Performance enhancement of the vision system for the augmented reality Thymio robot

Michael A. Flückiger

Bachelor Thesis
September 2017

Dr. Stéphane Magnenat, Dr. Fabio Zünd
Prof. Dr. Robert W. Sumner

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Game Technology Center

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten Semester-, Bachelor- und Master-Arbeit oder anderen Abschlussarbeit (auch der jeweils elektronischen Version).

Die Dozentinnen und Dozenten können auch für andere bei ihnen verfasste schriftliche Arbeiten eine Eigenständigkeitserklärung verlangen.

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

Performance enhancement of the vision system for the augmented reality Thymio robot

Verfasst von (in Druckschrift):
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Flückiger	Michael Alexander

Ich bestätige mit meiner Unterschrift:
- Ich habe keine im Merkblatt „Zitier-Knigge" beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
Zürich, 3.September 2017	

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.

Abstract

The aim of this thesis is the improvement of the vision system of the *Thymio Programming Adventure*, an application which uses augmented reality to help children discover programming by letting them control an educational robot through a virtual world. The improvement is about the handling of multiple markers observed by the application and the use of its position to augment the observed world with virtual objects. In particular, the implementation of an appropriate data structure shall improve the vision system for cases where a strict subset of the markers is occluded. As a supplementary objective, experiments with the purpose to gather insights into the existing vision system are conducted.

To start with, a detailed statement of the problem is followed by a discussion of theoretical aspects important for this project. The thesis then presents the implementation of a data structure able to store and retrieve transformations followed by the description how this data structure can be integrated into the existing application to solve the stated problem. Lastly, the execution and results of experiments investigating the quality difference of the used markers and the functionalities of the implemented data structure can be found.

The implementation of the data structure presented does improve the vision system of the application. It allows to describe all virtual objects relative to one marker. Its position is known as long as at least one of multiple markers is visible, and therefore no reordering of the scene graph is necessary regardless what strict subset of markers is occluded. Through the conduction of experiments, a quality difference between the used markers is identified. The author recommends further investigation to determine the reason for the disparity since the quality of the marker is essential for the performance of the vision system.

Contents

List of Figures

1

Introduction

1.1 Thymio Programming Adventure

Thymio Programming Adventure [Thya] is a tablet application currently under development with the goal to provide a unique learning experience involving robotics and augmented reality. It helps children to discover programming through solving tasks with a *Thymio* [Thyb] robot. The tasks are embedded in an ongoing story where the robot is one of the main characters. The user observes the real world through a tablet whose video camera is used to detect different markers placed in the world. The markers are printed on paper and are representing different 2D objects, e.g. a house or a fountain. For every detected marker, a transformation between this marker and the camera can be determined. This transformation can then be used to render a 3D object on to the screen, that it looks like the object is part of the real world as illustrated in Figure 1.1. A good survey of the so called field of augmented reality is given by [BCL15].

As long as a marker is visible for the camera, its corresponding 3D objects can be added to the scene. But if a marker is hidden, for example through the robot driving in front of it, its position is no longer known, and augmenting the world with objects relative to this marker is no longer possible. In the case, where there is only one virtual object relative to this hidden marker, and the virtual object is drawn exactly at the position of the marker, this is not a problem. Because hiding a real world object corresponds in a natural way with the occlusion of a virtual object at the same position. But if the marker is needed to determine the position of multiple virtual objects, not necessarily drawn direct at the position of the marker, the occlusion of the marker leads to the disappearance of all these objects even if they are not hidden. Imagine a virtual protagonist walking through the scene whereby its position is given relative to a certain marker. If this marker is suddenly hidden, the protagonist disappears as well. The same is true when the tablet is slightly moved, such that the marker is no longer visible but the walking protagonist still should be. One possible solution to this problem is to express the position of the walking protagonist relative to another, still visible marker as soon as the primary marker gets hidden.

Figure 1.1: *The virtual world seen through the Thymio Programming Adventure application. The world is augmented with different objects depending on the markers currently detected. [Thya]*

This, however, requires a reordering of the scene graph, i.e. the description of how the virtual objects are related to each other. Another approach is to define one marker to be the so called world center marker, whose position relative to the camera is known as long as at least one of multiple markers is visible. The position of this world center marker can then be used as reference point for the positioning and drawing of virtual objects which are visible as long one arbitrary marker is detected. Meaning it is possible to express the whole virtual scene with a static scene graph and to draw this scene whenever one arbitrary marker is observed.

This project aims to implement the last-mentioned approach by pursuing the objectives defined in the following section.

1.2 Objectives of this work

The main goal of this project is the development of a data structure named *TransMem*, which allows to store and retrieve transformations over time. With its help, the application should be able to maintain the position of a world center marker. Therefore, objects can be drawn as long as at least one marker is visible. To show that *TransMem* makes this possible, it needs to be integrated into an algorithm which operates as a connection between the tracker, the data structure and the drawing engine. The development of such an algorithm and the data structure *TransMem* is described in Chapter 3. An experiment to prove that *TransMem* fulfills the desired task is described in Section 4.4. A reminder about the most important mathematical concepts used in this project as well as a description of the functionality of the algorithm responsible for the detection of the markers is given in Chapter 2. An additional goal is to improve the quality of the tracking by including the information obtained from an orientation sensor. This improvement is discussed in Section 3.4. During the project, a difference in the quality of the various markers was observed. The determination of this quality difference is therefore stated as an additional goal. The outcome of the investigation is described in Section 4.2, whereby the

similarity measure used in the experiments is introduced in Section 2.2.

1.3 Related work

The development and design of *TransMem* is inspired by *ROS' tf* library [Foo13]. This library is part of the Robot Operating System *ROS* [QCG+09] and helps to keep track of multiple coordinate frames in robotic applications. *TransMem* represents its data internally in a graph. The *Boost Graph Library* [Boo] is a state-of-the-art generic graph library. In this project, quaternions are averaged simply by averaging its components. A more sophisticated approach for a weighted averaging of multiple quaternions is presented in [MCCO07]. As a more comprehensive source of this topic serves [HTDL13]. The tracking of an augmented reality application can be improved by a fusion of the vision and the inertial measurement unit as described in [KVR+14]. The necessary computation of the position of these two devices relative to each other can, for example, be done with [KS10] or [Jor]. An overview and comparison of different metrics for three-dimensional rotation matrices are presented in [Huy09].

2

Theory

This chapter refreshes the basic mathematical concept of coordinate transformations, as this is an important basis for the implementation described in the next chapter. Furthermore, a simple approach to compare transformation matrices is introduced. The presented similarity measure is used in the examination of the marker quality (See Section 4.2.) and in the design of the example marker model (See Chapter 3.3.). Finally, a description of the functionality of the algorithm used to track the marker is given. The tracking algorithm itself is not touched by this project, but since it is essential for the functioning of the *Thymio Programming Adventure* an overview is given.

2.1 Coordinate transformations

In computer graphics, as well as in robotics, there is often the need to transform the coordinates of a point in terms of one frame into its representation in terms of another.

Let $\Lambda = (\vec{b_x}, \vec{b_y}, \vec{b_z}, \vec{O_\Lambda})$ be a coordinate frame formed by the orthonormal basis vectors $\vec{b_x}, \vec{b_y}, \vec{b_z}$.

Let $\Upsilon = (\vec{r_x}, \vec{r_x}, \vec{r_x}, \vec{O_\Upsilon})$ be another frame with its orthonormal basis vectors $\vec{r_x}, \vec{r_y}, \vec{r_z}$ and its origin $\vec{O_\Upsilon}$ described relative to Λ. Let $\vec{p_\Upsilon} = (c_x, c_y, c_z)^\intercal$ be an arbitrary point in Υ.

The coordinates of $\vec{p_\Upsilon}$ relative to Λ can be calculated as described in the following equation:

$$\vec{p_\Lambda} = R \cdot \vec{p_\Upsilon} + \vec{t} = [\vec{r_x}\ \vec{r_y}\ \vec{r_z}] \cdot \vec{p_\Upsilon} + \vec{O_\Upsilon} \tag{2.1}$$

Since R consists of the orthonormal basis vectors of Υ, R fulfills the properties of a rotation matrix, i.e. $det(R) = \pm 1$ and $R^\intercal = R^{(-1)}$.

This means that the transformation of point coordinates between two frames is described by a

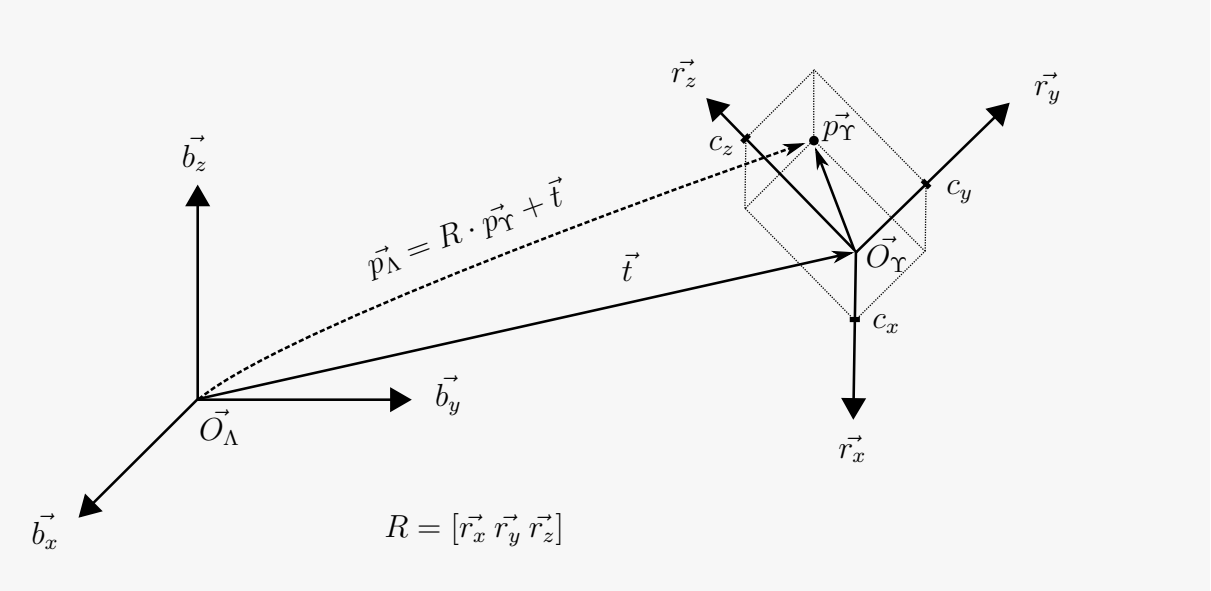

Figure 2.1: *Illustration of a coordinate transformation. An arbitrary point $\vec{p_\Upsilon}$ relative to a frame Υ can be described relative to another frame Λ through a rotation followed by a translation.*

rotation R followed by a translation $\vec{t}$. If homogeneous coordinates are used, this transformation can be expressed in one matrix $T_{\Lambda \leftarrow \Upsilon}$.

$$\begin{bmatrix} \vec{p_\Lambda} \\ 1 \end{bmatrix} = T_{\Lambda \leftarrow \Upsilon} \cdot \begin{bmatrix} \vec{p_\Upsilon} \\ 1 \end{bmatrix} = \begin{bmatrix} R & \vec{t} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \vec{p_\Upsilon} \\ 1 \end{bmatrix} \tag{2.2}$$

Furthermore, it is obvious that multiple transformations can be combined to obtain a new transformation $T_{f_n \leftarrow f_1}$ describing a direct mapping between the two frames f_1 and f_n.

$$T_{f_n \leftarrow f_1} = T_{f_n \leftarrow f_{n-1}} \cdot \ldots \cdot T_{f_3 \leftarrow f_2} \cdot T_{f_2 \leftarrow f_1} \tag{2.3}$$

2.2 Similarity measure for transformations

This section presents a simple approach for a measure $\phi(T_1, T_2)$ describing the similarity of two transformations T_1, T_2. As discussed in Section 2.1 any transformation T between two frames can be described through a rotation matrix R and a translation vector $\vec{t}$. Furthermore, every translation $\vec{t}$ can be described as $\vec{t} = \vec{d} \cdot \kappa$ where the unit vector $\vec{d}$ encodes the direction and κ the magnitude of the translation, whereas every rotation can be described through an unit vector representing the rotation axis $\vec{a}$ and an angle value φ.

