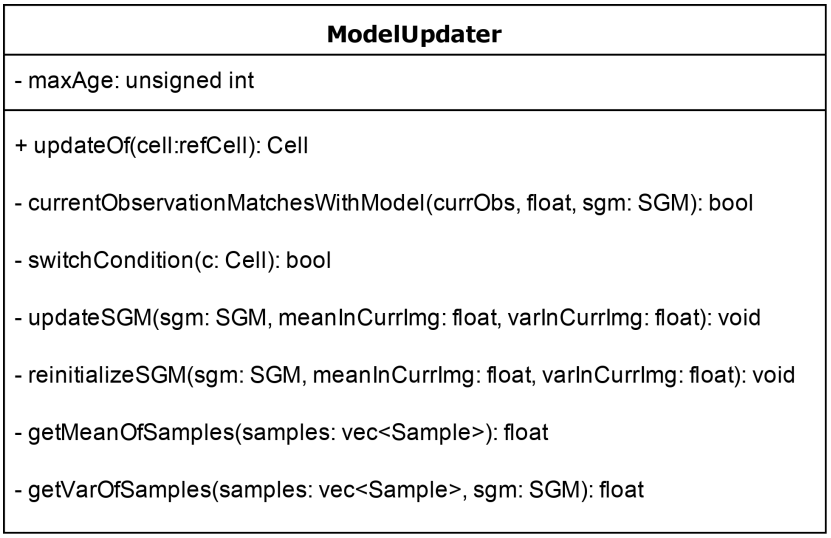

Figure 11: Member overview of the *ModelUpdater* class. The method *updateOf* updates the mean, variance and age of the passed cell with samples of the current depth image and uses thereto the displayed functions.

4.2.5 Model Updater

When the background model from the previous frame has been transformed onto the current grid, it can incorporate the current observations. Incorporating the current depth image is the responsibility of the model updater. Since the update logic of one cell does not depend on any other cell, the functions of the *ModelUpdater* class process only one cell at the same time. The incoming and outgoing data can hence be presented as:

Input A cell with the Gaussian models containing information from the previous frame

Output The input cell updated with the current depth image

The *ModelMaintainer* holds the instance of the *ModelUpdater* and calls the *updateOf* function after the background model from the previous frame has been motion compensated. When *updateOf* is called, it is first checked if the sample set of the passed cell is empty. If so, the passed cell is returned unmodified. The sample set for the current frame has been determined by the sampling module in the previous frame (see subsection 4.2.7). Next, *getMeanOfSamples* and *getVarOfSamples* calculate the mean depth value of the sample set and the variance following equations 11 and 12. Pixels with an invalid depth measurement, here indicated by a zero value, are not considered. If it turns out the sample

set consists of only invalid measurements, the cell is also returned unmodified. In the next step, *currentObservationMatchesWithModel* checks whether the samples' mean fits into the apparent model (equation 13). If so, the apparent model is updated with the sample's mean and average value. Otherwise it is checked whether the samples support the candidate model. If it does, the update is performed on the candidate model. The model that is selected for the update increments its age value by one as one more frame supports the state of the model. The age is capped at *maxAge*. When *currentObservationMatchesWithModel* is true for neither the apparent nor the candidate model, *reinitializeSGM* is called passing the candidate model. This reinitializes the candidate model with the samples' mean and variance and sets the age to one. The update process finishes with the call of *switchCondition*, which returns true if the age value of the candidate model is greater than the age of the apparent model. In this case the candidate model becomes the new apparent and the new candidate is reinitialized with the current samples' observation. Finally, *updateOf* returns the passed cell with updated apparent and candidate models.

4.2.6 Background Subtracter

BackgroundSubtracter
+ bgSubtraction(samples: vec<Sample>, apparentBgModel: Image2D<Cell>): cv::Mat
+ calcFGProbability(samples: vec<Sample>, apparentBGModel: Image2D<Cell>): cv::Mat
+ makeBinary(diffImg: cv::Mat, threshold: unsigned int): cv::Mat
+ applyBoxFilter(foregroundMask: cv::Mat, Ksize_x: unsigned int): cv::Mat

Figure 12: Methods of the BackgroundSubtracter namespace. Besides of generating the differencing image and binary mask based on the background model and the current depth image, the BackgroundSubtracter is also responsible for applying the box filter that smooths the output mask.

The background subtracter module consists of a name space containing three functions covering the functionality explained in section 3.4. Background subtraction is applied only on the samples selected by the sampling module (see subsection 4.2.7).

The first function, *bgSubtraction*, takes the samples and the background model as its input. From each sample the mean of the apparent background model of the subjacent cell is subtracted. The subjacent cell can be found by considering the coordinates of the sample. Having executed this operation for every sample, the differencing image is returned. For all other pixels that are no samples, the value in the differencing image is zero. From the differencing image, the foreground probability of a sample is determined in *calcFGProbability* by considering also the variance of the underlying apparent model (see equation 60). The foreground probability is zero when the pixel in the current depth image is farther away than in the background model, which is the case when the aforementioned difference is positive. The function *makeBinary* thresholds this differencing image, so that the returned mask has value one if the pixel in the differencing image is larger than a passed threshold and zero otherwise. One values indicate a dynamic object and the desired foreground mask is thus obtained. Lastly, the function *applyBoxFilter* realizes the concept from section 3.5. It takes the raw foreground mask as its input, applies box filtering with a kernel size of $Ksize_x \times Ksize_x$, and returns the smoothened final binary detection mask.

4.2.7 Sampling

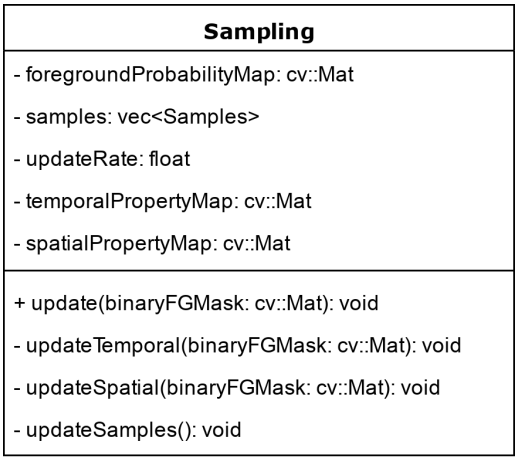

Figure 13: The sampling class being responsible for extracting pixel positions that are used as samples in the next frame. Regions are densely selected as samples if dynamic objects have covered the region in the previous frames.

After the binary foreground mask has been determined for the current frame, pixel positions

are selected that serve as samples for the upcoming frame. In- and outputs of the *Sampling* module are

Input Binary foreground mask from time t

Output Set of pixels which serve as samples for time $t + 1$

The pixels selected for sampling are stored in the *samples* field. Besides of the pixel position, a sample consists also of the observation at that pixel. However, as it is not yet known for the upcoming frame, only the position of the sample is set. The value is assigned to the samples when the new frame arrives.

