UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Combining Pose Tracking with Sketch Recognition for Augmented Drawings

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von

Jan Heitger

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Dipl.-Inform. Dominik Grüntjens (Institut für Computervisualistik, AG Computergraphik)

Koblenz, im September 2011

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

....

(Ort, Datum) (Unterschrift)

Abstract

Augmented Reality bedeutet eine reale Umgebung mit, meistens grafischen, virtuellen Inhalten zu erweitern. Oft sind dabei die virtuellen Inhalte der Szene jedoch nur ein Overlay und interagieren nicht mit den realen Bestandteilen der Szene. Daraus ergibt sich ein Authentizitätsproblem für Augmented Reatliy Anwendungen. Diese Arbeit betrachtet Augmented Reality in einer speziellen Umgebung, mit deren Hilfe eine authentischere Darstellung möglich ist.

Ziel dieser Arbeit war die Erstellung eines Systems, das Zeichnungen durch Techniken der Augmented Reality mit virtuellen Inhalten erweitert. Durch das Anlegen einer Repräsentation soll es der Anwendung dabei möglich sein die virtuellen Szeneelemente mit der Zeichnung interagieren zu lassen. Dazu wurden verschiedene Methoden aus den Bereichen des Pose Tracking und der Sketch Recognition disktutiert und für die Implementierung in einem prototypischen System ausgewählt. Als Zielhardware fungiert ein Android Smartphone.

Kontext der Zeichnungen ist eine Dungeon Karte, wie sie in Rollenspielen vorkommt. Die virtuellen Inhalte nehmen dabei die Form von Bewohnern des Dungeons an, welche von einer Agentensimulation verwaltet werden. Die Agentensimulation ist Gegenstand einer eigenen Diplomarbeit [18]. Für das Pose Tracking wurde ARToolkitPlus eingesetzt, ein optisches Tracking System, das auf Basis von Markern arbeitet. Die Sketch Recognition ist dafür zuständig die Inhalte der Zeichnung zu erkennen und zu interpretieren. Dafür wurde ein eigener Ansatz implementiert der Techniken aus verschiedenen Sketch Recognition Systemen kombiniert. Die Evaluation konzentriert sich auf die technischen Aspekte des Systems, die für eine authentische Erweiterung der Zeichnung mit virtuellen Inhalten wichtig sind.

Contents

1 Introduction

1.1 Motivation

Recent advances in hardware development have led to an increased availability of augmented reality capable platforms. Modern smartphones are equipped with dual-core processors, ram of 512MB or more, high resolution cameras and a variety of sensors. In addition increasingly fast network speeds make large databases available for mobile online use.

Consequently augmented reality applications are becoming more common. Augmented Reality games for example are quickly becoming an established genre. Sony Entertainments Eye of Judgment has been an early big production title of 2007 that featured AR content in the form of a card game. Later, Invizimals from Novarama [36], see figure 1, in 2009 used the camera attachment of the Playstation Portable to display virtual creatures the player could control. More recently Nintendo released their new handheld gaming system, the Nintendo 3DS, which includes cards that can be used to play various AR games. In addition there are numerous AR games available for smartphones, making use of the increasing hardware capabilities of the devices.

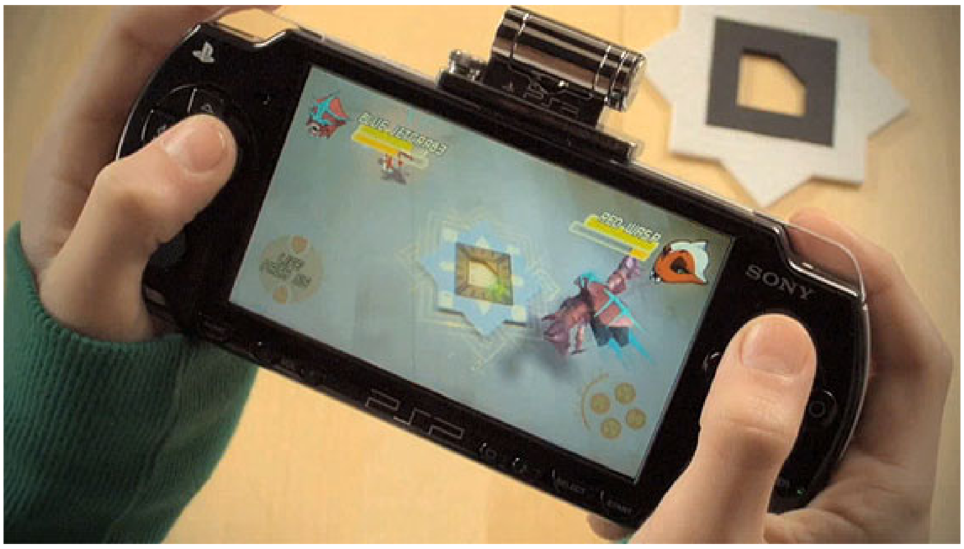

Figure 1: Invizimals [36]

At the same time, AR content is also being utilized in new ways. So called AR-Browser applications, like Wikitude, Junaio or Layar, use position information to allow users to view and create contextual information at specific locations in the form of superimposed images and text. Overlays like this serve many different purposes from navigation to communication and entertainment. A recent example for an AR translation application is Questvisuals WordLens [49], which uses the built-in camera to locate text and presents a translated version to the user looking at the or "through" the device. See figure 2 for an example.

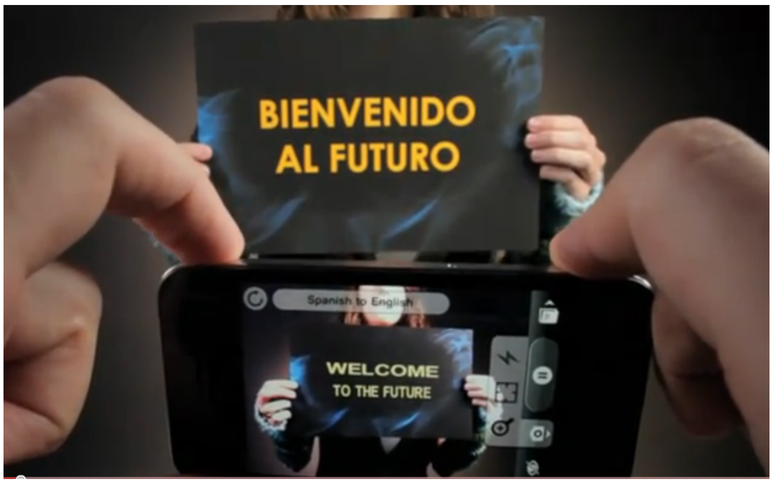

Figure 2: WordLens [49]

All these applications share the basic techniques for Augmented Reality, superimposing virtual content, mostly images, over real world scenery. Position information gained from fiducial markers or other sources, like gps or other sensors, is used to define the position for the virtual objects. In most cases the positioning is sufficiently accurate. Lacking further information about the real world scenery though, the virtual content is limited in its potential for interaction. The virtual creatures from Invizimals can only walk on the virtual plane defined by the fiducial marker, they cannot recognize walls or the end of a table. This creates a general credibility problem for AR content.