Let $T_1 := (R_1, \vec{t_1}) = ((\vec{a}_1, \varphi_1), (\vec{d_1}, \kappa_1))$ and $T_2 := (R_2, \vec{t_2}) = ((\vec{a}_2, \varphi_2), (\vec{d_2}, \kappa_2))$ be the two transformations intended to compare. T_1, T_2 are equal, if both consist of identical rotation and translation parts. This means, ϕ can be composed of two measures ϕ_R and ϕ_T, where ϕ_R is

a measure for the similarity of the rotational parts and ϕ_T a measure for the similarity of the translational parts.

$$\phi := f_{C1}(\phi_R, \phi_T) \tag{2.4}$$

Similarity measurement for rotations. Two rotations R_1, R_2 are equal if they describe a rotation around the same axis by the same angle. This means, ϕ_R can be composed of Δ_a and Δ_φ to describe the similarity of two rotations.

$$\phi_R := f_{C2}(\Delta_a, \Delta_\varphi) \tag{2.5}$$

$$\Delta_a := ||\vec{a_1} - \vec{a_2}|| \in [0, 2] \qquad \Delta_\varphi := \sqrt{2 \cdot (1 - \cos(\varphi_1 - \varphi_2))} \in [0, 2] \tag{2.6}$$

Δ_a is the distance between the two normalized rotation axes $\vec{a_1}$ and $\vec{a_2}$. The more similar the rotation axes, the smaller the value of Δ_a.

Δ_φ is the distance between a unit vector perpendicular to the rotation axis and the same vector rotated by the difference of the two angles $\varphi_1 - \varphi_2$. Then it holds: The smaller the difference between the two angles, the smaller the value of Δ_φ.

Similarity measurement for translations. Two translations $\vec{t_1}$, $\vec{t_2}$ are equal if both translate in the same direction by an equal magnitude. ϕ_T is composed of Δ_d and Δ_κ to describe the similarity of two translations.

$$\phi_T := f_{C3}(\Delta_d, \Delta_\kappa) \tag{2.7}$$

$$\Delta_d := ||\vec{d_1} - \vec{d_2}|| \in [0, 2] \qquad \Delta_\kappa := \left\{ \begin{array}{ll} 2 \cdot (1 - \frac{\kappa_1}{\kappa_2}), & \text{if } \kappa_1 < \kappa_2 \text{ and } \kappa_2 \neq 0 \\ 2 \cdot (1 - \frac{\kappa_2}{\kappa_1}), & \text{if } \kappa_2 < \kappa_1 \text{ and } \kappa_1 \neq 0 \\ 0, & \text{if } \kappa_1 = \kappa_2 \\ 2, & \text{else} \end{array} \right\} \in [0, 2] \tag{2.8}$$

Δ_d is the distance between the two normalized translation directions $\vec{d_1}$ and $\vec{d_2}$. The more similar the directions of the translation, the smaller the value of Δ_d.

Δ_κ is the ratio of the two magnitudes κ_1 and κ_2 explicit mapped to $[0, 2]$. The smaller the difference between the magnitudes, the smaller the value of Δ_κ.

Similarity measurement ϕ_4. From the Equations 2.4, 2.5 and 2.7 follows the similarity measure ϕ:

$$\phi(T_1, T_2) := f_{C1}(f_{C2}(\Delta_a, \Delta_\varphi), f_{C3}(\Delta_d, \Delta_\kappa)) \tag{2.9}$$

ϕ is a combination of Δ_a, Δ_φ, Δ_d and Δ_κ whereby the combination is described through the three functions f_{C1}, f_{C2} and f_{C3}.

For this project the functions are chosen in such a way, that ϕ_4 represents all four measures as a vector.

$$\phi_4(T_1, T_2) := (\Delta_a,\ \Delta_\varphi,\ \Delta_d,\ \Delta_\kappa)^\intercal \tag{2.10}$$

As it is possible to choose any other combination of the functions f_{C1}, f_{C2} and f_{C3}, it allows to create different definitions of the similarity measure based on Δ_a, Δ_φ, Δ_d and Δ_κ.

2.3 Tracking algorithm

This section describes the functionality of the tracking algorithm which is already implemented in the *Thymio Programming Adventure*. The tracker processes successively the images of the video stream. Let f_c be the current image processed by the tracker and let m be the only marker which is tracked by the system.

Key point detection using *BRISK*. In a first step, it is determined if m is already tracked. If there exists no information from processing the previous image f_{c-1} the marker m is considered to be not tracked. In this case *BRISK* [LCS11] is used to detect and compute the keypoints K_c in the image f_c. These keypoints are then matched against the template key points of the marker m using a threshold criterion.

Approximation of the homography H_c using optical flow. On the contrary, the marker m is considered to be already tracked if there exists a homography H_{c-1} between the template of the marker m and its image in f_{c-1}. In this case, there also exists a set of key points K_{c-1} detected in the previous image. With a pyramidal approach of the *Lucas Kanade Feature Tracker* [yB00] the optical flow of f_{c-1} and f_c is calculated and used to approximate the new position of the keypoints K_{c-1} in the current image f_c. The homography H_{c-1} is then used to make sure the new position of these keypoints is good enough. This is done by comparing the region around the new position of each keypoint with the region around the corresponding template keypoint, transformed by H_{c-1}. The comparison is done with a template matching using *Normalized Cross Correlation* [BH01]. If there are enough key points for which the comparison yields satisfying result, the current homography can already be approximated. This approximation $\tilde{H}_c$ is set to H_{c-1}: $H_{c-1} = \tilde{H}_c$.

Calculation of H_c using active search. By using the homography H_{c_1} the position of each keypoint of K_{c-1} in the current image f_c is calculated. Furthermore is H_{c_1} used to map a patch around each corresponding template keypoint into the image space. Note that the possible approximation $\tilde{H}_c$ described in the previous subsection comes into effect here. Then the refined

position of each keypoint is determined. This is realized by template matching a keypoints corresponding wrapped patch in a window around the keypoints transformed position. The template matching is also done with an implementation of [BH01] and regarded valid if a certain threshold criterion is met.

Calculation of the pose and the confidence. After proceeding either the strategy for an already tracked or a newly tracked marker, a set of keypoints K_c is given. For these keypoints, the position in the template image of the marker as well in the current image f_c is known and their correspondence is reasonable high. If there are enough keypoints in K_c the homography H_c can be calculated and then be used to estimate the current pose of the marker m. This is done by transforming the corners of the marker from template space into the current image space and solving the *Perspective-n-Point (PNP)* problem. Concluding the confidence of the current pose estimation is calculated. This is simply the number of keypoints in K_c divided by the number of all template keypoints of this marker. So the higher the confidence value of a pose estimation the more keypoints of a marker were detected reliably in the current image f_c. If there are not enough keypoints K_c the pose of the marker is set to the identity matrix and the confidence is set to zero.

3

Implementation

This chapter first illustrates the importance of being able to store and retrieve transformations over time in the context of the presented problem which this work aims to solve (See Section 1.1 and Section 1.2.). Subsequently, the implementation of a data structure which allows the storage and retrieval of transformations is presented followed by the description of a so called marker model. This model makes use of the data structure and provides the desired functionality of being able to maintain the position of a world center marker as long as at least one of multiple markers is observed. A short discussion about integrating a rotation sensor to improve the tracking concludes this chapter.

3.1 Initial situation

As an initial system, a camera continuously observes a real world scene with multiple markers located in it. The transformation $T_{cam \leftarrow m_x}$ mapping from the position of a visible marker m_x to the camera in 3D space is determined and updated by a tracker (See Section 2.3.). These transformations between marker and camera can then be used to augment the observed world by drawing virtual objects relative to the markers position. But if a marker gets occluded or is no longer visible, the virtual objects described relative to this marker can no longer be drawn. As already mentioned in Chapter 1, this is especially unpleasant in the case where a virtual object is projected not directly at the markers position but somewhere relative to it. In this case hiding the marker is not consistent with the disappearance of the virtual object. On the left-hand side of the Figure 3.1, an illustration of the initial situation with three markers m_0, m_1 and m_2 observed by the camera is shown. At the position of each marker, the scene is augmented with a virtual object. Additionally, there is a virtual object O_{m_0} drawn relative to the marker m_0. If the marker m_0 gets occluded, as illustrated on the right-hand side of Figure 3.1, the virtual object O_{m_0} disappears since its position relative to the camera is no longer known.

As long as at least one of multiple markers is available, it would be possible to reorder the scene graph, i.e. describing the disappearing virtual objects relative to this still visible marker, to solve the problem.

Aim of this work. This work aims to implement a solution which does not require this reordering, by defining a world center marker m_0 whose position relative to the camera can be determined as long as at least one marker is visible. This marker m_0 can then be used as reference point for the positioning of all virtual objects in the scene as well as for the positioning of the virtual camera necessary to render the scene.

The position of the markers relative to each other is assumed to be fixed. Therefore, the position of m_0 can be calculated if the fixed transformation $T^f_{m_x \leftarrow m_0}$ between a possibly hidden marker m_0 and a still visible marker m_x is known: $T_{cam \leftarrow m_0} = T_{cam \leftarrow m_x} \cdot T^f_{m_x \leftarrow m_0}$. The fixed transformation $T^f_{m_x \leftarrow m_0} = T^t_{m_x \leftarrow cam} \cdot T^t_{cam \leftarrow m_0}$ can be computed by using $T^t_{m_x \leftarrow cam}$ and $T^t_{cam \leftarrow m_0}$ obtained for a point at time t, where both markers m_0 and m_x were visible. Hence, by storing every update $T^t_{cam \leftarrow m_v}$ obtained at time t for every marker m_v which is visible, the fixed transformation between any two marker which were observed at the same time, can be calculated. And with this the position of m_0 can be calculated as long as one arbitrary marker is visible. Figure 3.2 illustrates the just explained situation. On the left-hand side, the world center marker m_0 and another marker m_x are visible at the same time t. At this time the fixed transformation $T^f_{m_x \leftarrow m_0}$ can be calculated and stored. On the right-hand side a later situation where m_0 is occluded is shown. Yet the position of the hidden marker relative to the camera can still be calculated, since the current position of m_x and $T^f_{m_x \leftarrow m_0}$ is known.

Having a data structure, which allows the storage and retrieval of transformation over time, should allow solving the problem explained in this section.

3.2 TransMem

In this section, a possible implementation of a data structure is introduced, which allows storing and retrieving transformations over time. The data structure is named *TransMem* as an abbreviation for transformation memory. The source code of the implementation can be found on *GitHub*[1]. The three main functionalities of *TransMem*; storing a transformation, query for a transformation and query for a best transformation are discussed in the following subsections. Some remarks about the implementation are made afterward.

3.2.1 Storing a transformation

Let $T_{dstId \leftarrow srcId}$ be a transformation valid at time t. The source $srcId$ and the destination $dstId$ of the transformation are each described by a frame identifier. *TransMem* is able to store such a transformation $T_{dstId \leftarrow srcId}$ together with the time stamp t for a certain duration $dstor$. The duration $dstor$ is chosen by the user, but it cannot be zero since this is not a reasonable timespan to store something.