Samples are selected in the *Sampling* class by following the method from section 3.5. To this end, an alternative foreground probability is calculated for each pixel based on whether dynamic objects have been detected in this region in the previous frames. The pixel-wise defined images *temporalPropertyMap* and *spatialPropertyMap* correspond to $L_t^{tmp}(p)$ and $L_t^{spat}(p)$ from equations 64 and 65. When the foreground mask is calculated for the current frame, the *Coordinator* passes is to the *update* function, which triggers the update of the temporal and spatial property images. In *updateSpatial*, a box filter is used to average the neighboring pixels. After updating the temporal and spatial property images, the *foregroundProbabilityMap* is obtained by element-wise multiplication of these two images. Then, the *samples* field is updated with the strategy discussed in section 3.5.

4.2.8 Coordinator

Finally, the coordinator concatenates the afore presented modules. In a loop processing every incoming frame, the coordinator initiates the processing stages:

```
1 // called for every frame
2 void loop() {
3   // estimate the camera motion
4   cv::Mat homography = cameraMotionEstimator.calcTrans(prevImg.depth,
        currImg.depth);
5   // update the model with the current observation
6   modelMaintainer.update(homography, currImg.depth, samplingMap.
        m_samples);
7   // calculate foreground probability through background subtraction
8   cv::Mat fgProb = BGSubtraction::calcFGProbability(samplingMap.
        m_samples, modelMaintainer.getApparent());
```

```
9   // threshold foreground probability image with threshold 4
10  cv::Mat fgMask = BGSubtraction::makeBinary(fgProb, 4);
11  // generate the samples for the next interation
12  sampling.update(fgMask);
13 }
```

The calculated foreground mask *fgMask* is then ouput in a stream readable for other systems. The current and previous depth and intensity images are captured beforehand from the sensor. As the logic of the system does not require the time intervals between frames to be equal, the most recent available frame is processed. If the loop has finished and the next frame is not available yet, the coordinator fills the gap with an active wait until a new image can be tapped from the stream. Processing of a frame ends by setting up the previous frame for the next iteration, i.e. overwriting *prevImg* with *currImg*.

5 Experiments and Results

5.1 Experimental Setup

Experiments are conducted on a workstation computer with an Intel(R) Xeon(R) W-2133 CPU at 3.60GHz with six cores using the openSUSE Leap 42.3 operating system. As in most of related work [8][7][9][18], an image resolution of 320×240 is chosen for both depth and RGB images. Four image sequences from the *LIRIS* human activity dataset [34] are selected to evaluate the proposed system. The *LIRIS* dataset consists of training data and test data. The training data is used to configure the parameters of the system. Quantitative measurements are performed on two other sequences from the test data. Both camera motion and dynamic object movements are included in the selected sequences *vid0121*, *vid0172* (training data) and *vid0066*, *vid0124* (test data). The dynamic objects are humans walking in the field of view in order to execute a certain activity. This scenario matches with the motivation of this work as Rollin' Justin should be able to segment persons operating in the surrounding area. The videos are recorded from a Kinect mounted on a mobile robot [34]. At a fame rate of 25fps and a resolution of 640×480 pixels, the dataset provides depth and gray images. Depth images are converted to metric units before being used in the proposed system.

Ground truth information is provided in the form of bounding boxes of the dynamic object. The bounding box coordinates are valid for the gray image. Since the primary goal of the *LIRIS* dataset is to train systems classifying human activities, the annotation file containing the bounding boxes only covers certain time intervals in the video. Also, the bounding box is only provided for one dynamic object at the same time. Evaluation metrics can therefore not be determined for sequences containing multiple dynamic objects. As the output of the proposed system is a pixel-wise mask, its quantitative performance cannot be measured using bounding boxes. Instead, in order to obtain a ground truth mask that segments every pixel belonging to the dynamic object, *k-means* clustering is performed with *sklearn* [35] in Python on the depth values inside the bounding box. Setting $k = 2$, the cluster whose center is closer to the camera is believed to contain the pixel values of the dynamic object. Hence, all pixels in the bounding box are extracted that belong to this foreground cluster. This technique works only for sequence intervals where there are no other objects in the bounding box in a similar or closer distance to the camera. Quantitative comparison of the proposed system is

therefore restricted to these sequences and intervals. Depth error measurements are filtered out before applying *k-means*. Furthermore, the bounding box is enlarged beforehand by ten percent of the image width and height because the bounding box coordinates are valid for the gray image, but the depth image is not registered [1] with the gray image in the dataset. Camera calibration parameters are provided, but for simplification of the evaluation process registration of the gray and depth image is not performed.

Omitting the registration stage also tests the robustness of the grid-wise background modeling technique with respect to inaccurate estimates of the camera motion since the gray images are used for the camera motion estimation with KLT and the depth images are used for background modeling.

5.2 Performance Evaluation

Quantitative evaluation is performed by pixel-wise comparison of the output mask and the ground truth. Using the default parameters from table 2, the system is evaluated with respect to precision, recall, F-Score and the Jaccard index, which are explained in the following. All metrics range from 0 to 1, where a score is better. The number of true positives, true negatives, false positives and false negatives is denoted as TP,TN,FP and FN. The precision $TP/(TP + FP)$ represents the ratio of correctly detected foreground pixels to all detected foreground pixels. The recall $TP/(TP + FN)$ describes the ratio of correctly detected foreground pixels to actual foreground pixels. Both metrics in combination are relevant for estimating a system's performance. Under-detection typically results in a good precision since the number of false positives is low. The recall, however, is high, since many actual foreground regions have not been detected. Conversely, this applies to over-detecting systems. Therefore, the F-Score is measured which combines precision and recall: $F = 2 \cdot (precision \cdot recall)/(precision + recall)$. Finally, the Jaccard index, also known as Intersection over Union (IoU), is measured. Expressed in terms of true and false positives and negatives, it is $J = TP/(TP + FP + FN)$. Hence, the Jaccard index is always less than or equal to the precision and recall individually.

[1] Two images are registered if their data is in the same coordinate system

Table 2: Standard parameters used when measuring performance metrics

Parameter	Value	Description	Reference
n_{rows}	320	input image height	sec. 3.1
n_{cols}	240	input image width	sec. 3.1
N	4	cell edge length	sec. 3.1
$maxPointsToFind$	100	for cv::goodFeaturesToTrack	subsec. 4.2.2
$minQualityForAGoodPoint$	0.01	for cv::goodFeaturesToTrack	subsec. 4.2.2
α_{max}	170	model age cap	sec. 3.1
λ	0.3	model update rate	sec. 3.1
σ_{min}	100	minimum variance to keep	sec. 3.1
k_{var}	2	variance fit factor	sec. 3.1
k_{switch}	2	switch models factor	sec. 3.1
λ_{smpl}	0.25	sampling map update rate	sec. 3.5
θ_{smpl}	0.02	sample probability threshold	sec. 3.5
θ_{fg}	2	difference image threshold	sec. 3.4
$Ksize$	7×7	kernel size box filter output mask	sec. 3.5

Table 3: Quantitative results on *test* sequences from the *LIRIS* dataset averaged over all frames where ground truth data is available. The sequence *vid0124** is generated by prepending 75 frames (3 seconds) to the original sequence *vid0124*. The prepended frames are all equivalent to the the first frame of *vid0124*.

Sequence	Frames	Precision	Recall	F-Score	Jaccard index
vid0066	140-222	0.979	0.983	0.978	0.957
vid0124	93-214	0.774	0.444	0.564	0.441
vid0124*	168-289	0.868	0.928	0.900	0.853

Table 3 shows the performance of the system elaborated in this work. The program was run with the parameter setup listed in table 2. They were manually chosen based on the sequences *vid0121* and *0172* in the training data.