To solve this problem an application would need more information about its real world environment. If there was a way to incorporate the geometry information (relative position, size) of the table, it could be supplied to the application controlling the virtual creature walking on top of it, causing it to stop, when it reaches the edge of the table, which would result in more believable virtual content.

Tangible AR applications alleviate some of the problems, as they make parts of the scene available for user interaction, thus linking the virtual content to the real world counterpart. For example a wand-like controller, made available to the application through fiducial markers on it, can't be moved through a wall.

The basic problem still persists though. If the virtual content is supposed to act according to its surroundings, geometry information is needed. This information can be created beforehand or dynamically at run-time and both cases pose problems. If the information is created in advance, it will only reflect the environment at a distinct time. Changes in the environment are still unsolved. Additionally the creation of the needed data can be complex and time consuming.

Creating accurate geometry information at real-time from an optical input is an ongoing topic in computer vision research. Simultaneous Localization and Mapping systems are often employed in robotics research and are capable of providing accurate geometry information. However they come along with high demands on hardware resources and are not yet suited for augmented reality applications, which have to manage other tasks like rendering and often run on less capable hardware (e.g. Smartphones).

1.2 Goal

The desire to explore ways for a stronger interaction between the real and virtual scene elements in an AR system led to a special case that simplifies the problem. Instead of using complex 3D environments as scene context, a 2D drawing is used as environment. A simpler environment allows for a faster recognition process and the construction of a data representation of the surroundings.

In this way, it is possible to let the superimposed virtual scene elements have knowledge about their less complex environment and to make them react accordingly. For example virtual agents walking on the plan of a building, drawn on a sheet of paper, would behave in a more believable way: using doors to pass from one room to the other, walking along the drawn corridors and not through walls, etc. By doing so, they heighten the authenticity of the virtual scene elements. The faster recognition process would also allow for dynamic changes in the environment. New rooms could be added to the building plan by drawing them next to the existing ones and new information could be added to existing rooms in the form of annotations.

While the simplified environment does not equal a real world environment, it allows for the exploration of new ways in which real and virtual scene elements can interact.

The goal of this work is to analyze ways the aforementioned system can be realized, how it needs to be structured, what components can be used and in which ways they interact. A prototype system is to be created and evaluated.

1.3 Structure

The work uses the following structure:

Section 2 illustrates the concept of the application and gives a summary of the system use.

Section 3 details the basics necessary for this work and gives a short overview of pose tracking and sketch recognition.

Section 4 discusses the components used for the different tasks in this work and states the requirements of the system.

Section 5 details the implementation of the program, starting with a basic system overview followed by details about the pose tracking and sketch recognition parts.

Section 6 evaluates the system according to the requirements, focusing on pose tracking and sketch recognition. Special consideration is given to the performance aspect to see if real-time sketch recognition is feasible on the given hardware.

Section 7 summarizes the work and gives a conclusion as well as an outlook for further improvements and research ideas.

2 Concept

As mentioned in the previous section the basic idea of the application is to enrich a real world environment, in the form of a user created drawing, with virtual scene elements. These elements interact with their real world environment, in order to heighten the authenticity of the augmentation.

For this application the environment takes the form of a floor plan drawing, or more precisely a dungeon drawing. Dungeons are an environment type often found in computer or traditional roleplaying games. They provide many interaction options for the virtual scene elements and are usually presented in the form of drawings. Figure 3 shows an example of a dungeon drawing and section 3.5 provides more detail about the dungeon context and why it was chosen for this work.

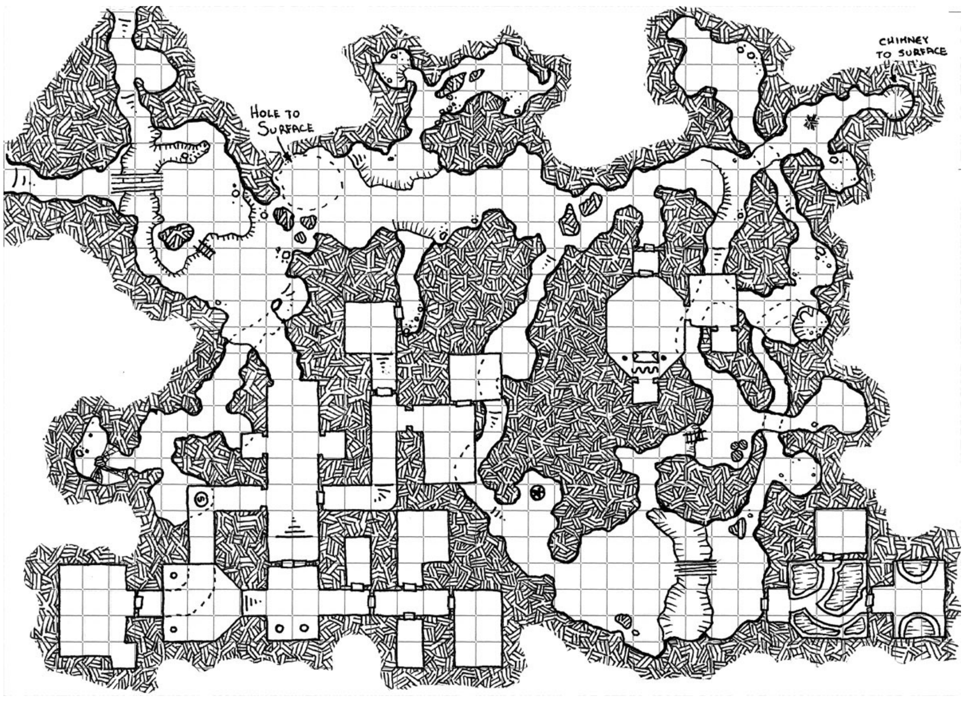

Figure 3: Dungeon drawing [17]

Inside this environment, the virtual scene elements take the form of dungeon inhabitants, that is creatures akin to those found in a usual roleplaying context like orcs and goblins. Their interactions with the environment and with each other are governed by an agent simulation that is part of the application. The agent simulation is the focus of another diploma thesis and more information about it can be found in [18].