[1] https://github.com/fluckmic/transmem

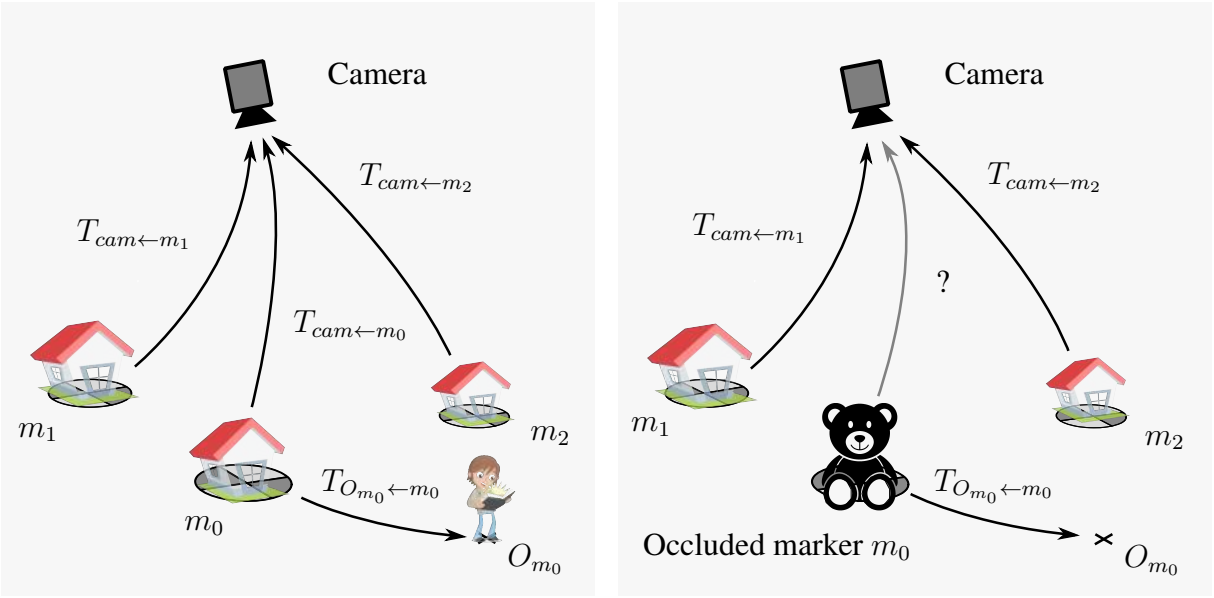

Figure 3.1: *As long as the marker m_0 is visible, the virtual objects described relative to m_0 can be drawn (l.). If hidden (r.) then the mapping between the marker m_0 and camera is no longer known, therefore the position of a virtual objects described relative to m_0 can no longer be determined which thus disappear.*

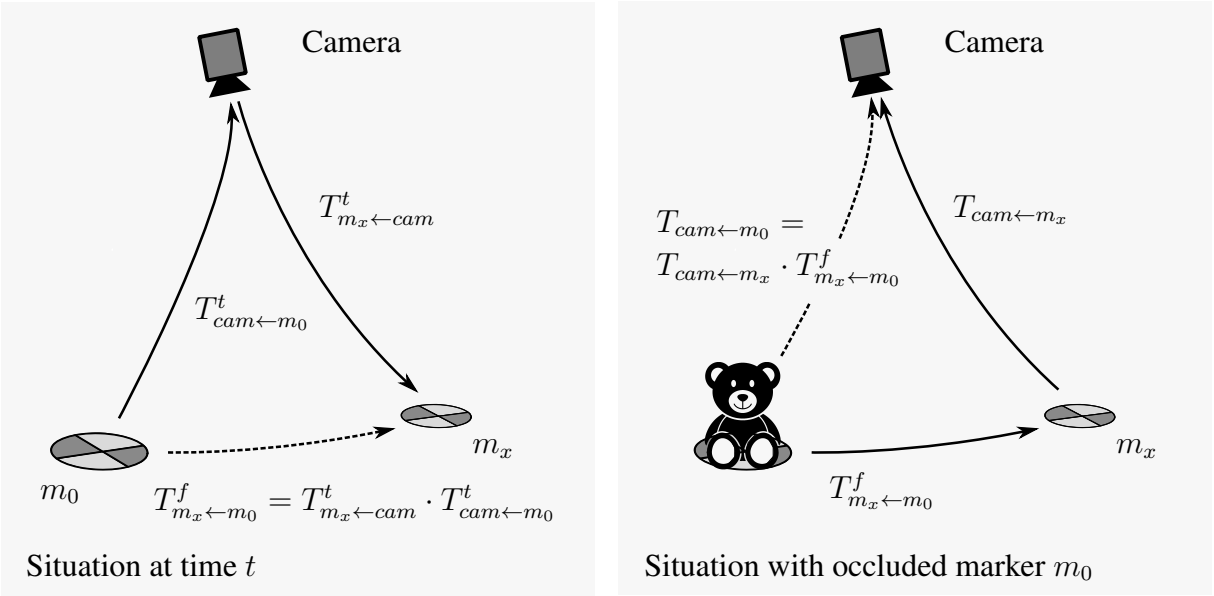

Figure 3.2: *If the world center marker m_0 and another marker m_x are visible at the same time t, the fixed transformation $T^f_{m_x\leftarrow m_0}$ can be calculated and stored (l.). The stored transformation can then be used in a later situation where m_0 is occluded (r.) to calculate the position of the hidden marker relative to the camera (r.).*

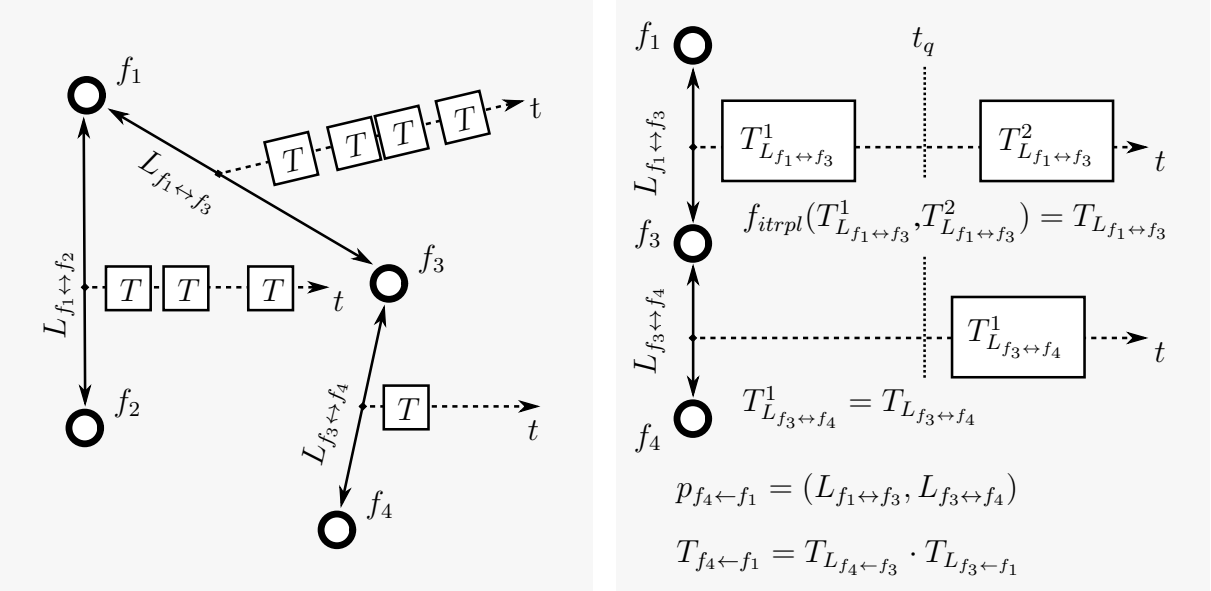

Figure 3.3: *The figure on the left illustrates the graph after a number of link updates. On each link there is at least one transformation stored, describing a mapping between the frames connected trough the link. The illustration on the right is an example for a path $p_{f_4 \leftarrow f_1}$ along which a transformation $T_{f_4 \leftarrow f_1}$ is calculated choosing the corresponding transformation T_L on each link.*

The storage is internally organized as a graph. This design emerges in a coherent way from the nature of the problem, which should be tackled with this data structure. The nodes of the graph are called frame where each node represents a frame identifier. A new frame is created for every unknown frame identifier encountered when a new transformation is stored in *TransMem*. The edges of the graph are called links. A link between two frames stores the transformations, all with a different time stamp, mapping between these frames. Two frames f_1, f_2, are connected through a link $L_{f_1 \leftrightarrow f_2}$ if it stores at least one transformation. The very first saving of a transformation between two frames, i.e. the creation of a new link, is called a registration of a link. Storing further transformations on an existing link is named a link update. On the left-hand side of the Figure 3.3 such a graph is illustrated. After insertions of transformations mapping between the frames f_1 and f_2, f_1 and f_3, as well as f_3 and f_4 the graph exists of four nodes representing these four different frames. The nodes are connected according to these mappings.

It is further possible to store a confidence value on each existing link. The confidence value on a certain link can either be set directly together with the storage of a new transformation on this link or independent of it. The user determines the meaning of the confidence value.

As already mentioned, *TransMem* stores the transformation just for a certain time. Whenever a link is updated, all stored transformations on that link, which are too old, are removed. A transformation is considered to be too old, if the time difference between its time stamp and the time stamp of the newest transformation stored on the link is larger than the duration $dstor$. Every existing link is guaranteed to contain at least one transformation, since the measurement which determines whether a transformation is removed is relative to a transformation, and $dstor$ can not be zero. Therefore, neither links nor frames in the graph are ever removed.

3.2.2 Query for a transformation

TransMem can be queried for a transformation $T_{dstId \leftarrow srcId}$ mapping from a source $srcId$ to a destination $dstId$ valid at query time t_q. As in Subsection 3.2.1, $srcId$ and $dstId$ are represented by a frame identifier.

The replay of the query can be divided into three steps.

Step 1 - Path searching. A path is an ordered set of links, $p_{f_n \leftarrow f_1} = (L_{f_1 \leftrightarrow f_2}, L_{f_2 \leftrightarrow f_3}, .., L_{f_{n-1} \leftrightarrow f_n})$ connecting the two frames f_1 and f_n in the graph. In order to calculate a transformation from $srcId$ to $dstId$, a path $p_{dstId \leftarrow srcId}$ needs to exist. Only in this case it is possible to combine the transformations stored in the paths links together to the requested transformation. If such a path is not existing *TransMem* is not able to answer the query successfully.

In the first step *TransMem* searches for a shortest path $p_{dstId \leftarrow srcId}$. The shortest path is computed by means of the number of links. Note that it is unnecessary to recalculate a path for every query as long as no new links are registered. Therefore, all calculated paths are cached by *TransMem* until a new link is registered.

Step 2 - Choosing a transformation of a single link. In the second step, for each link $L \in p_{dstId \leftarrow srcId}$ a transformation T_L valid at time q_t needs to be extracted. These transformations are then used in the third step to calculate the requested transformation.

- If the query time t_q is smaller than the time stamp of the oldest transformation stored on L, this oldest transformation is used as T_L.
- If the query time t_q is larger than the time stamp of the newest transformation stored on L, this newest transformation is used as T_L.
- If the query time t_q lies in between the time stamp of two stored transformations T_L^1 and T_L^2, T_L is obtained by interpolating between these two transformations T_L^1 and T_L^2.
- If the query time t_q is equal to the time stamp of a stored transformation, this transformation is used as T_L.

Step 3 - Combination of transformations. In the third and final step, the transformations $T_{srcId \leftrightarrow f_1}, T_{f_1 \leftrightarrow f_2}, .., T_{f_{n-1} \leftrightarrow dstId}$ along the path $p_{dstId \leftarrow srcId}$ can be combined to the requested transformation (See Equation 3.1.). It is obvious, that these transformations can be used to map in either direction. Therefore possible inversions are not mentioned explicitly.

$$T_{dstId \leftarrow srcId} = T_{dstId \leftarrow f_{n-1}} \cdot T_{f_{n-1} \leftarrow f_{n-2}} \cdot ... \cdot T_{f_1 \leftarrow srcId} \tag{3.1}$$

On the right-hand side of Figure 3.3 a path $p_{f_4 \leftarrow f_1}$ between the frame f_1 and the frame f_4 is illustrated. On each link of the path the corresponding transformation T_L is chosen with regard to the query time t_q. For $T_{L_{f_1 \leftrightarrow f_3}}$ this is the interpolation of the two closest stored transformations, for $T_{L_{f_3 \leftrightarrow f_4}}$ the only available one stored on that link. These transformations are then combined to the mapping $T_{f_4 \leftarrow f_1}$.