Not all frames of the sequence are considered for determining the metrics as ground truth annotations are only partially available. Ground truth data were to be extracted first based on bounding boxes. This was not possible for all frames (see sec. 5.1). However, the time interval for which the shape of the dynamic object could be extracted is fully included in the measurements.

Sequence *vid0066* shows a person walking through an open door in order to put some documents in a shelve inside the room. An early frame is depicted in the left images of

figure 14. Shortly after the person passed the door and still while they are walking, the camera is panned. Scores of all above 0.95 in table 3 show that the system is able to handle simultaneous camera and object movements.

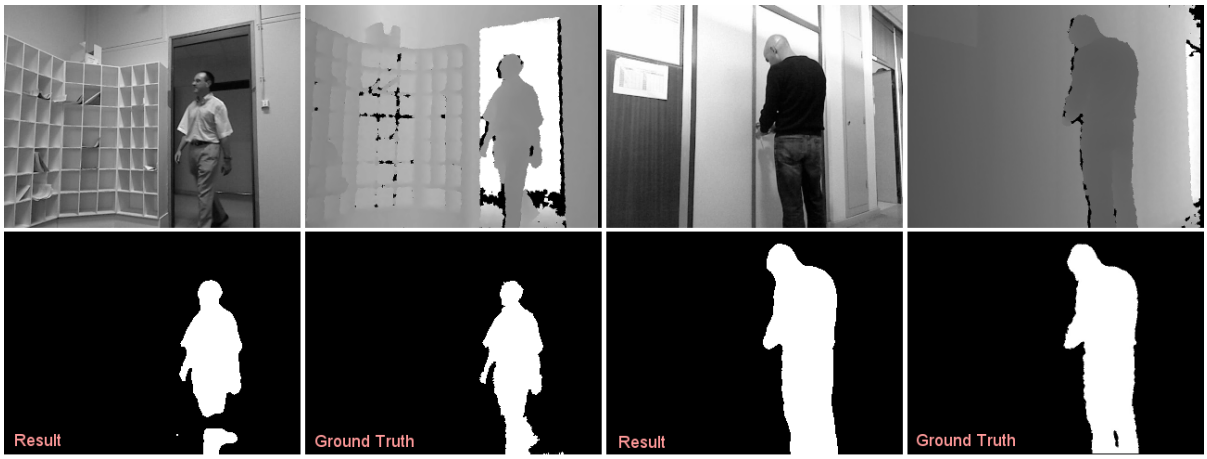

Figure 14: Gray image, depth image (top row), segmentation results and ground truth (bottom row). The four images on the left show frame 93 of sequence *vid0066* and the images on the right represent frame 93 of sequence *vid0124*.

Figure 15: Dynamic object detection results on the 160th frame of *vid0124* (left) and the 235th frame of *vid0124** (center). The input depth image is identical for both frames since *vid0124** was generated by prepending 75 frames to vid0124 $(160 + 75 = 235)$. Ground truth data valid for both is depicted in the right image. The figure shows that the person is detected significantly better on the modified sequence *vid0124**.

Testing with the sequence *vid0124* yielded poor results in particular for the recall. A person entered the scene in the 51st frame and stopped at a door trying to open it with their keys. As it is depicted in the images on the right in figure 14, foreground segmentation works properly at frame 93. At about frame 150, the switch condition from section 3.1 takes effect in some

regions of the image. This is because the number of frames where the person was observed is more than twice ($\theta_{switch} = 2$) as large as the number of frames where the background was recorded. Hence, the *apparent* and *candidate* models are switched and the person is incorporated in the background model. In the following frames the person is consequently no longer detected as a dynamic object. This effect, visualized in figure 15, explains the large number of false negatives leading to the low recall score from table 3. Increasing the threshold θ_{switch} could be a countermeasure, however, initial wrong classifications would then remain longer in the image. The primary issue is that the age of the background model is not large enough before the person occludes the static background. When increasing the time before the person enters the scene, the problem can be eliminated. Therefore, in the sequence *vid0124**, the first frame of *vid0124* is duplicated 75 times. Before the real sequence starts, the system is thus supplied 75 times with the initial image. Figure 15 shows that the defect of sequence *vid0124* does not eventuate in sequence *vid0124**. Evaluating the same images as in *vid0124*, table 3 shows that significantly higher scores are achieved with *vid0124**.

Runtime

Table 4: Computation time per frame per functional module. The CameraMotionEstimator (CME) consumes the least time, whereas the *CameraMotionCompensator* (CMC) is responsible for more than 50% of the time spent on both maximum and averaged frames.

	CME	CMC	BGSubtractor	Deblurrer	ModelUpdater	Sampling	Total
avg	0.2ms	22.5ms	5.1ms	2.5ms	3.4ms	5.0ms	38.7ms
max	0.2ms	32.9ms	5.1ms	3.9ms	4.6ms	7.6ms	54.0ms

Running on the sequences of the *LIRIS* dataset, an average computation time of 39ms per frame is achieved with a variance of 1.3ms. This performance allows the system to be run with a live camera input stream at 25fps. Table 4 shows the composition of the total computational cost based on the time spent in each functional module. The largest observed time spent on a single frame was 54ms. Computational cost increases mainly through two factors, which are a large occlusion from dynamic objects and camera motion. Large rates of the image belonging to a dynamic object induce the *Sampling* module to extract more pixels which are to be processed in the next frame. When the camera moves, the transformation of the background model becomes necessary. Transforming the background model is achieved through the cell

mixture process, which is the most time consuming part of the proposed system. On average, 58% of the computation time is spent within methods of the *CameraMotionCompensator* module.

5.3 Effects of Parameters

The detection results are examined under various parameter setups. Depending on the environments in specific use case scenarios, an adaption of the default parameters from table 2 may be sensible. Testing is performed on the training subset of the *LIRIS* dataset.

Figure 16: Detection results on frame 146 of *vid0059* while the camera is shifting to the right. Gray and depth images which are the input for the system are shown in the top row. The foreground mask output with the default parameter setup from table 2 is shown at the bottom left. At the bottom right the ground truth mask is displayed.

A frame of sequence *vid0059* is selected to examine how the results change under the modification of parameters. Running the system with the default parameter setup, the mask displayed at the bottom left of figure 16 is output. Figure 17 shows how the foreground mask from figure 16 changes when exactly one of the following parameters is modified.