To show the virtual elements to the user an augmented reality setup that includes pose tracking is used. This enables the system to render the virtual scene elements on top of a camera image, in such a way that they appear to be part of the user-created drawing. The pose tracking part of the system assures that the virtual elements are correctly rendered, even if the camera is moved. The user perceives all this by means of a smartphone that acts as a "magic lens", through which the mixed reality environment of camera image and virtual scene elements can be seen.

The application runs on a smartphone, though other options were explored in the component discussion chapter as well. Figure 4 shows a concept image of the application setup that was created during the collaborative planning phase with AĖiting [18].

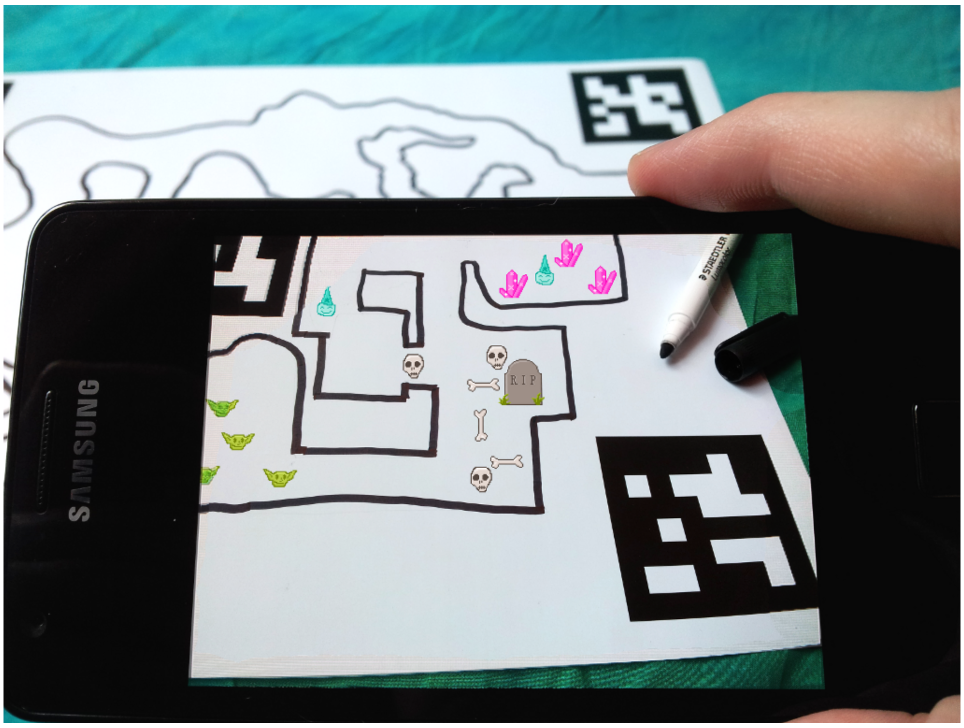

Figure 4: Concept image showing the setup and agents in the dungeon [18]

The drawing is interpreted by the application by employing a sketch recognition process. This technique allows the system to create a virtual representation of the user drawing, which forms the basis for the agent simulation.

Figure 5 illustrates an example scenario for the final system:

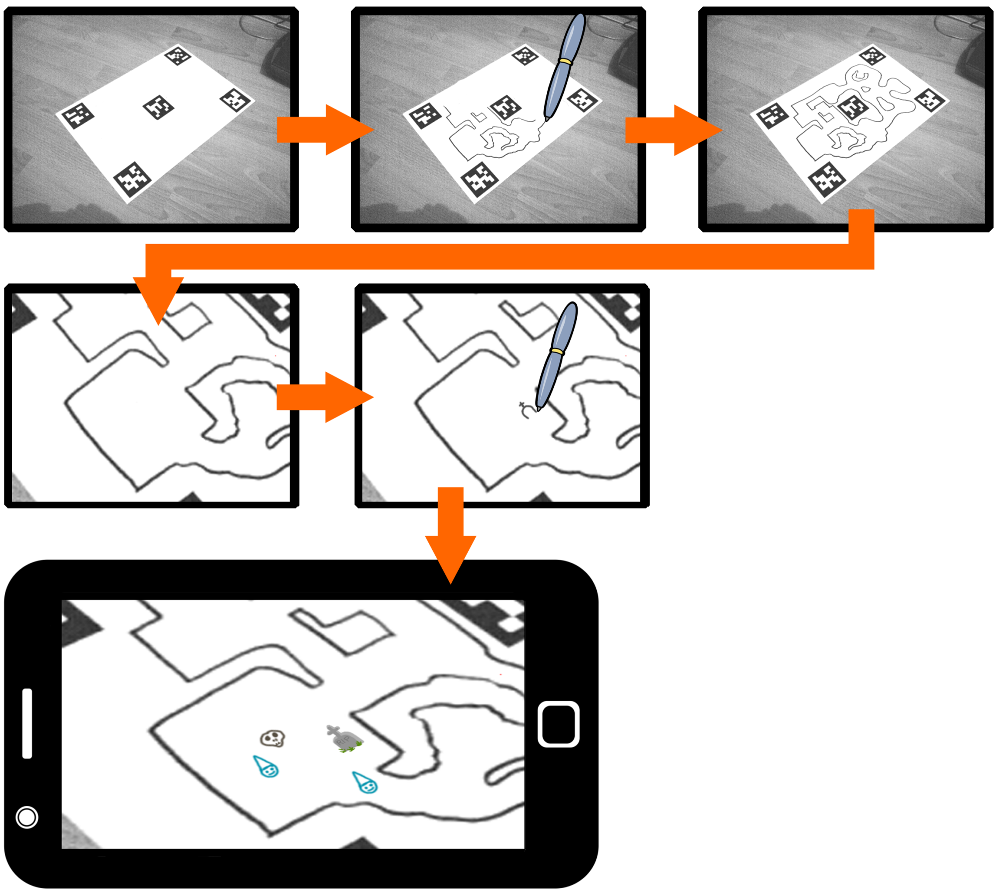

Figure 5

The user draws the dungeon on a sheet of paper. He can then add additional elements to the drawing to influence the simulation. Afterwards he uses the smartphone to watch the creatures inside the dungeon. Following that, new walls or additional symbols that represent interaction sources for the dungeon inhabitants could be added.

3 Basics

This section provides an overview of the basic principles relevant to this work and gives an example of the notations used throughout the document. It explains the techniques used for the augmented reality setup of this work and provides details about the subject of sketch recognition, which is used to process the user-created drawing.

3.1 Augmented reality

Augmented Reality is a part of computer graphics, which aims to combine real world scenes with computer generated information (often visual information), thus creating a mixed reality environment. Augmented Reality is part of the Mixed Reality Continuum formulated by P. Milgram and F. Kishino [34], which encompasses the spectrum from the real world to virtual reality.