For each successful query *TransMem* calculates two values which allow the user to estimate the quality of the obtained transformation $T_{dstId \leftarrow srcId}$. The first value is the average link confidence, which is defined as the average of the confidence of all links used for the calculation of

$T_{dstId \leftarrow srcId}$. The second value is defined as the largest distance between any time stamp of a stored transformation T_L used in any way for the calculation of the transformation $T_{dstId \leftarrow srcId}$.

3.2.3 Query for a best transformation

TransMem can also be queried for a best transformation $T_{dstId \leftarrow srcId}$ mapping a source $srcId$ to a destination $dstId$. Similar to the retrieval of a transformation described in Subsection 3.2.2, the first step is the computation of a path $p_{dstId \leftarrow srcId}$. For this query, the user does not specify at what time t_b the desired best transformation should be valid. Therefore, in a second step *TransMem* calculates this point in time, whereby the best refers to the minimizing characteristic of t_b.

Let $\delta_L = |t_b - t_n|$ be the distance between t_b and the time stamp t_n of the next closest transformation T_n stored on the link L. *TransMem* chooses t_b, such that t_b minimizes the sum of these quadratic distances over all links of a given path. I.e., t_b minimizes $\sum \delta_L^2$ over all $L \in p_{dstId \leftarrow srcId}$.

After the calculation of t_b, the query is processed similarly as described in Subsection 3.2.2. For each link of the path, a transformation T_L is chosen. Finally, $T_{dstId \leftarrow srcId}$ is computed by combining these transformations. For a best transformation query, the same additional quality information as for a query described in the previous section is provided by the data structure. *TransMem* caches every successful response $T_{dstId \leftarrow srcId}$ until all links along the path $p_{dstId \leftarrow srcId}$ are updated again and all these updates lie within an interval. The user specifies the length of this time interval.

3.2.4 Remarks about the implementation

The data structure is implemented in *C++* and makes use of *Qt* types for the representation of transformations as well as for some containers. Additional to the main functionalities there is some debug functionality implemented. The current state of *TransMem* including all transformations currently stored on all links can be dumped as a *JSON* file or just the structure of the graph as a *GraphML* file. Furthermore is there an implementation of an unit test which is also accessible on the *GitHub*[2] repository.

[2] https://github.com/fluckmic/transmem

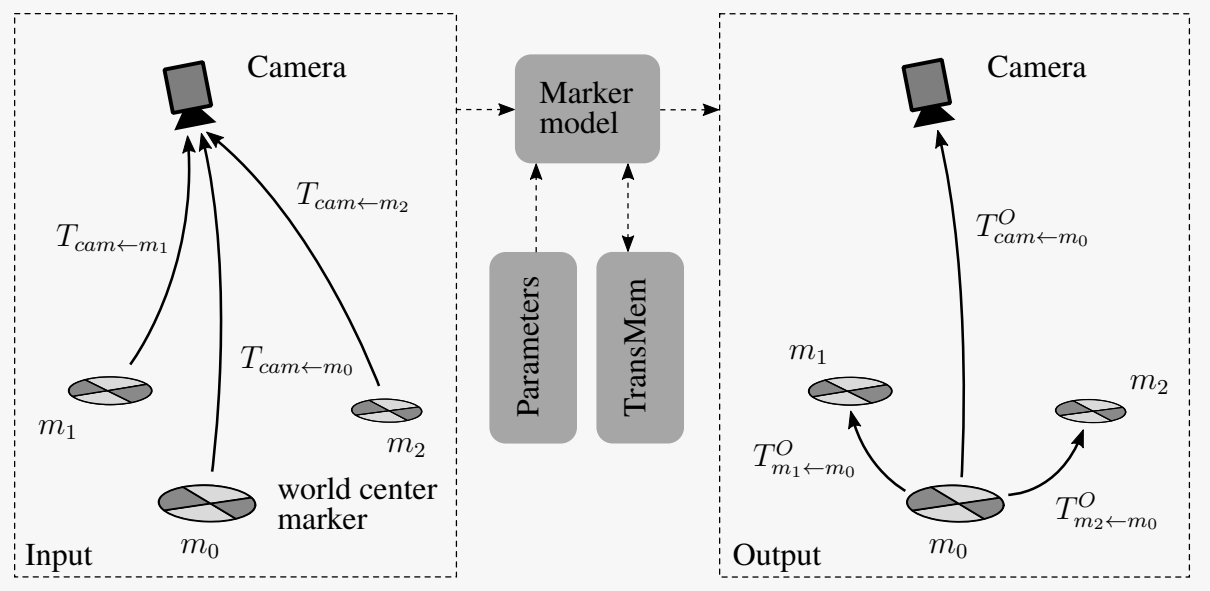

Figure 3.4: *A marker model creates a new relationship between multiple markers. It takes as an input a number of markers whose positions relative to a tracking device are regularly updated and creates a new relationship between these markers as an output.*

3.3 Marker model

This section is about using *TransMem* to achieve the pursued objective of being able to express the position of a world center marker whenever at least one of multiple markers is visible. A general overview of what is understood by a marker model is given first, followed by the implementation details.

3.3.1 What is a marker model?

A marker model is a functionality that takes a number of markers as an input and produces as an output a certain description of how the different marker are related to each other. The position of each marker relative to a tracking device is updated regularly and available for the marker model. In the following subsection, the implementation of a simple approach for a marker model which makes use of *TransMem* is described. The output generated by this model fulfills the pursued goal described in Subsection 3.1.

3.3.2 Implementation of a simple marker model

As already mentioned above, a marker model takes as input a number of markers and produces a new relationship between these markers as an output. In the first subsequent part of this section, the input, as well as the output of the simple marker model, is described. Following this, the two core functionalities of the implementation are described. This is the receiving part, which handles the position updates of each marker and the updating part, which is responsible

for the generation of the output. The relation between input and output of the marker model is described by the Figure 3.4. The indices used in the following sections correspond to the indices used in this illustration.

Input and output of the marker model

Input of the marker model. Multiple markers $m_0, .., m_n$ serve as input of the marker model. A tracker provides regularly updates about its position $T_{cam \leftarrow m_j}$ relative to the camera for every marker m_j for $0 \leq j \leq n$. Every transformation update comes together with a confidence value $c \in [0, 1]$. The higher this confidence value, the more likely the transformation update represent the real transformation between the marker and the camera. In the next step, one of these markers needs to be defined as the world center marker. For the remainder of this description, m_0 takes over this function. Additionally, the value of five parameters, which affect the behavior of the model, needs to be defined. These parameters are named and discussed in the upcoming parts of this chapter.

Output of the marker model. As output, the marker model produces a description of the position $T^O_{m_i \leftarrow m_0}$ of every marker m_i for $1 \leq i \leq n$ relative to the world center marker m_0. Additionally, a mapping $T^O_{cam \leftarrow m_0}$ between the world center and the camera is produced. The calculation and the recalculation, respectively, of the output, happens whenever triggered by the user. There is no other way to cause the (re)-generation of the output. The mapping $T^O_{cam \leftarrow m_0}$ is updated if at least one marker is visible, whereas the mapping $T^O_{m_i \leftarrow m_0}$ is updated if the marker m_i is visible. Whether a marker is visible or not is determined by the marker model. It is assumed that the position of the markers relative to each other is fixed. Furthermore a marker is not visible until it has been seen together with the world center marker once.

Receiving part

For every marker m_j regular updates of $T_{cam \leftarrow m_j}$ together with a confidence value c are received. If the confidence c is larger than the update threshold *TH-CONF-UPDATE* the transformation is considered as good. The transformation is stored in *TransMem* and the confidence of the link $L_{cam \leftrightarrow m_j}$ is updated. If c is smaller than the threshold just the confidence on the link is updated without storing the transformation. It needs to be considered that the confidence value of the link $L_{cam \leftrightarrow m_j}$ reflects how well the tracker currently observes the marker m_j.

Updating part

Whenever the user triggers the updating part, the output of the marker model is recalculated. The updating process consists of the determination whether a certain marker in the output is visible, and the computation of the transformations $T^O_{m_i \leftarrow m_0}$ and $T^O_{cam \leftarrow m_0}$.

Determination of visibility. For every marker m_j the model determines, if the marker is currently visible or not. If for a marker m_j, the confidence stored on the link $L_{cam \leftrightarrow m_j}$ is lower than the threshold *TH-CONF-VISIBLE*, the marker is considered to be hidden. The case where

the confidence drops below this threshold reflects the situation when the marker is clearly obscured. Therefore by choosing the value of the visibility threshold *TH-CONF-VISIBLE* it is decided what it means for a marker to be occluded.

As a second criterion for the visibility of a marker, the marker model considers the amount of time since the last update on the link $L_{cam \leftrightarrow m_j}$ occurred. If there was no update within this time *TH-LAST-UPDATE* the marker is also considered to be hidden. There are two intentions behind this second criterion. First, some sort of margin is needed, even if the marker m_j under consideration is currently tracked with a confidence higher than the update threshold TH-CONF-UPDATE. The point in time, when the last position update was inserted into *TransMem* and the time of visibility determination might differ even if both happen frequently. Second, a simple approach of temporal smoothing shall be implemented. A marker is considered to be still visible if the confidence of a marker drops below the update threshold but stays above the visibility threshold for just a few updates.

As a third criterion for marker visibility, it is required that *TransMem* stores a good fixed transformation $T^f_{m_i \leftarrow m_0}$ mapping from the world center marker to the marker m_i. The fixed transformation is considered to be good, if the two markers m_0 and m_i are both visible within a time span expressed by the threshold *TH-FIX-INTERVAL*. Once such a transformation exists, i.e. the two markers were observed together, this third criterion is always fulfilled due to the storing ability of *TransMem*.

Calculation of $T^O_{cam \leftarrow m_0}$. For every visible marker m_j there exists a sufficient good transformation $T^C_{cam \leftarrow m_j}$ at the point in time where the update was triggered. Furthermore, for every visible marker which is not the world center marker, a transformation $T^f_{m_i \leftarrow m_0}$ exists.

So if at least one marker is visible, the mapping $T^O_{cam \leftarrow m_0}$ can be updated, since either the Equation 3.2 or the Equation 3.3 can be used.

$$T^O_{cam \leftarrow m_0} = T^C_{cam \leftarrow m_0} \tag{3.2}$$

If the world center marker m_0 is visible, the current transformation between the world center marker and the camera can directly be used for the update (See Equation 3.2.).

$$T^O_{cam \leftarrow m_0} = T^C_{cam \leftarrow m_i} \cdot T^f_{m_i \leftarrow m_0} \tag{3.3}$$

The update can also be done by using the current transformation between a visible marker m_i and the camera together with the stored fixed transformation (See Equation 3.3.). If multiple marker are visible, multiple options $T^{O_1}_{cam \leftarrow m_0}, .., T^{O_n}_{cam \leftarrow m_0}$ for the searched transformation can be calculated. In this case, the possibilities are averaged as described in Equation 3.4. The averaging mechanism *avgIfEq* itself is described in a later part of this chapter.

$$T^O_{cam \leftarrow m_0} = \textit{avgIfEq}(\ .. \textit{avgIfEq}(T^{O_1}_{cam \leftarrow m_0},\ T^{O_2}_{cam \leftarrow m_0})\ ..\ ,\ T^{O_n}_{cam \leftarrow m_0}) \tag{3.4}$$

Calculation of $T^O_{m_i \leftarrow m_0}$. As already mentioned, for every visible marker m_i there exists a sufficient transformation $T^C_{cam \leftarrow m_i}$ as well as a fixed transformation $T^f_{m_i \leftarrow m_0}$. From the previous

part it is also known that if at least one marker is visible, it can determine the current transformation mapping from the world center to the camera. Therefore, the current transformation $T^C_{m_i \leftarrow m_0}$ mapping from the world center to the marker m_i can be calculated without having to query *TransMem* again.

$$T^C_{m_i \leftarrow m_0} = T^C_{m_i \leftarrow cam} \cdot T^O_{cam \leftarrow m_0} \tag{3.5}$$