For the record of the top left image, the **model update rate** λ is decreased from 0.25

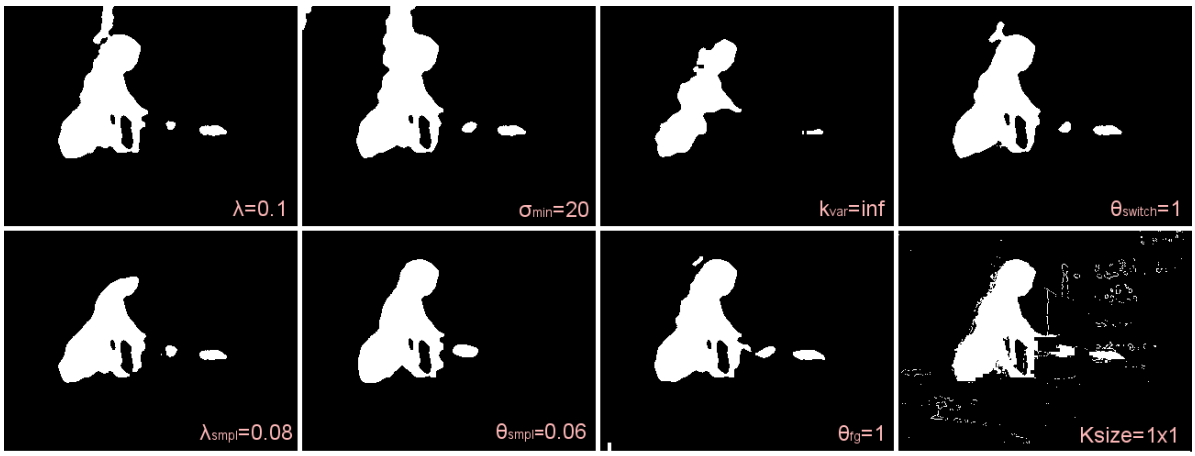

Figure 17: Results on frame 146 of *vid0059* when changing exactly one parameter each.

to 0.1. This leads to false foreground positives above the head of the women since the background model was unable to adapt to the changing depth while the camera moved. Hence, parts of the wall are closer in the current depth image than in the background model. After applying background subtraction, these parts of the wall are incorrectly extracted as foreground regions. A too low update rate λ can thus cause wrong foreground classifications when the camera moves. Fast camera motions require a larger update rate λ. If a high λ is chosen, the observations of new frames replace the existing model more quickly. Single erroneous measurements are thus more harmful. Furthermore, when dynamic objects are initially in a similar distance to the camera as the background but then move towards the camera, the model adapts its mean value to the depth of the dynamic object in each frame. Hence, the dynamic object remains undetected.

The second image in the top row shows the effects when the **minimum variance to keep $\boldsymbol{\sigma_{min}}$** is decreased. Similar to the first image in the top row, but to a greater extent, areas above the women's head are incorrectly classified as foreground. The minimum variance to keep σ_{min} describes the value which is assigned to the model's variance $\sigma_{t,c}$ whenever it would fall below σ_{min}. It prevents the model from becoming unreceptive for small changes in cells where all samples are very similar. Samples with a very similar depth value occur frequently when observing planes frontally. In the sequence of figure 17, the camera pointed directly to the wall for some frames and then started to move. Since the variance became very low while observing the wall, some cells did not adapt to the slightly changed depth value after the camera has moved. The resulting mask in the image exhibits false foreground positives

arising from this. On the other hand, if σ_{min} is chosen too large, foreground objects cannot be distinguished from the background if they are close to the background. The reason for that is that background subtraction is performed considering also the variance (see equation 60).

In the third image of the top row it is depicted what happens if the **variance fit factor $\mathbf{k_{var}}$** is assigned an infinite value. With $k_{var} = inf$, the condition from equation 14 is always false. This means the variance of the current observation is not required to be similar to the variance of the model when selecting the model for the update process. This condition, complementing equation 13, was introduced in this work to prevent dynamic object boundaries from being incorporated in the background model. Section 3.1 explained how wrong incorporations can be caused by a gradual increase of the variance at boundaries of dynamic objects. Consequently, the image of the resulting mask without this condition shows that the boundaries of the person are incorrectly classified as background. On the contrary, when activating the condition from equation 14 with a $k_{var} < 1$, the apparent model would reject a sample set if the variance among the samples is higher than the variance of the model. This is not sensible because the model's variance could never increase and is therefore limited to the initial value.

The **model switch factor $\mathbf{k_{switch}}$** formulates by which factor the age of the candidate model must exceed the age of the apparent model so that they are switched. A low k_{switch} facilitates the switch, meaning the candidate model must be supported by a lower number of frames in order to become the apparent model. The age of the candidate model becomes large in particular when dynamic objects stand still for a longer time. Hence, it is useful to select a higher k_{switch} if dynamic objects are expected to stop after moving in the field of view. On the other hand, a higher k_{switch} also increases the time for which initial false classifications remain in the output detection mask.

The **sampling map update rate $\boldsymbol{\lambda_{smpl}}$** determines how fast the sampling image L_t^{fg} adopts to changing positions of dynamic objects (see section 3.5). Faster object movements require a higher λ_{smpl}. If λ_{smpl} is chosen too low, the resulting mask lacks of foreground detections in the moving direction of dynamic objects. This is shown by the image on the bottom left of figure 17. The under-detection is due to the fact that samples are not selected densely in these regions and therefore the sparse detection points are eliminated when smoothing the mask later on. If, on the other hand, a too large λ_{smpl} is chosen, single wrong detections are more likely to persist in the output mask. Such single wrong detection occur in particular on

depth discontinuities of static objects.

Increasing the **sample threshold $\boldsymbol{\theta_{smpl}}$** primarily affects regions in which the dynamic object covers only a few pixels. In the second image in the bottom row it can bee seen that the book in the hand of the person is not detected. This is because for the calculation of L_t^{fg} the spatial neighborhood of the detection mask from the previous frame has been considered. For thin objects like the hand on the book, L_t^{fg} is consequently smaller and samples are no longer selected if $L_t^{fg}(p)$ is smaller than θ_{smpl}.

The **detection threshold $\boldsymbol{\theta_{fg}}$** directly regulates over- or under-detection. If the difference between the input depth image and the background model divided by the variance of the background model is greater than θ_{fg}, the respective pixel is classified as foreground. The third image in the bottom row of figure 17 shows the output mask for $\theta_{fg} = 1$. In Comparison to the results from figure 16, where $\theta_{fg} = 2$, a small region above the women's head is falsely detected. In this region, the difference between input depth image and model was therefore between one and two times the variance. Because the variance describes the maximum deviation of the background in the respective cell, a θ_{fg} smaller than one is not sensible as pixels showing the background would be additionally assigned the foreground label. When θ_{fg} is chosen too large, the system will be unable to detect dynamic objects in front of depth discontinuities in the background since the variance of the background model is high for these cells.

Ksize is the size of the kernel for box filtering the output mask which is executed as the final step in each frame. The box filter smooths the binary output mask, removes single foreground classifications and fills holes. Without this step, the output mask would exhibit multiple tiny false classifications as shown by figure 17 in the bottom right. However, a large $Ksize$ can also reduce the accuracy as fine structures are removed from the output mask. Hence, when the dynamic objects are expected to cover only a small region in the image, a smaller $Ksize$ is advisable.

5.4 Effects of Deblurring Technique

Evaluation of the anti-blurring technique introduced in this work is performed on the TUM RGB-D dataset [36]. In contrast to the relatively short sequences in the *LIRIS* dataset used above, sequence *walking_static* in the dynamic object category of the TUM dataset

contains continuous camera motion over 25 seconds (723 frames). Blurring of the background model happens gradually, therefore its consequences can be seen best using longer sequences. Analyzing the behavior for longer sequences is also reasonable as the system developed in this work should be able to serve as a filter in long-term applications.

Figure 18 shows how the background model becomes blurry when the blur neutralization mechanism is skipped. A blurry background model can cause both false positives and negatives in the foreground detection mask. Although no dynamic objects are visible in frame 350, figure 18 shows that parts of the table are classified as foreground without deblurring. These false positives originate from the fuzzy edge in the background model. The input depth images clearly shows the edge of the table, whereas in the background model it has almost disappeared. Background subtraction yields thus a large difference for the affected cells. Consequently, the table edge is assumed to be a foreground object.