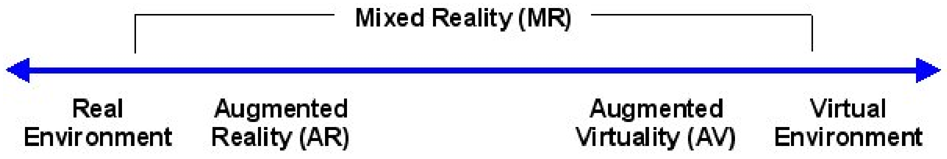

Figure 6: Mixed Reality Continuum [34]

Between these two extremes Mixed Reality defines a space in which virtual scenes are combined with real world information (Augmented Virtuality) or in which real world scenes are enriched by virtual information (Augmented Reality). Augmented Reality is already used in a multitude of applications, ranging from sports tv to construction and medicine, where information is presented as overlays over real world objects to guide the user. The added information is presented in different forms, e.g. as virtual object or text. Figure 7 shows an example of an augmented reality application running on a smartphone

To ensure the proper positioning of the contextual information a tracking system is used. These systems operate with different sensors to determine relevant positions, e.g. mechanical tracking, ultrasound, GPS or optical tracking.

3.2 Projection models

Projection, in this context, describes the mapping of a point in three dimensional space to a point on a two dimensional plane. For an AR application

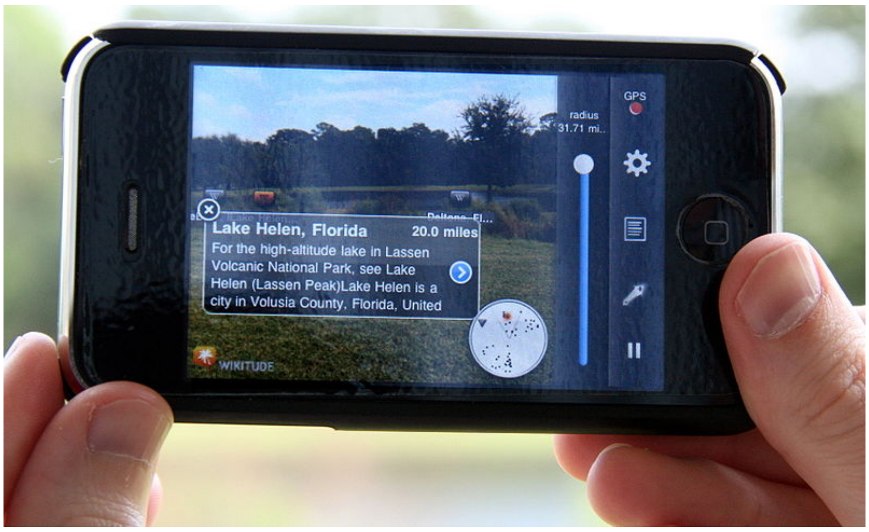
Lake Helen, Florida
20.0 miles
For the high-altitude lake in Lassen Volcanic National Park, see Lake Helen (Lassen Peak)Lake Helen is a city in Volusia County, Florida, United
radius 31.71 mi..
GPS
WIKITUDE

Figure 7: Wikitude application [1]

this could mean, a point p^c in a real world scene in camera coordinates and its corresponding point p^i on the camera image. The following explains some additional properties of this concept and how they are used in a computer vision context. Additional information about this topic can be found in [39].

3.2.1 Orthographic projection

Orthographic projection is a simple projection model, that works by omitting the depth information from the 3D point. This means all objects appear at the same size, no matter the distance.

$$p^i = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} p^c$$

in homogeneous coordinates:

$$\begin{pmatrix} x^i \\ y^i \\ w^i \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x^c \\ y^c \\ z^c \\ w^c \end{pmatrix}$$

3.2.2 Perspective projection

Perspective projection describes a non-linear transformation, that most closely resembles the basic process of a standard camera model (e.g non telecentric lens). Far away 3D objects are depicted smaller in the image plane, distances between points and angles between lines are not preserved.

Matrix notation can only be given in homogeneous coordinates due to the non-linearity. F denotes the focal length.

$$\begin{pmatrix} x^i \\ y^i \\ w^i \end{pmatrix} = \begin{pmatrix} F & 0 & 0 & 0 \\ 0 & F & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x^c \\ y^c \\ z^c \\ w^c \end{pmatrix}$$

3.2.3 Total perspective projection matrix

In addition to the orthographic or perspective projection more transformations are needed for an accurate representation of an actual camera system. The total perspective projection matrix describes the necessary transformations from world coordinates to pixel coordinates.

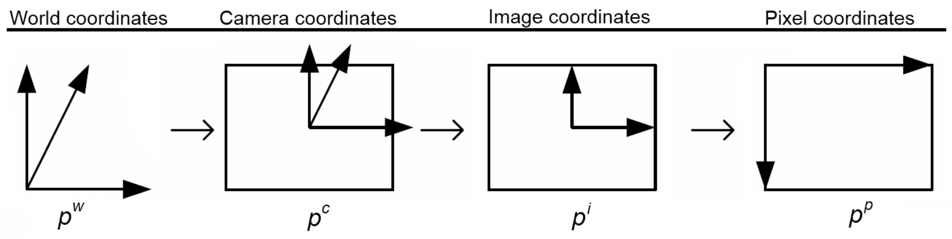

Figure 8: From world to pixel coordinates

World coordinates describe the point relative to an external point of origin, whereas camera coordinates specify the point's location relative to the camera's position. In the next step the projected point is converted to 2D image coordinates. Finally the point of origin is changed and effects like radial distortion and other camera specific parameters are taken into account when the point is transformed into pixel coordinates. Note that in other contexts such as image processing, image coordinates are usually given with their origin already transformed, that is in the upper left corner. Throughout this work the point of origin will be explicitly stated for the given image context.