If the fixed transformation and the current transformation are similar enough, they are averaged to obtain $T^O_{m_i \leftarrow m_0}$. If not, only the current transformation is used. This averaging mechanism *avgIfEq* is described in the following part.

$$T^O_{m_i \leftarrow m_0} = \textit{avgIfEq}(T^C_{m_i \leftarrow m_0},\ T^f_{m_i \leftarrow m_0}) \tag{3.6}$$

Averaging transformations. As described in the previous two parts, there may situations where two options T^{O_1} or T^{O_2} can be chosen from for a result T^O. The marker model uses a simple approach to average the transformations in case are somehow equal. This functionality is described by Equation 3.7 and Equation 3.8 together with the similarity measure ϕ_4 for transformations as defined in Section 2.2.

$$\textit{avgIfEq}(T^{O_1},\ T^{O_2}) := \{ \begin{array}{ll} avg(T^{O_1},\ T^{O_2}), & \text{if} \quad \phi_4(T^{O_1}, T^{O_2}) \check{<} \textit{TH-TRANS-EQ} \\ T^{O_1}, & \text{else} \end{array} \} \tag{3.7}$$

$$\vec{a} \check{<} \vec{b} := (a_1, .., a_n) \check{<} (b_1, .., b_n) := \{ \begin{array}{ll} true, & \text{if} \quad a_i < b_i \quad \forall i\ 1 \leq i \leq n \\ false, & \text{else} \end{array} \} \tag{3.8}$$

The threshold *TH-TRANS-EQ* is a four-dimensional vector, which is used as the criterion to decide whether two transformations are similar enough to be averaged. The fusion of two transformations $avg(T_1, T_2)$ is done by averaging the rotation quaternion and the translation vectors of the two transformations component wise. The obtained rotation quaternion is renormalized afterward. The intention behind the averaging is to try to reduce the scatter of the calculated transformations. Chapter 4.3 describes an experiment where the effect of the averaging is evaluated.

3.4 Rotation sensor support

If a device is equipped with a rotation sensor it can be used to improve the tracking. The tracking algorithm delivers transformation updates, which are, due to the relatively high computational costs of the tracking, much less frequent than updates from a rotation sensor. The idea is to correct the last transformation obtained from the tracker with the rotation values from the sensor until the next update from the tracker arrives.

$$T_{cam\leftarrow m}^{curr} = T_{intr}^{(-1)} \cdot T_{\delta R} \cdot T_{intr} \cdot T_{cam\leftarrow m}^{last} \tag{3.9}$$

In Equation 3.9, $T_{cam\leftarrow m}^{last}$ represents the transformation obtained from either the tracker or through a previous correction step. The mapping $T_{\delta R}$ encodes the current rotation update from the rotation sensor, while T_{intr} the transformation mapping from the camera frame to the rotation sensor within the device. The problem of this approach is the fact, that the pose of the inertial rotation relative to the camera T_{intr} is not necessarily known.

In this project T_{intr} was determined through heuristic testing. For the *SHIELD Tablet K1*, the right transformation was found and the tracking results could be improved. A short video showing the difference with and without the support of the rotation sensor can be found on *GitHub*[3]. An approach to determine T_{intr} via calibration is described in [KS10] or [Jor]. Implementing such an approach in the future development of the project is reasonable since the internal transformation T_{intr} is device dependent, as a test with a different device showed. Of course, not only the change in rotation but also the change in translation can be used to correct the pose for an appropriate inertial sensor

The appropriate integration of an inertial sensor to support the tracking algorithm was not the main part of this work. However, the improvements visible through the spartan use of the rotation sensor promise potential. Especially the integration of both, rotation and translation, should help to obtain a better result, and maybe even allow to reduce the tracker frequency to relieve system resources.

[3] https://github.com/fluckmic/transmem

4

Experiments

This chapter first presents the realization and outcome of an experiment investigating the perceived quality difference of the used markers. For the examination two different measures have been used. The confidence value obtained together with the transformation updates from the tracker (See Section 2.3.) as well as the introduced similarity measure ϕ_4 (See Section 2.2.). A second experiment, determining whether there is a positive effect of the averaging implemented in the marker model (See Section 3.3.) is discussed subsequently. This experiment makes also use of the similarity measure ϕ_4. Conclusively a third experiment which shall confirm the desired functionality of the implementation of *TransMem* and the marker model is described.

4.1 Preparations

To be able to conduct these experiments under manageable, repeatable conditions, a so-called *Experiment-Box*, as shown in Figure 4.1, was constructed. With this setup, the experiments were executed in a steady environment without suffering from external influences as for example changing light conditions. The dimensions were chosen to allow the box to be portable, while still allowing investigations from a distance, which is considered to represent a displacement from which the markers are observed in the later use of the application. With the construction of the *Experiment-Box* and the

The camera used for testing is a *Logitech HD Pro Web cam C920* with fixed internal parameters. A dimmable *PHOREX Ringleuchte 75W* at its brightest level served as the light source. All tests were executed on a system running a *64-bit Ubuntu 16.04 LT*S equipped with an *Intel Core i7-6700 CPU@ 3.40GHz x 8* and a *GeForce GTX 1050*.

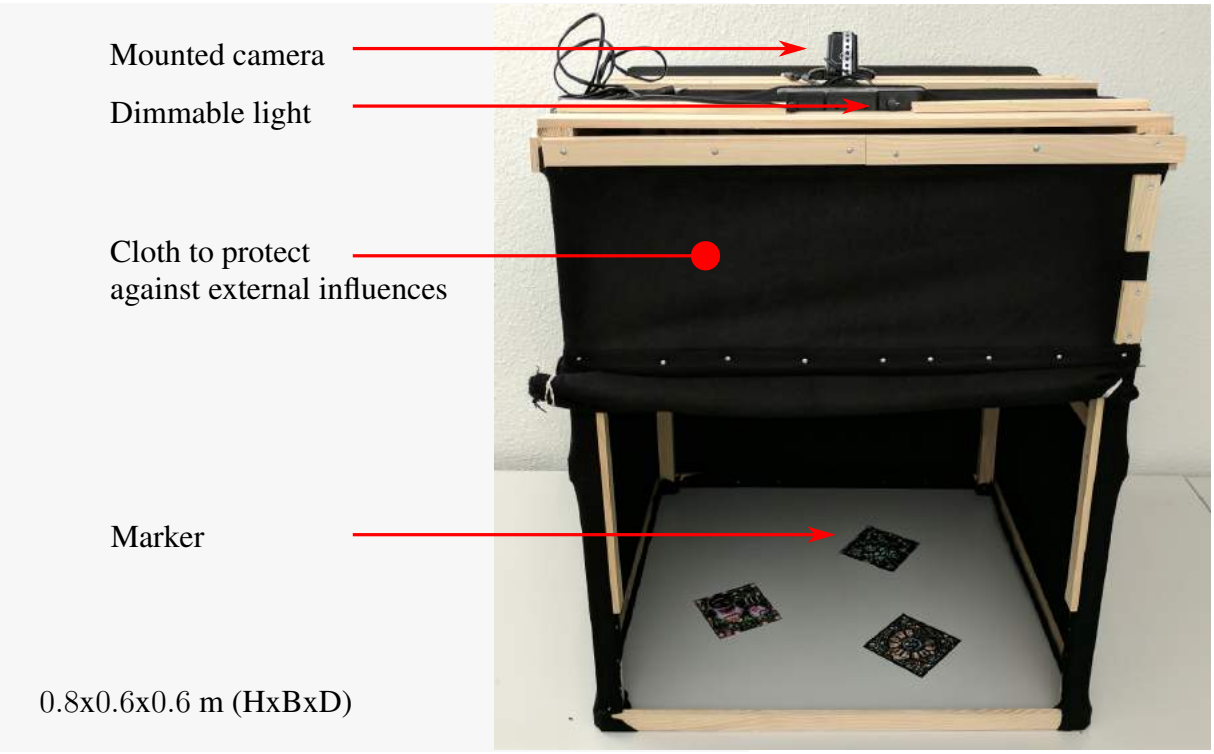

Figure 4.1: *An image of the Experiment-Box with its most important parts labeled.*

4.2 Quality difference of markers

4.2.1 Goal of the experiment

In the course of this project, three different markers were used. These are called *wcm* (world center marker), *ohm* (orange house marker) and *ahm* (ada house marker) for the remainder of this chapter. Figure 4.2 shows the three markers.

During the development, the impression arose that the tracker is not able to track all these markers with equal ease. More specifically, it seemed that the *wcm* is tracked more accurately than the other two markers. The goal of this experiment is the attempt to clarify this impression and to show how a tool like the *Experiment-Box* can be used to evaluate the quality difference of markers in general.

4.2.2 Procedure

In the *Experiment-Box* two lines indicating four directions separated by 90 degrees are tagged on the ground. Every marker is placed four times with its center congruent to the interception point of the two lines into the box, in each case, pointing in a different direction (*NE*, *SE*, *SW*, or *NW*). For every placement, 1800 position-updates, i.e. transformations, together with the corresponding confidence values received from the tracker are recorded. The camera observing a marker and delivering the images to the tracker is mounted at a fixed position. Gathering 1800

wcm (world center marker) *ohm* (orange house marker) *ahm* (ada house marker)

Figure 4.2: *The three marker wcm (l.), ohm (m.) and ahm (r.) used during development and experiments.*

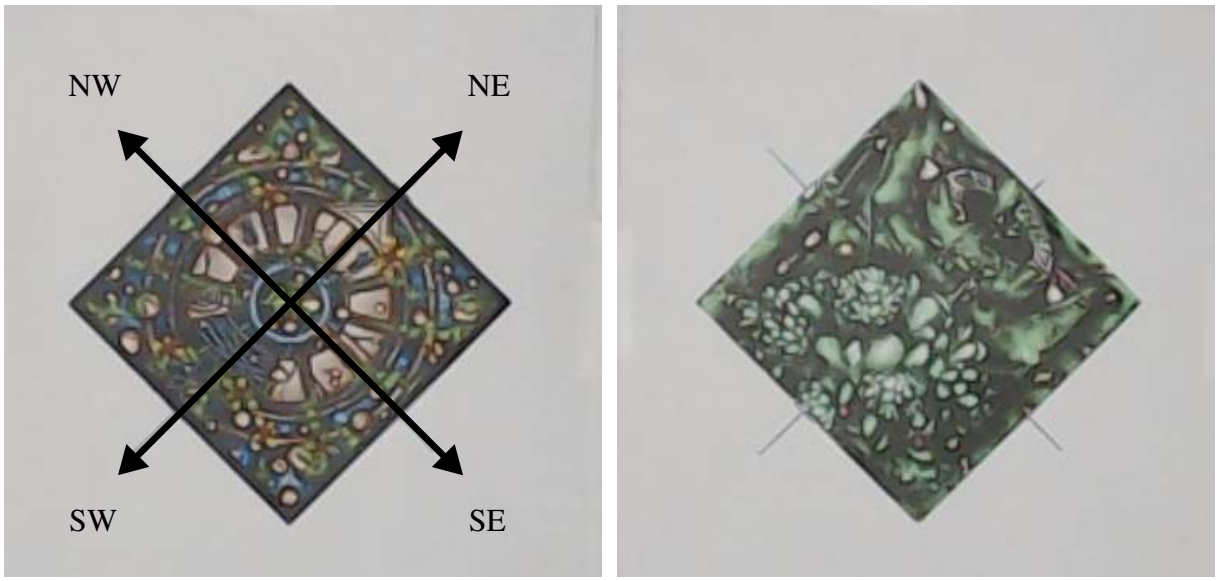

Figure 4.3: *The wcm pointing towards SW (l.) and the ahm pointing towards NE (r.).*

measurements takes about one minute what seems to be a reasonable duration yielding a comprehensive result with a manageable number of data points. The decision to place the markers in these orientations is made to also be able to investigate the rotation dependent behavior. As an example Figure 4.3 shows the *wcm* pointing towards *SW* on the left and the *ahm* pointing towards *NE* on the right.