This can be prevented by the proposed blur neutralization scheme. In the image showing the background model with deblurring for frame 350, the table edge is still visible in a darker gray than the wall above and the floor below. The resulting foreground mask shows that the number of false positives is significantly lower also for frame 190 and 650.

In the last column of figure 18, the two persons returned to their chairs. The left chair has been moved and it influenced the background model in its different positions. Therefore, the region around the left chair can be perceived in darker gray in both the blurred and deblurred background model. As the depth of the person sitting on it does not significantly differ from the chair in the background model, only the head which is above the chair is segmented in the foreground mask. The legs from the person on the left are detected only when using the blurred background model because the right outer edge of the chair is not clearly visible in the blurred background model. On the other hand, the legs of the person on the right are only detected with activated deblurring since the blurred background model shows the entire region in darker gray.

Figure 19 focuses on the realization of the blur compensation method focusing a single frame. The image in the center shows the absolute values of the blur compensation image $L_t^{\Delta\mu}(c)$ from subsection 3.3.3. This is the image which is calculated while mixing cells from the previous frame and added afterwards to the mean of the apparent background model. Adding $L_t^{\Delta\mu}(c)$ to the left image in figure 19 (means of apparent background model) results

in the right image (blur compensated apparent background model). Not only the edges have been sharpened, but also white regions have been preserved from turning gray.

From a quantitative perspective, the detection accuracy can be significantly enhanced by the proposed anti-blur technique. Deactivating this mechanism, the Jaccard index for *vid0066* of the *LIRIS* dataset drops from 0.957 to 0.686. Also for the second measured sequence *vid0124*, the achieved Jaccard index decreases without deblurring (0.394 instead of 0.441).

5.5 Comparison to Results Achieved in Related Work

This work seizes the idea of a dual-mode background model from MCD5.8 [8]. Extensions from Yun *et al.* [7] are adopted. However, a performance comparison to their work is not possible for two reasons. First, neither quantitative results nor any images are presented alongside the original method MCD5.8. Second, this work introduced depth background modeling, but both MCD5.8 and the system from Yun *et al.* use RGB images only. The dataset used by Yun *et al.* contains therefore no depth data. Xu *et al.* [21], Kim *et al.* [22] and Wu *et al.* [23] used also only intensity images. Other work, including StaticFusion [18] and the method of Sun *et al.* [31] provide only measures for their improved visual odometry. Xu *et al.* [27] evaluated their *IVibe* system on the Princeton dataset [37], where multiple objects move, but ground truth annotation is only available for one object. For the evaluation of *IVibe*, they added therefore ground truth manually to each frame, however, they did not publish these annotations.

Consequently, comparison of this work is limited. In order to provide a comparison nevertheless, the DEVB [38] method and an own implementation of the method from Yun *et al.* is selected.

The DEVB was not considered up to here as their system is only operable with a static camera. However, as the system proposed in this work should also work when the camera is static, this constraint does not forbid a comparison. Figure 20 compares the qualitative results of DEVB and this work. It shows that the result mask of this work is smoother and without holes. Dynamic objects near the right margin of the image remain undetected in this work as they move in front of regions where no depth could be measured. With no information in the background model, the dynamic object cannot be distinguished from the

background. Therefore, the proposed system works best in indoor environments where the background is in a measurable distance.

Additionally, the results of this work are compared to the results achieved when using intensity images only. Background modeling with intensity images is performed in the method from Yun *et al.* , which also uses MCD5.8 as its baseline. The system developed in *this* work can be viewed as the system from Yun *et al.* extended by additional rules, parameters, smoothing and anti-blurring and modified by using depth images for the background model. Therefore, to compare this work to the work from Yun *et al.* , evaluation is performed without all aforementioned contributions sticking to the system as it is proposed in their paper [7]. Figure 21 shows the results from that evaluation compared with the results achieved in this work.

Improvements by using depth images for the background model

When modeling then background using the intensity image, false detections occur at the edges of the shelve, where the intensity value differs strongly. The variance is therefore high in cells covering these edges. As a moving object can only be detected in front of a cell if the intensity of the moving object is outside the interval spanned by $mean \pm sqrt(variance)$, false negatives arise at the edges of the shelve. Their emergence can be seen in figure 21 as the thin black lines protruding in the person's body in the mask at frame 150. The long term effect of these false positives becomes visible at frame 200, where large parts of the person are incorporated in the background model (third row). Consequently, their detection is no longer possible.

Figure 21 shows also false positives at the edges of the shelve and the door, which can be seen best in frame 150. One factor for them is the blurring of edges, which makes them disappear in the background model but not in the current intensity image. A second factor is that the intensity is assumed to remain constant over time, however, it can change when the camera moves. Although the shelve is actually plain white, the observed intensity value is lower in the inner panels. This is because their reflectance is lower than that of frontally observed edges. Additionally, the shelve shadows itself partially. With the camera starting to move, the angle of incidence changes and therefore also the observed intensity value. The mean and variance of the background model does thus no longer accurately describe the

intensity in the current input image, resulting in false foreground detections.

Furthermore, illumination changes are critical for systems maintaining an intensity background model. In the foreground mask based on intensity only, a region with false positives was extracted in the top left of the image of figure 21 at frame 150. Shortly before, the camera adjusted its exposure settings. Hence, the new incoming images are brighter than the background model. In regions with a low variance like the wall in the top left of the image, this offset leads to false detections firstly.

Lastly, when the intensity value of the background is equal to the intensity of the dynamic object, it is impossible to detect the dynamic object.

Figure 21 shows that most of the aforementioned issues in related work do not occur in this work since depth images are used for modeling the background. Firstly, visible dynamic objects are always located in a smaller distance to the camera than the background behind them. Hence, the problem of very similar values for the background and the dynamic object does not only occur when the dynamic object is immediately in front of the background object. Secondly, depth modeling can exploit the fact that coherent objects have a similar depth value, reducing the amount and magnitude of edges in the background. Problems with large variances of cells occur thus far less frequently when maintaining a depth background model. The depth image is also resistant to illumination and reflectance changes.

Modeling the depth of the environment could thus reduce detection holes inside dynamic objects as well as false detections at edges while keeping the computational cost equal to intensity modeling.

5.6 Summary and Discussion of the Results

The basic concept of modeling the background with two Gaussian models extended by an age value [8] proved to be efficient and adaptive. Pooling pixels in cells allows the method to run in real-time. Maintaining two models keeps the model adaptive as initial false classifications can be rectified. By storing a mean and a variance in each model, it is possible to distinguish foreground and background if the foreground object is not similar to any background object in this cell. If there is an edge running through a cell, meaning the pixel values in this cell differ strongly, the variance of the model is high and ambiguities cannot be solved if the pixels on

the dynamic objects have a value which is not beyond the margin spanned by the variance. In comparison to intensity-based methods as *MCD5.8*, the usage of depth images could lower the occurrence of such ambiguities since objects typically have a smooth depth gradient, but their colors and intensities may vary. Additionally, foreground objects are believed to be always in front of the background. All foreground detections are thus filtered where the depth in the current image is larger than in the background model, reducing the amount of false positives in this way. Tests on a scene of the LIRIS dataset showed that the segmentation results are clearly more complete when modeling the depth instead of the intensity. The depth of the moving person differs from the background at the entire body, whereas the intensity only differs strong enough in some regions.