The total perspective projection matrix P_t is given by:

$$P_t = K \cdot P \cdot D$$

and transforms a point $p^{\tilde{w}}$ in world coordinates into pixel coordinates $p^{\tilde{p}}$

(up to a scale factor s). The tilde denotes a point in homogeneous coordinates.

$$sp^{\tilde{p}} = K \cdot P \cdot D \cdot p^{\tilde{w}}$$

Its components are:

The camera calibration matrix K holds the internal camera parameters, also called intrinsic parameters and describes the transformation from image to pixel coordinates. The parameters vary between cameras and have to be determined by a calibration process for correct use of the system. If the intrinsic camera parameters are known, a camera is said to be calibrated.

$$K = \begin{pmatrix} k_x & s & H_x \\ 0 & k_y & H_y \\ 0 & 0 & 1 \end{pmatrix}$$

The internal parameters are:

- k_x, k_y: radial distortion parameters of the camera lens
- s: skew
- H_x, H_y: principal point, origin of the image coordinate system
- d_x, d_y: pixel size
- f: focal length

Pixel size and focal length are given by $k_x = \frac{f}{d_x}$ and $k_y = \frac{f}{d_y}$

The projection matrix P usually holds a perspective projection matrix. The product of camera calibration matrix and projection matrix is also referred to as camera matrix.

$$P = \begin{pmatrix} F & 0 & 0 & 0 \\ 0 & F & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

The external parameter matrix D comprises the rotation and translation needed to transform a point in world coordinates into camera coordinates. The rotation is given as a 3×3 matrix and the translation in 1×3 form. When given on its own, the external parameter matrix is often written as a 3×4 matrix , combining rotation and translation, and is then referred to as the camera pose. In homogeneous coordinates the external parameters matrix becomes a 4×4 matrix:

$$D = \begin{pmatrix} & R & & t \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3.3 Pose estimation

When talking about the pose estimation problem the term usually refers to estimating the external parameter matrix with prior knowledge of the internal parameters of the camera system. This is done using point correspondences between 3D points in the world coordinate system and their projections in the image coordinate system. Sources dealing with pose estimation often make no discrimination between the image coordinate system and the pixel coordinate system as the internal parameters are assumed to be known in this context, thus points could easily be transformed between the coordinate systems. This work differentiates between pixel and image coordinates.

As the camera pose is part of the total projection matrix and the internal parameters matrix are at least partly known, the missing elements of the total projection matrix have to be determined. There are different ways of calculating the missing information from point correspondences. The camera calibration process by Tsai/Lenz[48] determines the external parameters, radial distortion of the lens and focal length. The direct linear transformation (DLT) [39, 19] omits the radial distortion and directly estimates the total projection matrix, from which the needed parameters can be extracted. Even though the DLT computes internal and external parameters it is not directly used for pose estimation. In most cases better results can be achieved by first estimating the internal parameters by DLT and using a separate approach for the camera pose.

With prior knowledge of the internal parameters less point correspondences are needed and a unique solution for the pose can be achieved with 4 point correspondences, which may be coplanar but not aligned.

3.3.1 Pose estimation from a 3D plane

In [31] Lepetit and Fua give a survey about different techniques used for pose estimation of which one will be detailed here. Estimating the camera pose from a 3D Plane is an often used simplification of the pose estimation problem as 3D planes are usually easier to recognize in images than more complex scene elements. The mapping of a plane to another plane can be described as a homography, which is a 3×3 matrix.

For a point $\tilde{p^p}$ in pixel coordinates and its corresponding point on the plane in world coordinates, it holds ($R_1 R_2 R_3$ denote the columns of the rotation matrix):

$$\begin{aligned}
\tilde{p^p} &= P_t \cdot \tilde{p^w} \\
&= KP(R_1 R_2 R_3 t) \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} \\
&= KP(R_1 R_2 t) \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \\
&= H \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}
\end{aligned}$$

If the homography H, along with the internal parameters, is known, the camera pose can be calculated. The homography itself can be estimated with four corresponding feature points using a DLT algorithm.

The camera pose consists of rotation and translation; two columns of the rotation matrix and the translation vector can be gained from $(KP)^{-1} \cdot H$, as $H = KP(R_1 R_2 t)$. The third column of the rotation matrix can then be calculated by cross product $R_1 \times R_2$.

3.3.2 Image rectification using homographies

Image rectification is an important step in some pose estimation procedures, in [33] Liljequist gives an example of an augmented reality system that estimates pose information from scene planes and uses homographies to rectify images during the process. Conversely an estimated pose can be used to obtain a homography, as will be detailed below.

Image rectification describes a process that transforms a given image in such a way as to bring it into a "neutral" state, in which it is not affected by geometric image warping (e.g. a perspective warp). It is used as a preprocessing step in image processing to eliminate unwanted image warping or to align images, thus creating a common ground for image comparison.

To correct a perspective warp, the transformation describing the warp has to be inverted and applied to the image. Perspective transformations are described by 3×3 matrices or homographies. Given the correct homography H an image can be rectified by applying H^{-1}.

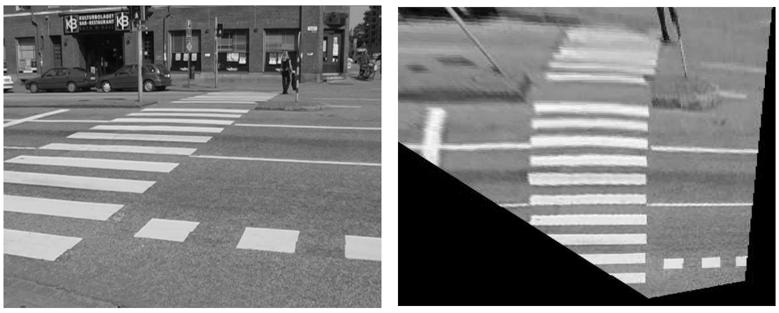

Figure 9: Image rectification example [33]

In Figure the homography H has been estimated for the street plane on the left. Applying the inverted homography H^{-1} rectifies the image with regard to the street plane, resulting in the right image.

Similar to the aforementioned procedure to calculate the pose matrix from an image homography it is possible to calculate a homography from a pose matrix by omitting a rotation column from the pose matrix (the external parameter matrix D in the total projection matrix).

$$\begin{aligned}
\tilde{p^p} &= K \cdot P \cdot D \cdot \tilde{p^w} \\
&= KP(R_1 R_2 R_3 t) \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} \\
&= KP(R_1 R_2 t) \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \\
&= KP \cdot H^{'} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \\
&= H \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}
\end{aligned}$$

In the same way the pose matrix describes a transformation from the camera's position to a new position in the scene, the resulting homography $H^{'}$ describes a transformation from the image plane of the camera to a plane inside the scene. As this transformation is happening in the camera coordinate system the result still has to be transformed into the pixel coordinate system by applying the K and P matrices. Alternatively the matrices can be directly applied to the homography $H^{'}$ resulting in H.

The plane's orientation depends on the column omitted from the pose matrix. For example, omitting the third column describes the xy-plane of the transformed coordinate system that is the result of applying the transformation given by the pose matrix. Figure 10 shows an example of applying a homography transformation. In this case the z-axis was omitted, mapping the image plane to the xy-plane of the originating pose's coordinate system x', y', z'. (note that in the figure the Homography H only maps to the image coordinate system, not to the pixel coordinate system for ease of explanation. The camera parameter matrix K would still need to be applied to transform to pixel coordinates).