The determination whether two markers are of equal quality is done based on two different measures. On the one hand, the confidence value is used to compare different markers. With the confidence value, $c \in [0, 1]$ the tracker quantifies for each position update of a marker the quality of the transformation. It is calculated as the ration of the number of detected *BRISK* features and the total number of *BRISK* features (See Section 2.3.) of a certain marker. The comparison of the markers based on the measured confidence value is presented in the first part of the following results section.

On the other hand is the similarity measurement ϕ_4 introduced in Section 2.2 used. The idea was

to establish a measure, which allows to make statements about the quality of different markers without using the confidence obtained from the tracker. However, this should be seen as an idea how one could realize such a comparison and not as a fully developed procedure. Theoretically, if a static marker is observed with a mounted camera, all transformations received from the tracker should be perfectly equal. Obviously, this is not possible, since already the tracking process suffers from some inaccuracy. The transformations received vary to some extent where this variation can be used as a quality measure of a marker.

$|\phi_4|$-*Histogram*. One possible way to represent this variation is by calculating a so called $|\phi_4|$-*Histogram*: The transformations measured over a given time for a certain marker placed at a static position are pairwise compared. As a comparator, the similarity measure ϕ_4 is used. All the lengths $|\phi_4|$ of the vectors obtained through the pair wise comparison are then displayed in a $|\phi_4|$-*Histogram*. The quality of different markers can, therefore be compared by comparing their corresponding $|\phi_4|$-*Histogram* as presented in the second part of the following results section.

4.2.3 Results

Comparison based on the confidence c

Intra marker comparison. Figure 4.4, Figure 4.5 and Figure 4.6, show the measured confidence values for the three markers *wcm*, *ohm* and *ahm*, respectively, pointing in the four different directions. The plots confirm the rotation invariance of *BRISK* features used for the calculation of c, since no major difference between the values for different orientations of a marker can be observed. A further observation which can be made, is the fact that the confidence values of the *ohm* are clearly less high than the values of the *wcm* and the *ahm*, indicating a quality difference.

Inter marker comparison. The confidence values of the same marker, pointing in different directions, are combined. This can be done since the confidence values not depend on the rotation, as seen in the intra marker comparison discussed above. The Figure 4.7 shows the combined confidence values for each of the three markers placed at the same position. The distributions of the confidence for the *wcm* and the *ahm* are almost identical. The tracker recognizes these two markers equal well. By contrast, the majority of the confidence values of the *ohm* lie below the values of the other two markers. This supports the impression one got during the development. Relating to the confidence, the *ohm* is less well recognized than the *wcm*. However, one cannot make this statement for the *ahm*.

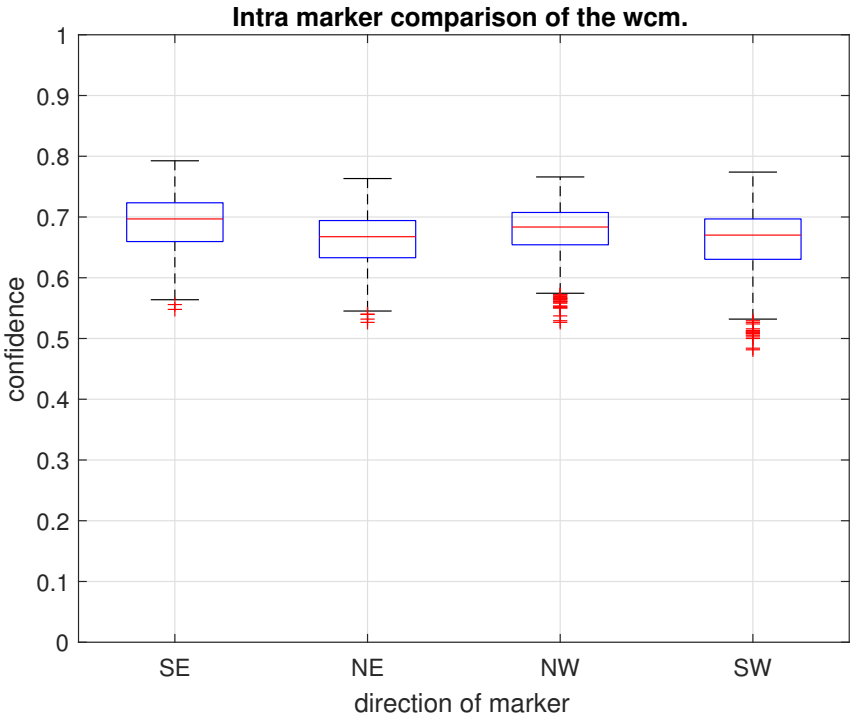

Figure 4.4: *Intra marker comparison of the wcm. There is no clear difference between the distribution of the confidence values for the different orientations noticeable.*

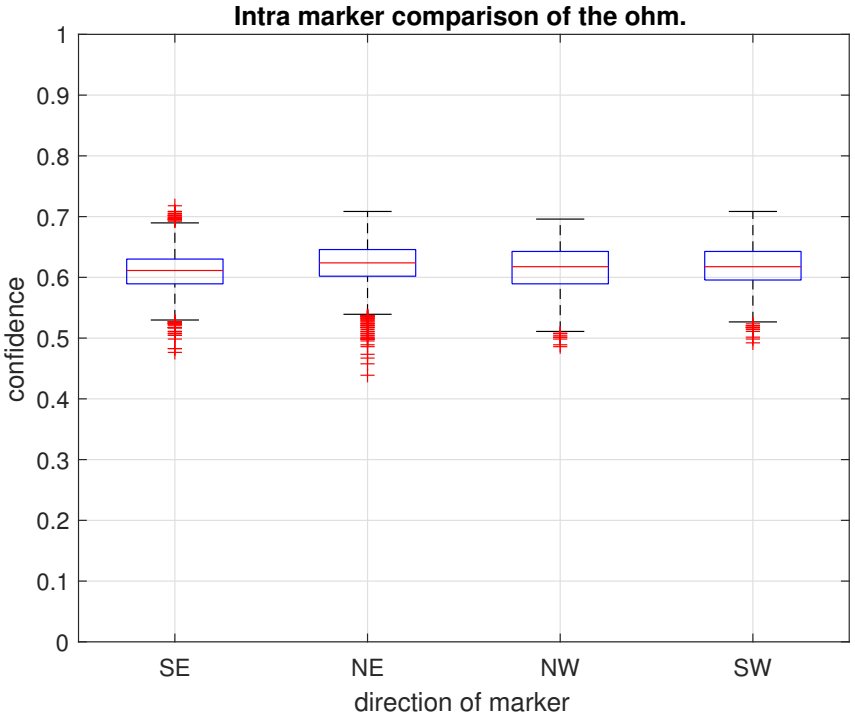

Figure 4.5: *Intra marker comparison of the ohm. There is no clear difference between the distribution of the confidence values for the different orientations noticeable.*

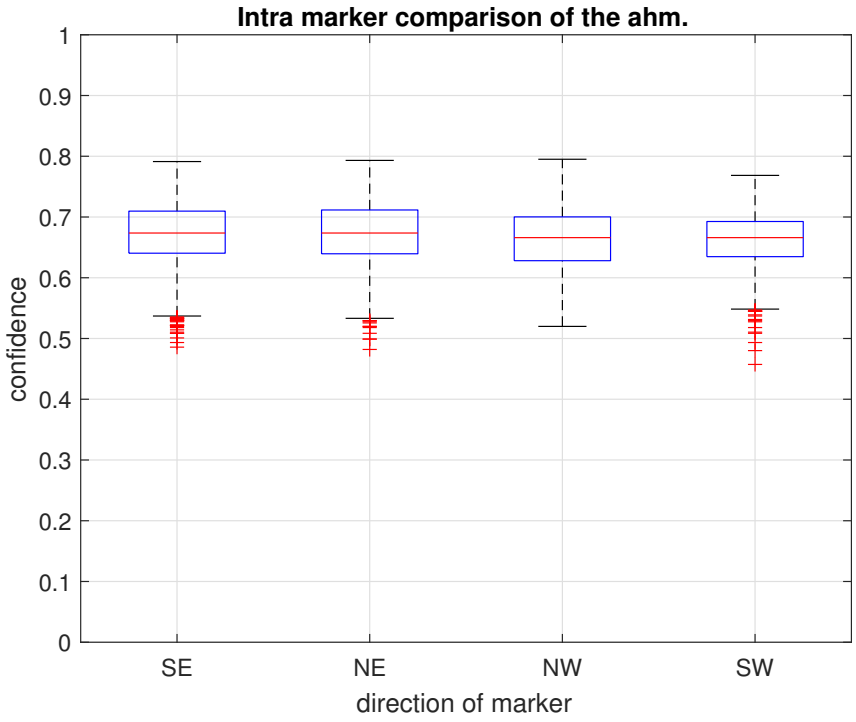

Figure 4.6: *Intra marker comparison of the ahm. There is no clear difference between the distribution of the confidence values for the different orientations noticeable.*

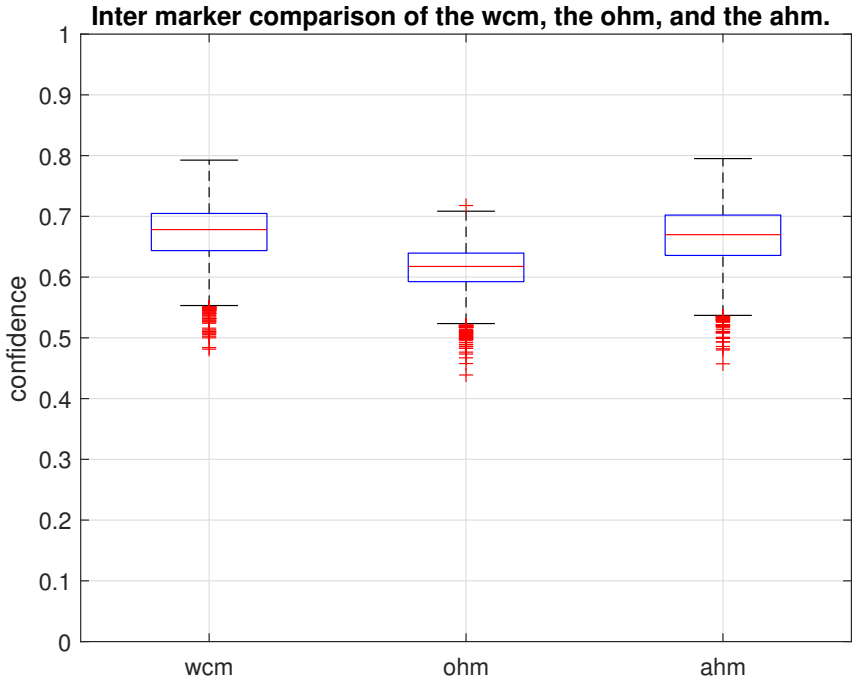

Figure 4.7: *Inter marker comparison of the three markers wcm, ohm and ahm. The ohm delivers clearly worse confidence values although all markers where tracked under equal conditions.*