Quantitative evaluation of the system was performed on the LIRIS dataset. On a sequence with both camera and object motion, the detection accuracy measured by the Jaccard index reached 0.957. For a sequence with an object entering the scene already in the beginning of the sequence, the detection accuracy was significantly lower. The number of frames showing the dynamic object was larger than the number of frames showing the background. Therefore the switch mechanism took effect and the *apparent* background model falsely contains the moving object. Adding frames at the beginning of the sequence could eliminate this adverse effect.

The developed system works thus best when the initial frames show the static scene without any dynamic objects. Wrong classifications caused by the switch mechanism may occur when the background model is recent and the object's motion ceases. However, dynamic objects can be detected as soon as they move even if they are already present in the first frame. With respect to the intended use on the mobile robot *Rollin' Justin*, this behavior is acceptable as it is a long-term use case, where the camera provides a continuous stream as soon as the robot's system is turned on.

Experiments furthermore showed that fast dynamic object movements relative to the camera can pose a difficulty for the segmentation. The concept of background subtraction is suitable for fast object movements in general, however, due to the sampling strategy background subtraction is only performed densely in regions with a high foreground probability. Therefore, if the dynamic object is recently located in an image region where the background has been observed in the previous frames, the foreground extraction is sparse and therefore incomplete. An increase of the foreground probability update rate is an adaption to scenes where either

the camera or objects move quickly.

Experiments to evaluate the proposed deblurring technique were conducted using the TUM dataset [36] as in short sequences as in the LIRIS dataset the blur is weak. When the camera makes small movements and meanwhile observes the same region for a longer time, the background model becomes blurry. Experiments showed that a blurry model inhibits the detection of dynamic objects as the variance also rises with the blur. Activating the deblurring mechanism, edges preserve their sharpness and planes retain their actual depth value. The background model does thus represent the scene more accurate and background subtraction yields thus in general results with a higher validity. False detections, in particular at edges which are due to the blur no longer visible in the background model, could be reduced applying the deblurring. However, in some cases a blurred background model proved to be useful as holes are filled through the interpolative characteristic of the blur.

The general performance of the proposed system is satisfactory in indoor environments with moderate camera motion and dynamic objects which cover more than a few pixels, but less than half of the image. While parameter adjustments can adapt the system to the roughly expected velocities of camera and objects, a too large dynamic object coverage leads to a wrong camera motion estimation. On the contrary, too small dynamic objects are liable to be eliminated in the smoothing post-process. Foreground segmentation is also not possible in regions where no valid background information is available. As the proposed system models the background, is not designed to work in outdoor environments or halls with large distances that cannot be measured by the camera.

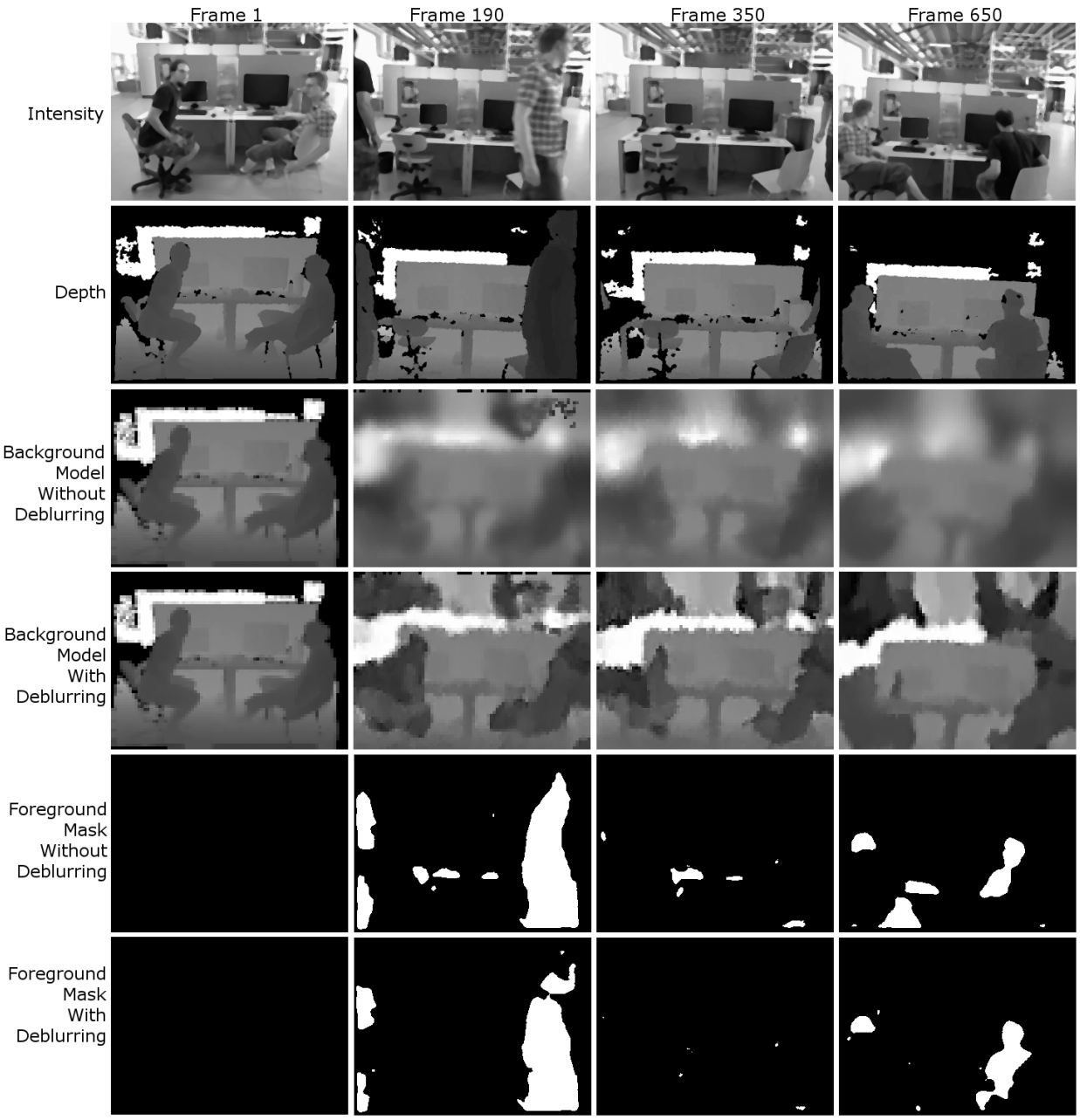

Figure 18: Effects of the deblurring technique using the sequence *walking_static*. The figure shows the *apparent* background model and the binary output mask with and without deblurring. In the first frame, the states are identical. The camera makes small movements over the entire sequence leading to a strongly blurred background model without the proposed deblurring technique (third row). The forth row shows that the sharpness can be preserved when applying deblurring. False detections on the table caused by a blurred background model can be eliminated with the proposed scheme (bottom rows). Also, the legs of the person on the right are only detected with the sharp background model.