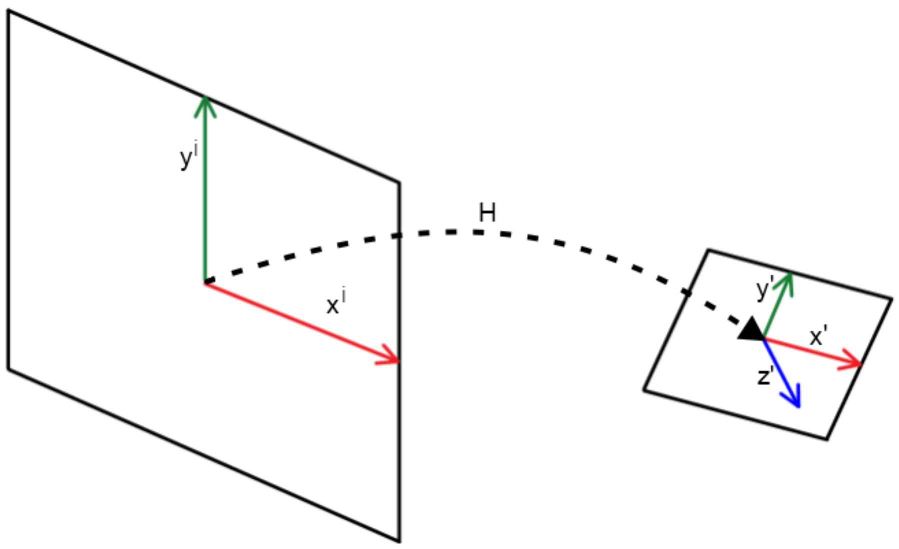

Figure 10: Applying a Homography, transforming the camera image plane

When applying a homography to rectify an image it is important to account for the fact that the coordinate system of the estimated pose matrix and the image coordinate system might have different properties (e.g. handedness), the same applies to the projection and camera matrices should a system provide such information. Section 5 shows an example of an image rectification process using homographies.

3.4 Sketch recognition

Sketch recognition is a part of human-computer-interaction that aims to identify and, in some cases further process, hand-drawn sketches. It was first used by Ivan Sutherland in 1963 [47] to assist users in drawing circuit board diagrams. Since then the focus of sketch recognition has expanded to more varied tasks, though some approaches remain very specialized in their intended use. There are, for example, systems to beautify hand-drawn strokes [40], recognize UML diagrams [35] and enable gesture input on touch screen devices [32].

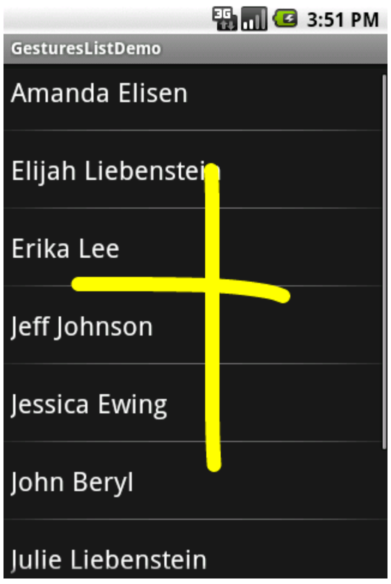

Figure 11: Multistroke Gesture for adding a new contact. Android GestureLibrary [20]

Sketch recognition can be divided into online and offline techniques. Online sketch recognition works with trajectory information in addition to the point coordinate information and involves the use of tablet interfaces or digital ink systems like the Anoto pen [2]. Online approaches usually achieve better results due to the cleaner input data and additional trajectory information, though not all systems make use of it.

Offline sketch recognition is based on sketches drawn on paper or other non-digital media, thus creating the need to digitalize the sketches first. The approach here is to work with scanned-in documents. As a consequence offline approaches usually pay less attention to system performance and real-time capability, whereas some online recognition systems operate in or near to real-time to enable direct feedback to the user. Systems for of-

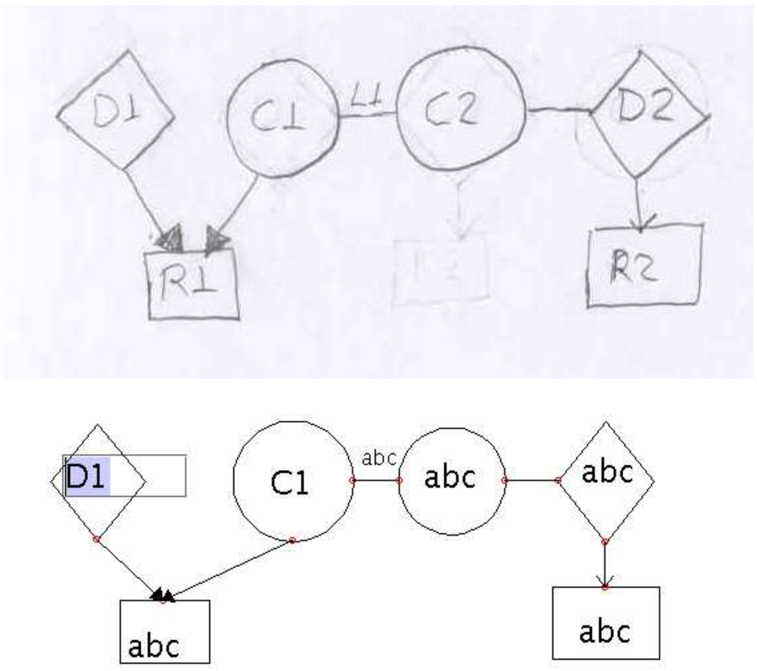

Figure 12: UML paper sketch and recognized diagram (partly user edited)[35]

fline sketch recognition that use a different approach were not found during research, neither were offline systems that are running in real-time.

As there exist different approaches, a brief summary over the techniques used in Sketch Recognition is given, further summarizing comparisons from Oltmans [37] and Hammond [25]

3.4.1 Stroke based

Stroke based recognizers are often used for interactive systems, where simple inputs are used in contrast to complex sketches. They are similar to (and in some cases overlap) with techniques often called gesture recognition, which is employed in touch screen interfaces to provide simple interaction techniques.

Early stroke based recognizers, like the one proposed by Rubine [45], operate on single stroke sketches or gestures, which are defined as one consecutive pen motion, started by a pen down press and ended with the pen lifting. They analyze the properties of the stroke, like distance between start and end points, angle between parts of the stroke and parameters of the bounding box. Based on this information a matching process against a template database is started to classify the stroke. As a consequence of the chosen properties the recognizer is sequence sensitive, which means the system can differentiate between a circle that was drawn clockwise versus one that was drawn counterclockwise. If that feature is desirable depends on the application. For example a gesture based interaction system for touch screens would want to differentiate the direction of a horizontal stroke, if that gesture is used to flip forward or backward in a reader.