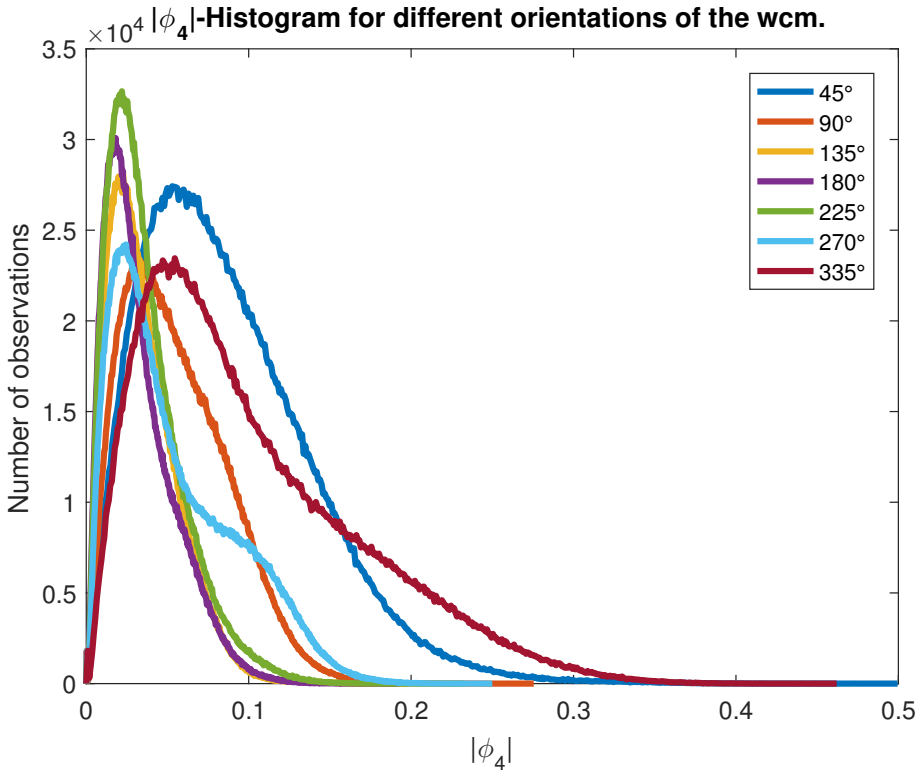

Figure 4.8: *$|\phi_4|$-Histogram for different orientations of the wcm. The distributions depend on the orientation of the marker since the ϕ_4 is not rotation invariant.*

Comparison based on the similarity measure ϕ_4

During the experiments a flaw of the similarity measurement ϕ_4 became obvious. It was not possible to observe a visible difference in the performance of a marker, placed at the same position pointing in different directions, by looking at the augmented object representing the currently obtained transformations. Figure 4.8 shows the $|\phi_4|$-*Histograms* for the *wcm* pointing in different directions labeled with the rotation angle, part of the correct transformation between camera and marker. According to these histograms is the tracker able to deliver the mappings between camera and marker with lesser scatter the larger the rotation angle of the correct mapping is. But this statement is incompatible with the observations made.

Rotation dependence of ϕ_4. If two nearly identical transformations are compared, which both encode a mapping with a small rotation part close to 0 degrees, then the difference between their rotation axes is less meaningful as in the case where the mapping consists of rotation part close to 180 degrees. Figure 4.9 illustrates this fact. Imagine a cuboid K which is green at the top and blue at the bottom. Let K have two perpendicular rotation axes r_1 and r_2. For a small rotation angle of 5 degrees, the transformation of K either along r_1 or r_2 results in a nearly similar object. However, the length between the two rotation axes amounts to $\sqrt{2}$, a false indication for unequal transformations under ϕ_4. In the case of an angle of 180 degrees, a rotation around r_1 yields a completely different object than a rotation around r_2, in this case, the distance of $\sqrt{2}$ indicating two different transformations under ϕ_4 is correct. That is, the similarity measure ϕ_4 is not suitable to compare nearly identical transformations which encode both a small rotation

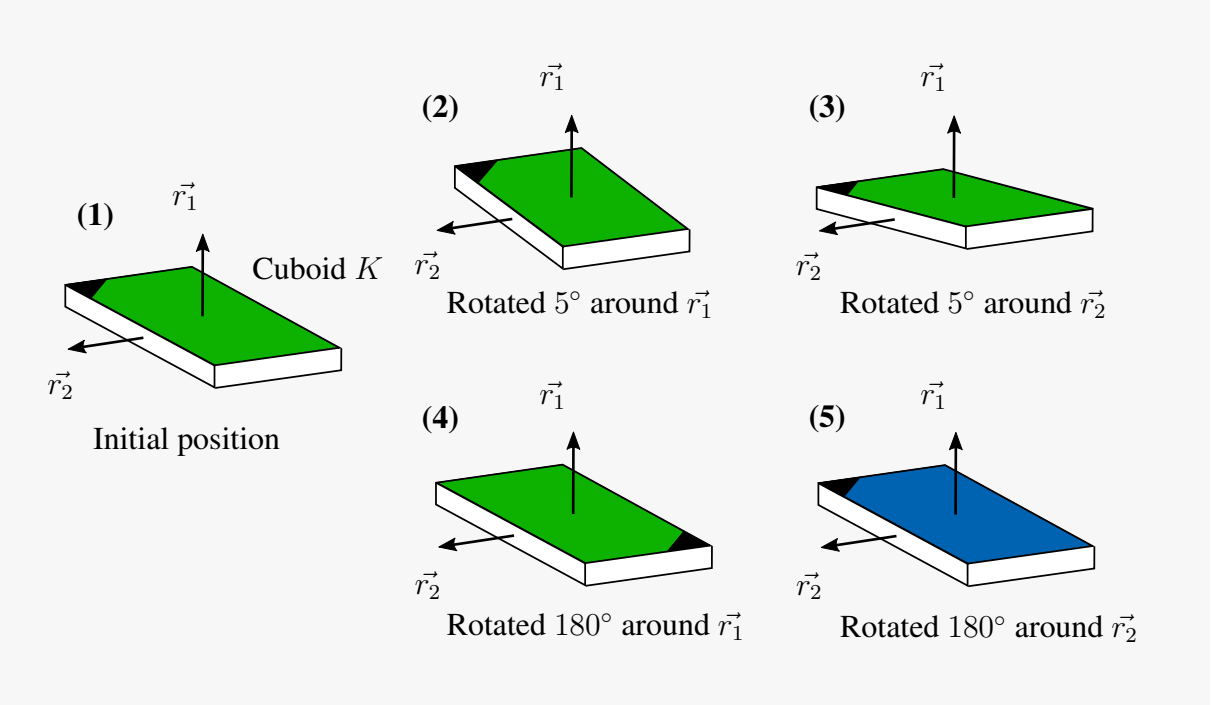

Figure 4.9: *The cuboid K in its initial position (1), after a rotation of $5°$ around r_1 (2) and after a rotation of $5°$ around r_2 (3). Although the rotation axes differ by a length of $\sqrt{2}$ the position of the cuboid only differs a little after applying either the transformation around r_1 or the transformation around r_2. By contrast, (4) shows the cuboid after a rotation of $180°$ around r_1 and (5) after a rotation of $180°$ around r_2. In this case the two final positions differ heavily, which is conform with ϕ_4.*

angle, since in this case, the length between the rotation axes can heavily vary without stating something meaningful about the similarity of the two transformations. Based on these considerations supported by the Figure 4.8, a comparison of the different markers with a ϕ_4-*Histogram* is expected to be most meaningful when pointing towards somewhere between northeast (NE) and north-west (NW). For these orientations the mapping between marker and camera consists of a large rotation angle.

Figure 4.10 visualizes the $|\phi_4|$-*Histogram* for the three markers placed at the same position in the *Experiment-Box* pointing towards north-east (NE). For the *wcm*, about 90% of the compared transformations yield a value for $|\phi_4|$ which is smaller than 0.05. This number lies at around 61% for the *ohm* and at about 57% for the *ahm*. If one considers the numbers of comparisons which yield a value for $|\phi_4|$ which is smaller than 0.1 a result of 99% for the *wcm*, 94% for the *ohm* and 89% for *ahm* is received. These numbers support the impression mentioned at the beginning of the chapter. The tracker is able to deliver transformations for the world center marker with less scatter than for the orange house marker and the ada house marker. Comparing the marker quality with the same approach for recordings obtained while the marker oriented towards north-west resulted in a similar finding.

This supports the impression the author got during the development of the project; the tracker is able to track the *wcm* with more ease than the other two markers. However, as already

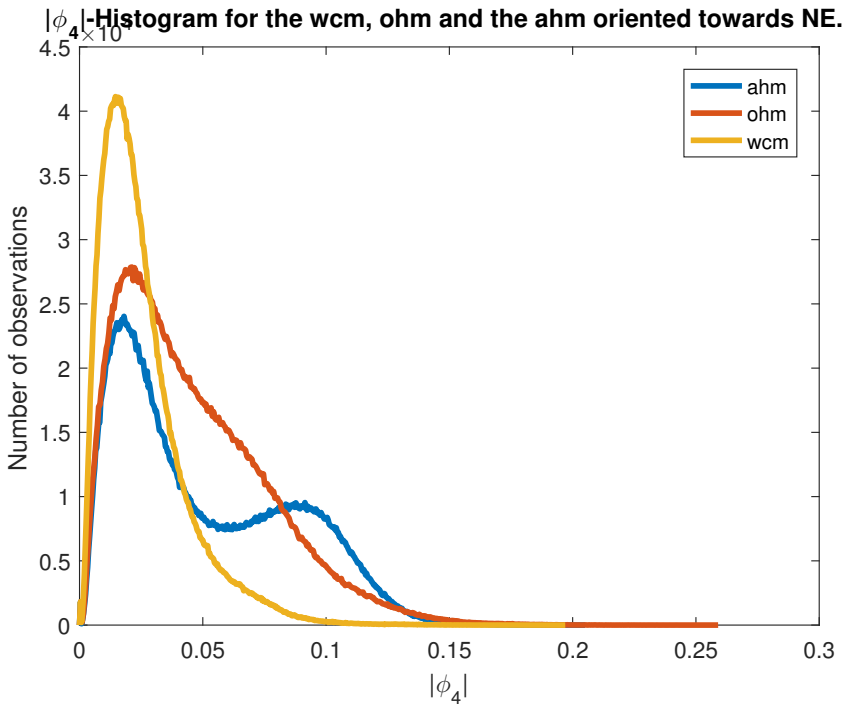

Figure 4.10: *$|\phi_4|$-Histogram for the three markers wcm, ohm and ahm placed at the identical position pointing towards northeast. The distribution of wcm decreases earlier to zero than the distribution of the other two markers, indicating a better quality of the world center marker.*

mentioned, this approach should just be viewed as an idea to compare the quality of the markers and not as a mature methodology. There are more in-depth investigations and the executions of experiments necessary to obtain a meaningful impression about how valid this approach is.

The author assumes that the reasons for the observed quality differences are related to the number and the quality of the *BRISK* features. Further investigations considering this features are advisable since the quality of the markers has a great influence on the performance of the vision system. With the construction of the *Experiment-Box* and the presented procedure, the foundation for such investigations is given.

4.3 Reducing scatter through averaging

4.3.1 Goal of the experiment

The marker model described in Chapter 3.3 implements an averaging of the transformation $T^{O}_{cam \leftarrow m_0}$, i.e. the position of the *wcm* relative to the camera. The goal of this experiment is to analyze if the averaging has a positive effect, i.e. if the scattering of the transformations reduces in the case the feature is used.

4.3.2 Procedure

The three marker *wcm*, *ohm* and *ahm* are placed inside the *Experiment-Box* in different arrangements. The samples were chosen to try to cover a good cross section of possible arrangements. For example, all markers close together in the middle of the camera or all markers placed at a distinct border of the *Experiment-Box*. For each positioning, 1800 transformation updates for $T^{O}_{cam \leftarrow m_0}$ are recorded with averaging and without averaging, respectively. Whenever averaging was used, the threshold *TH-TRANS-EQ* was set to a length of 0.4 what seems reasonable considering the findings of the previous experiment (See Section 4.2.3.). In the case without averaging the *TH-TRANS-EQ* is inherently a vector of length zero. As for the second part of the previous experiment $|\phi_4|$-*Histogram*s are used for investigation. If there is a positive effect, this should be visible in the comparison of the $|\phi_4|$-*Histogram*s obtained with and without averaging for the same arrangement of the markers.

4.3.3 Results

A positive effect could not be recognized in any experiment. The $|\phi_4|$-*Histogram*s resembled each other independent of the averaging. Even if all markers were placed in the middle of the scene, where one expects the best tracking result from all markers, no effect is noticeable. The Figure 4.11 shows the two $|\phi_4|$-*Histogram*s for the just mentioned arrangements of the three markers, which deviate from each other just marginally. This behavior can be explained by the fact that the quality of the transformations obtained from the *ohm* and the *ahm* is worse than the quality of the *wcm* as observed in the previous section. Since these transformations from the *ohm* and the *ahm* are used to calculate the alternative transformations used in the averaging, it is not surprising that the averaging causes no decline of scattering.