Figure 19: Visualization of the blur compensation for a frame from the *walking_static* sequence. From the previous to the current frame, the camera has moved. Image a) is the background model obtained by transforming and mixing background model cells from the previous frame. Due to the interpolative characteristic of the mixture process, the background model blurs. Image b) shows the absolute blur compensation values calculated for every cell. White represents a large value. Image c) is obtained by adding a) and b). Edges in the resulting image c) are sharper than in the original from a).

Figure 20: Comparison to DEVB [38] using a sequence with a static camera. The shown color images are converted to gray scale and depth images to metric units before being processed.

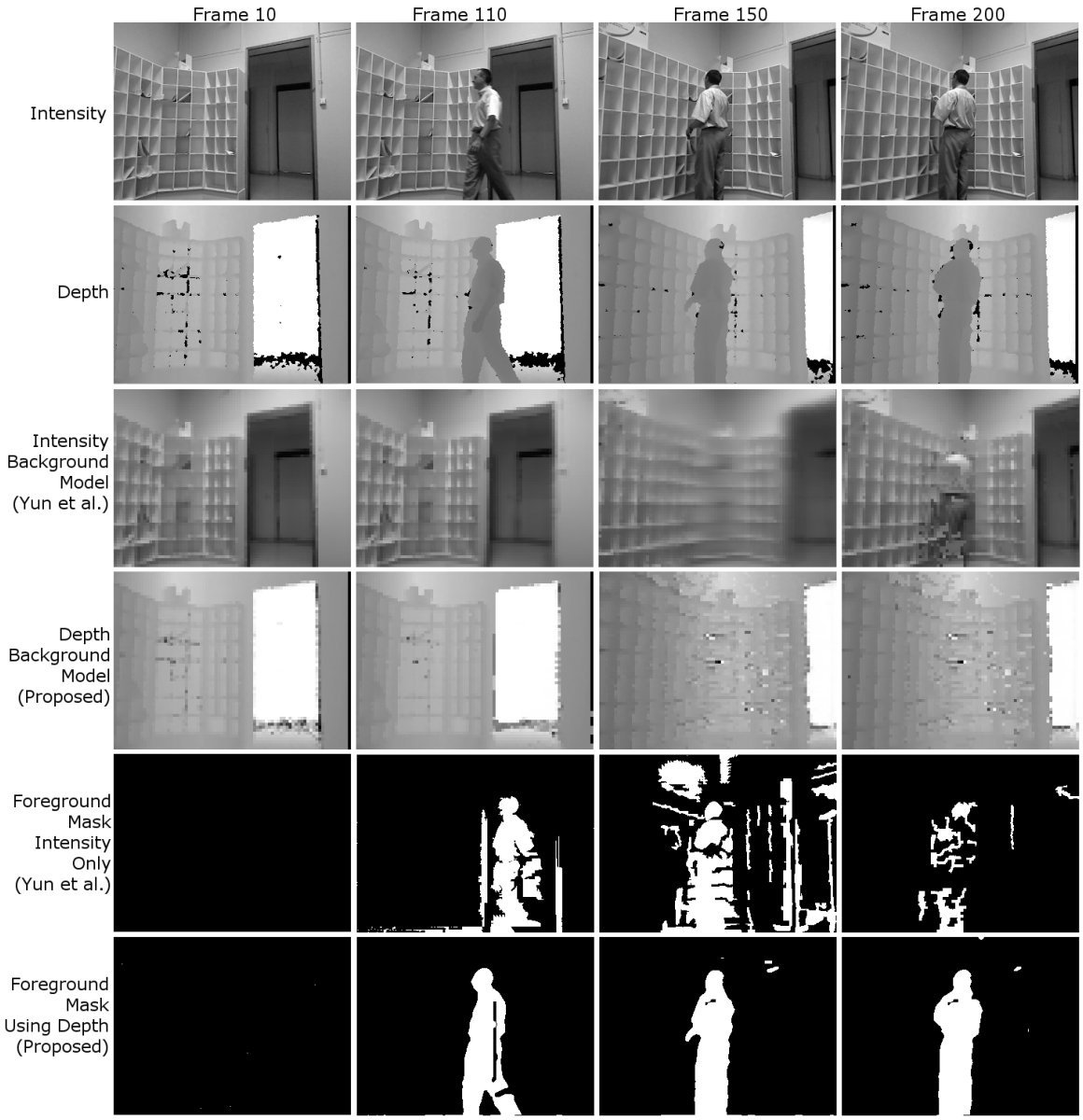

Figure 21: Comparison to intensity modeling as it is proposed by Yun *et al.* [7]. Sequence *vid0066* of the *LIRIS* dataset is used. The figure shows clearly that the detection accuracy could be improved by modeling the depth instead of the intensity.

6 Conclusion

In this thesis, an approach for foreground segmentation with a moving camera was presented. The following section summarizes the approach and the results of the proposed system. Eventually, section 6.2 contemplates future work enabled by the achievements in this work.

6.1 Summary

This work was motivated by the need to filter dynamic objects in the RGB-D camera stream of a mobile robot. Given the live color and depth streams, every pixel belonging to a dynamic object is to be extracted. Since objects that have moved once but then stop for a short time should still be detected as dynamic objects, frame to frame motion detection is not sufficient. Self-development became necessary since no suitable open source method could be found that is able to achieve this at a high frame rate without a GPU. Analyzing related work, background modeling proved to be the most suitable method for finding dynamic objects even if their motion ceases.

The proposed method is based on the background modeling strategy introduced in MCD5.8 [8]. Unlike MCD5.8, this work models the depth instead of the intensity. The input image is spatially partitioned and for each partition two Gaussian models are maintained. The Gaussian model's mean describes the depth value expected in this region. Modeling the same region twice enables storing two distinct depth values. The model that is supported by the most frames is assumed to model the depth of the static background. The desired foreground mask can be obtained by subtracting this background model from the input depth image. As the images are recorded from a moving camera, the background model has to be transformed from frame to frame. The background model was observed to blur gradually through consecutive transformations. This work improves the camera motion compensation technique by adding an anti-blur mechanism. The effects leading to this blur are analyzed, quantized and generalized to a formulation valid for all cells. Given a general formula which describes the emerging blur, the blur effects can be mostly neutralized.

The system is fully implemented in C++ using OpenCV libraries. Evaluation showed satisfying results for both detection accuracy and run time. With an average run time of

39ms (25fps) and a maximum observed run time of 54ms (18fps), the formulated real-time requirement is met. Impacts of adaptations show how the method can be adjusted for different use cases. With respect to the intended use on a mobile robot, the achieved detection accuracy and frame rate allows the usage of the proposed system as a pre-filter for incoming images from the camera.

6.2 Future Work

Evaluation of the method proposed in this work proved that dynamic objects can be successfully segmented from the static background given the images from a moving camera. The next step is the integration on DLR's mobile robot Rollin' Justin. The foreground detection mask is to be provided as an output stream which other systems can subscribe to. The system responsible for the localization process can then read the foreground mask and modify its logic so that only static objects are considered for the localization.

Regarding improvements of the system's performance, combining depth and intensity background modeling could improve the detection accuracy in many scenarios. Background subtraction on a depth model proved to be a reliable technique unless the dynamic object is very close to the background. When also considering the intensity value in such cases, the dynamic object could still be distinguished from the background.