3.4.2 Geometry based

Other stroke based recognizers, often called geometric recognizers, make use of a hierarchical system and work with multiple strokes. The lowest level is formed by simple primitives like lines, arcs, and circles, which are parameterized by starting and end point and other features depending on the primitive. The next higher level are component structures like triangles and rectangles. E.g. a triangle is formed by three lines with sequentially overlapping start and end points. User-defined semantic structures are on the highest level. A triangle on top of a rectangle for example, could be classified as a house, when the triangle's base line and the rectangles top line have similar properties or are identical. An example for a hierarchical recognizer is Hammond's LADDER system [25].

One of the most difficult parts of sketch recognition is the classification of the sketches. For a given set of strokes multiple interpretations are possible. This is one of the reasons many systems choose to work in a specific semantic environment (like UML or circuit diagrams). The SketchREAD system [8], also a hierarchical recognizer, creates multiple hypotheses for a given set of strokes, where high level interpretations can cause change in the interpretation of low level primitives. If, for example, in the context of UML diagrams, the user drew a sloppy rectangle for a class entry, which is classified as a circle instead of a rectangle, the system could then reclassify that circle as a rectangle if other factors indicate a plausible high level interpretation. In this example there could be arrows pointing towards the circle, indicating an inheritance relation, thus strengthening the rectangle hypothesis.

3.4.3 Global properties based

Other sketch recognition systems work with global properties, choosing to forgo the analysis of individual strokes. Shape based systems, like the one by Apte et al. [10], analyze global parameters like the bounding box area and convex hull area, classifying sketches by the comparison of these properties. E.g. a sketched rectangle's bounding box is nearly identical to its convex hull area, but for a sketched triangle these values differ by about half.

These systems analyze sketches by the way they appear and not by the way they were drawn, a fact whose desirability again, depends on the application. A problem global properties based recognizers share is the insensitivity towards smaller details as those usually don't affect the global parameters in a significant way.

3.4.4 Vision based

Vision based approaches use techniques of computer vision to analyze user sketches. They are the most suited approaches for use in offline sketch recognition, as they don't require trajectory information. They often use template matching techniques to classify normalized sketch images using different kinds of metrics like euclidean or Hausdorff distance [28]. Oltmans' approach relies on segmenting a sketch into several visual parts, analyses them and then classifies the sketch according to the composition of the parts, thus combining a vision based approach with the hierarchy idea of geometry based recognizers [37].

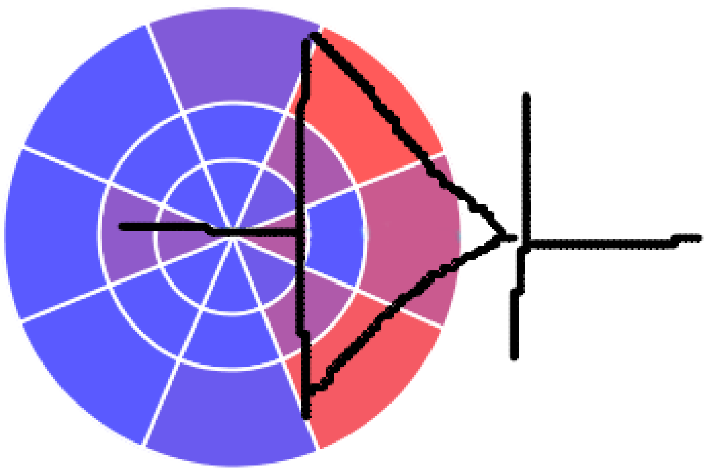

Figure 13: Bullseye structure to separate visual parts [37]

Another example for vision based offline sketch recognition can be found in Notowidigdo's work [35]. In this case hand drawn UML diagrams are recognized in a multi stage process: After preprocessing of the image, the individual diagram elements are located via specialized detectors (rectangle, circle, arrow, etc.). From these elements interpretations are generated in two different ways. Global interpretations are based on heuristic filters that use domain specific information to eliminate false positives from the detectors. In this way, for example, a shape that is detected by the rectangle as well as the circle detector can be eliminated as there are no overlapping shapes in the context of UML diagrams (see figure 14). Afterwards local properties of the shapes are analyzed to generate new interpretations and find the best global interpretation. All of these are then presented to the user for selection.

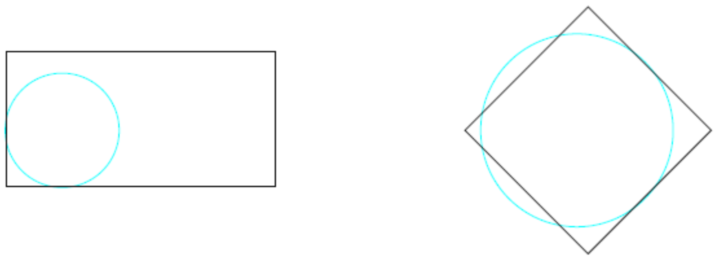
Figure 14: Overlapping detected shapes, [35]

3.5 Dungeons

Dungeons are a common environment type in roleplaying games. They provide an exploration setting for the players filled with different challenges for them to overcome. In this work the user created sketch is interpreted as a dungeon drawing and the virtual scene elements take the form of the dungeon inhabitants, whose behavior is controlled by an agent simulation [18]. The dungeon environment offers several advantages for the application:

Dungeons provide an effective way to convey information about the environment and help create a more immersive atmosphere for the augmentation.

They offer several interesting ways of environment interactions. Creatures in the dungeon can fight each other and have to navigate the terrain to gather resources and follow other goals. Traps and terrain features can be added to the environment for additional interactions.

Dungeon maps are usually discretized with squares or hexagons for use with a specific game rule set. This idea could be used to assist the pose estimation or guide the user with his drawings. They also provide a context for the sketch recognition. For example, dungeon maps are usually made up of two parts, the dungeon walls and annotations (for creatures, traps etc.). This can assist the application when deciding how to process different parts of the drawing.

Dungeon environments are used in a game context since the early versions of tabletop roleplaying games like Dungeons & Dragons [7]. There, dungeon maps are an important tool to convey information about the environments and the dangers it presents. The maps supplied with a written adventure are often used by the game master, who manages the dungeon creatures and other non-player characters in the game. Players in these games on the other hand, usually draw their own map while they play. Figure 16 shows an example of a dungeon map from the 1975 Advanced Dungeons & Dragons adventure "Tomb of Horrors".

Dungeon maps can also be found in computer games, where they inspired many level and game design ideas. A good example of this is the game "Dungeon Keeper" by Bullfrog Productions from 1997 [13]. Here the player is tasked with the creation and management of a dungeon and its inhabitants. Figure 15 shows a screenshot of the game.