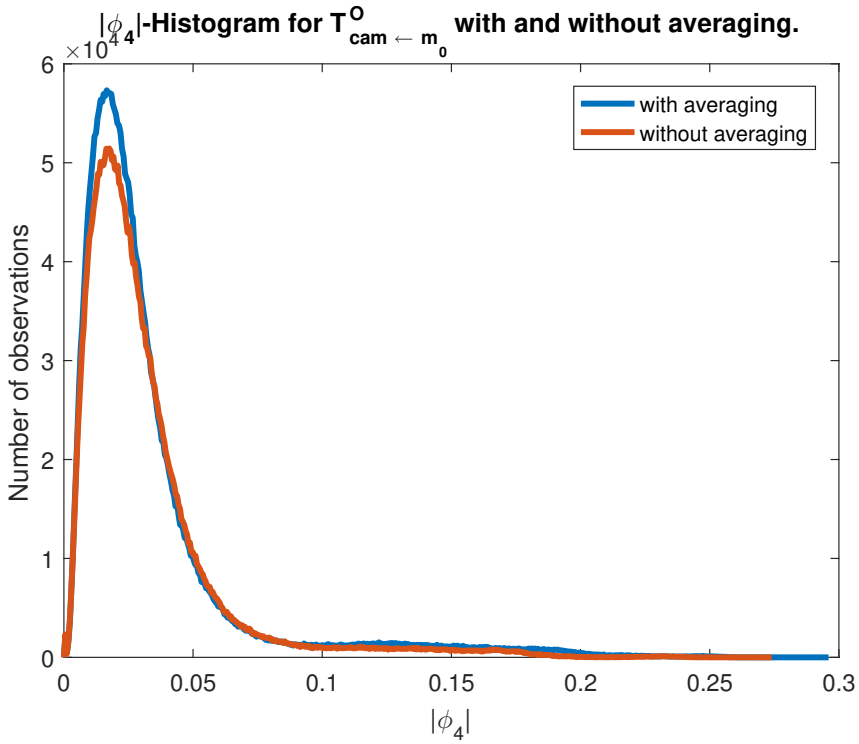

Figure 4.11: *$|\phi_4|$-Histogram for a identical arrangement of the three markers with and without averaging. The nearly identical distributions indicates that there is no positive effect of the averaging.*

4.4 TransMem at work

4.4.1 Goal of the experiment

The goal of this experiment is to show that the presented implementation of the marker model, (See Section 3.3.) which itself uses the implementation of *TransMem* (See Section 3.2.), is able to maintain the position of a world center marker as long as at least one of multiple marker is visible.

4.4.2 Procedure

The three markers, *wcm*, *ohm* and *ahm* are placed in the *Experiment-Box*. The *wcm* is defined as the world center marker. At the position of each marker, the scene is augmented with a virtual object. Additionally, a red cube rotating around the *wcm* is inserted to represent a protagonist walking through the scene. The position of the cube is described relative to the *wcm*. In the course of the experiment different combinations of the marker are hidden such in a way that always one marker is visible to the tracker. If the marker model works properly the rotating cube remains visible independent of the combination chosen.

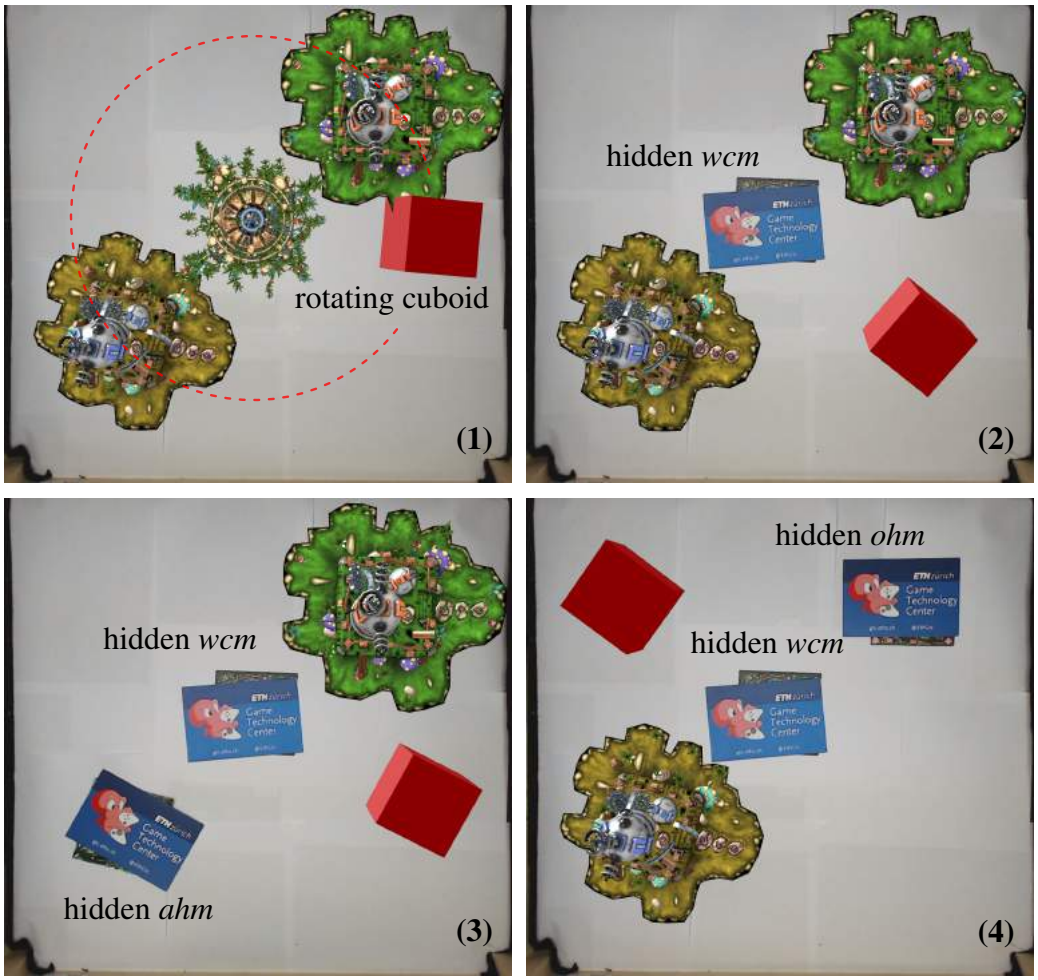

Figure 4.12: *At the position of each marker the scene is augmented with a virtual object. Additionally a red cuboid rotating around the wcm is inserted, representing a protagonist walking through the scene. Image (1) shows the scene without a hidden marker. The red cuboid can be drawn as long as at least one marker is visible as seen in (2), (3) and (4). This because of the marker model, which is together with TransMem able to maintain the position of the wcm as long as at least one arbitrary marker is visible.*

4.4.3 Result

The marker model together with *TransMem* is able to maintain the position of a world center marker as long at least one of multiple markers is visible. Figure 4.12 confirms the desired functionality. Even if the *wcm* is hidden in (2), (3) and (4) can the red cuboid be drawn without requiring a change of the scene graph.

5

Conclusion

This chapter provides a conclusion of the work done in the course of this project, discusses its limitations together with solution approaches and provides suggestions how the work can be continued.

Conclusion. The aim of this project has been the improvement of the vision system of *Thymio Programming Adventure*. The vision system should, in particular, be improved, such that it handles multiple markers which may get occluded, in a way that the virtual scene can be represented with a static scene graph. This objective has been achieved due to the presented implementation of a data structure able to store and retrieve transformations. Integrated into an also presented marker model can the data structure be used to maintain the position of a special world center marker as long as at least one of multiple markers is visible. This allows to express all virtual objects relative to this special marker and to draw them without having to reorder the scene graph regardless what strict subset of the markers is occluded. Another objective pursued by this project has been the investigation if there is a quality difference between the three used marker. To be able to conduct these investigations under controllable circumstances an *Experiment-Box* was constructed. A quality difference of the three used markers could be observed by comparing them based on the confidence value obtained from the tracking algorithm as well as by considering the introduced $|\phi_4|$-*Histogram*.

Limitations. The behavior of the presented marker model depends on five static parameters whereby two relate to the confidence value obtained from the tracker. Having static parameters is, for this reason, disadvantageous since no adaption to the changing environment happens. Namely, influences the light condition and the distance between a marker and the tracking the quality of the obtained transformations and therefore its corresponding confidence value. An option to decrease this limitation would be to adjust the parameters of the marker model dynamically. The distance between the observer and scene extracted from the transformations or the current lighting situation gathered from the illumination sensor of the device could serve as influence quantities.

Whenever a marker is detected together with the world center maker the fixed transformation between these two markers is updated. It may occur the situation where a perturbed fixed transformation is stored. This happens when the confidence of a tracking result remains enough high to be considered a good transformation, even if the result is already disturbed through an ongoing occlusion. This leads then to a wrong calculation of the position of the now hidden world center marker based on the disturbed fixed transformation. A thinkable approach to solve this problem is to refine the criterion of a fixed link update in such a way that an update just happens after a number of consecutive observations of both markers whereby the corresponding confidence values must not become smaller within these observations.

The presented implementation was developed and tested mainly on a desktop computer where the program ran without noticeable performance issues. However, when running on a tablet the performance decreased notably. Since this works is meant to be integrated into a software running on a tablet further investigations to determine the reason for this performance issue is essential. Reasonable would, for example, be to determine what fraction of the resources is consumed by the presented implementation. This allows to reason about a possible improvement of this work.

Future work. Besides the above-mentioned solution approaches for the limitations, there are further objectives thinkable to carry on this project. For example the implementation of a more sophisticated marker model which uses a machine learning approach or a probabilistic model for decision making. Especially meaningful is more over the integration of the *Thymio robot* into the marker model, since the vehicle is the protagonist of the *Thymio Programming Adventure*.

Bibliography

[BCL15] Mark Billinghurst, Adrian Clark, and Gun Lee. A survey of augmented reality. *Found. Trends Hum.-Comput. Interact.*, 8(2-3):73–272, March 2015.

[BH01] K. Briechle and U. D. Hanebeck. Template matching using fast normalized cross correlation. In D. P. Casasent and T.-H. Chao, editors, *Optical Pattern Recognition XII*, volume 4387, pages 95–102, March 2001.

[Boo] Boost graph library (bgl). http://www.boost.org/doc/libs/1_65_0/libs/graph/doc/.

[Foo13] Tully Foote. tf: The transform library. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pages 1–6, April 2013.

[HTDL13] Richard Hartley, Jochen Trumpf, Yuchao Dai, and Hongdong Li. Rotation averaging. *International Journal of Computer Vision*, 103(3):267–305, Jul 2013.

[Huy09] Du Q. Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, Oct 2009.

[Jor] Jorge Lobo. Inervis toolbox for matlab. http://home.deec.uc.pt/~jlobo/InerVis_WebIndex/InerVis_Toolbox.html.

[KS10] Jonathan Kelly and Gaurav S. Sukhatme. Fast relative pose calibration for visual and inertial sensors. In *Experimental Robotics*, pages 515–524. Springer Verlag, 2010.

[KVR+14] Kriti Kumar, Ashley Varghese, Pavan K. Reddy, N. Narendra, Prashanth Swamy, M. Girish Chandra, and P. Balamuralidhar. An improved tracking using IMU and vision fusion for mobile augmented reality applications. *CoRR*, abs/1411.2335, 2014.

[LCS11] S. Leutenegger, M. Chli, and R. Y. Siegwart. Brisk: Binary robust invariant scal-

able keypoints. In *2011 International Conference on Computer Vision*, pages 2548–2555, Nov 2011.

[MCCO07] F. Landis Markley, Yang Cheng, John L. Crassidis, and Yaakov Oshman. Averaging quaternions. *Journal of Guidance, Control, and Dynamics*, 30(4):1193–1196, 2007.

[QCG+09] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on*, Open-Source Software, 2009.

[Thya] Thymio programming adventure. http://www.mobsya.org/ext-media/Thymio_adventure%20-%20small.pdf.

[Thyb] Programming with thymio and aseba. https://www.thymio.org/home-en:home.

[yB00] Jean yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.