The robustness of the system developed in this work could be further improved by considering the foreground detection mask from the previous frame for the camera motion estimation in the current frame. When the dynamic objects covers a large ratio of the image, the estimation of the camera motion could fail because the most common motion in the image is the motion of the dynamic object. This is, however, a realistic case as a person may walk directly in front of the robot where the camera is mounted on. Therefore, future work will strive to enhance the stability in such scenarios by executing the camera motion estimation only on points that have been classified as background in the previous frame.

Maintaining both a depth and an intensity background model as well as filtering points for the camera motion estimation would increase the computational cost of the system. As the most time was spent on calculating the overlap ratios for every cell after the camera has moved, future work will target to optimize this stage with respect to the runtime.

Bibliography

[1] P. Remagnino et al. "Novel concepts and challenges for the next generation of video surveillance systems." In: *Machine Vision Appl.* Vol. 18. 3-4. 2007, 135–137.

[2] C. Wren et al. "Pfinder: real-time tracking of the human body". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1997, 780–785.

[3] W. Hu et al. "A survey on visual surveillance of object motion and behaviors". In: *IEEE Trans. on Systems, Man, and Cybernetics*. Vol. 34. 3. 2004, 334–352.

[4] Shile Li and Dongheui Lee. "RGB-D SLAM in Dynamic Environements Using Static Point Weighting". In: *IEEE Robotics and Automation Letters*. Vol. 2. 4. 2017.

[5] Alexander Borst and Martin Engelhaaf. "Principles of visual motion detection". In: *Trends in Neuroscience*. 1989, pp. 297–306.

[6] Berthold K.P. Horn and Brian G. Schunck. *Determining Optical Flow*. Tech. rep. Massachusetts Institute of Technology, 1981.

[7] Kimin Yun and Jin Young Choi. "Robust and fast moving object detection in a non-stationary camera via foreground probability based sampling". In: *2015 IEEE International Conference on Image Processing*. 2015.

[8] Kwang Moo Yi et al. "Detection of Moving Objects with Non-Stationary Cameras in 5.8ms: Bringing Motion Detection to your Mobile Device". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2013.

[9] Kengo Makino et al. "Moving-object detection method for moving cameras by merging background subtraction and optical flow methods". In: *IEEE Global Conference on Signal and Information Processing*. 2017.

[10] Jonas Herzog. *Comparison of Methods for Moving Object Detection with a Moving RGB-D Camera*. Tech. rep. Mannheim Cooperative State University and German Aerospace Center, 2019.

[11] Gunnar Farnebäck. "Image Analysis". In: Springer, 2003. Chap. Two-frame motion estimation based on polynomial expansion, 363–370.

[12] Jerome Revaud et al. "EpicFlow: Edge-preserving interpolation of correspondences for optical flow". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1164–1172.

[13] Till Kroeger et al. "Fast Optical Flow using Dense Inverse Search". In: *Computer Vision - ECCV 2016: 14th European Conference, Amsterdam*. 2016.

[14] Bruce D. Lucas and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *International Joint Conference on Artificial Intelligence*. 1981, pp. 674–679.

[15] M. Jaimez et al. "Motion Cooperation: Smooth piece-wise rigid scene flow from RGB-D images". In: *International Confrerence on 3D Vision (3DV)*. 2015, pp. 64–74.

[16] M. Jaimez et al. "A primal-dual framework for real-time dense RGB-D scene flow". In: *Int. Conference on Robotics and Automation (ICRA)*. 2015, pp. 98–104.

[17] M. Jaimez et al. "Fast odometry and scene flow from RGB-D cameras based on geometric clustering". In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. 2017, 3992–3999.

[18] Raluca Scona et al. "StaticFusion: Background Reconstruction for Dense RGB-D SLAM in Dynamic Environments". In: *IEEE International Conference on Robotics and Automation*. 2018.

[19] Robert C. Leishman, Daniel Koch, and Timothy W. McLain. "Robust Motion Estimation with RGB-D Cameras". In: *Brigham Young University, Provo* (2013).

[20] S. Izadi et al. "KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera". In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. Research conducted at Microsoft Research, Cambridge, UK. Santa Barbara, CA, USA, 2011.

[21] F. Xu et al. "Motion Segmentation by New Three-View Constraint from a Moving Camera". In: *Jiangsu Key Laboratory of Spectral Imaging and Intelligent Sense, Nanjing University of Science and Technology, China* (2015).

[22] Soo Wan Kim et al. "Detection of moving objects with a moving camera using non-panoramic background model". In: *Machine Vision and Applications* 24 (2013).

[23] Yi-Chan Wu and Ching-Te Chiu. "Motion clustering with hybrid-sample-based foreground segmentation for moving cameras". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017.

[24] Yuxiang Sun, Ming Liu, and Max Q.-H. Meng, eds. *Motion Removal from Moving Platforms: An RGB-D Data-based Motion Detection, Tracking and Segmentation Approach*. IEEE Conference on Robotics and Biomimetics. Zhuhai, China, 2015.

[25] Herbert Bay et al. *Speeded Up Robust Features*. Tech. rep. ETH Zurich, Katholieke Universiteit Leuven, 2006.

[26] Pierre Del Moral. "Non Linear Filtering: Interacting Particle Solution". In: *Markov Processes and Related Fields* 2.4 (1996), 555–580.

[27] Yue Xu, Qing Guo, and Juan Chen. "Dynamic Object Detection Using Improved Vibe for RGB-D SLAM". In: *IEEE International Conference on Systems, Man, and Cybernetics*. 2018.

[28] Junjie Huang et al. "An Efficient Optical Flow Based Motion Detection Method for Non-stationary Scenes". In: *Institute of Automation, Chinese Academy of Sciences, Beijing* (2018).

[29] E. Ilg et al. "FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[30] C. Liu. "Beyond pixels: exploring new representationand applications for motion analysis". In: *Proceedings of th10th European Conference on Computer Vision: Part III*. 2009, 28–42.

[31] Yuxiang Sun, Ming Liu, and Max Q.-H. Meng. "Motion removal for reliable RGB-D SLAM in dynamic environments". In: *Robotics and Autonomous Systems* (2018).

[32] Jean-Philippe Guertin, Morgan McGuire, and Derek Nowrouzezahrai. "A Fast and Stable Feature-Aware Motion Blur Filter". In: *High Performance Graphics*. 2014.

[33] *OpenCV Documentation: Optical Flow*. 3.3.1. URL: https://docs.opencv.org/3.3.1/d7/d8b/tutorial_py_lucas_kanade.html.

[34] C. Wolf et al. "Evaluation of video activity localizations integrating quality and quantity measurements". In: *Computer Visions and Image Understanding*. 2014.

[35] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[36] J. Sturm, W. Burgard, and D. Cremers. "Evaluating Egomotion and Structure-from-Motion Approaches Using the TUM RGB-D Benchmark". In: *Proc. of the Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RJS International Conference on Intelligent Robot Systems*. 2012.

[37] Shuran Song and Jianxiong Xiao. "Tracking Revisited using RGBD Camera: Unified Benchmark and Baselines". In: *Proceedings of 14th IEEE International Conference on Computer Vision*. 2013.

[38] Xiaoqin Zhou et al. "Improving Video Segmentation by Fusing Depth Cues and the Visual Background Extractor (ViBe) Algorithm". In: *MPDI Sensors* (2017).