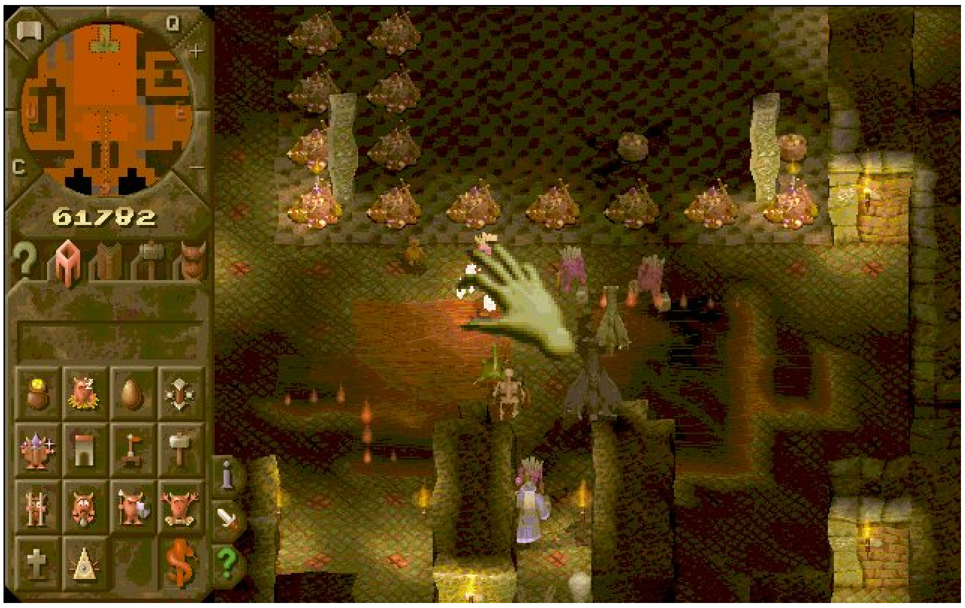

Figure 15: Dungeon Keeper [13], screenshot from Moby Games [4]

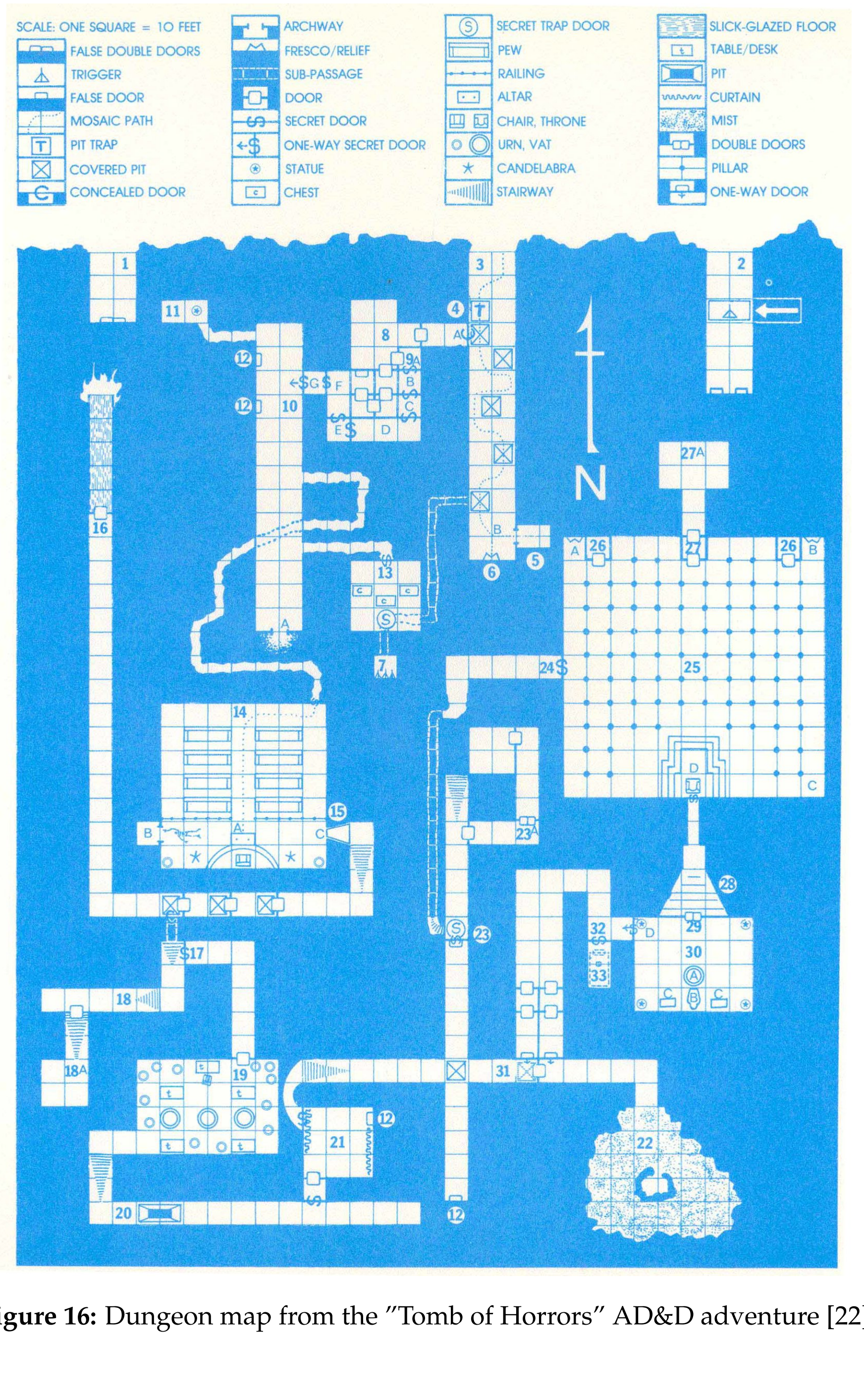

Figure 16: Dungeon map from the "Tomb of Horrors" AD&D adventure [22]

4 Component discussion

This section details some of the components that were considered during the planning phase of the work to realize the target system and describes the rationale for the selection of the used components. It does not aim to provide an exhaustive list of the current state of the art of the respective disciplines (pose estimation and sketch recognition) as this would go beyond the scope of this work. Instead it focuses on techniques that were suited to realize the given requirements. More information about pose estimation techniques can be found in Lepetit and Fua's work [31]. Survey information about sketch recognition can be found in works of Hammond [25] and Oltmans [37].

Every section will state the relevant system requirements along with a summary of the considered options and a conclusion.

4.1 Setup

4.1.1 Requirements

The setup should meet the following requirements:

- Must provide sufficient information to enable accurate sketch recognition and pose estimation
- Should be on a mobile platform
- Should only require pen and paper to draw the dungeon map

Apart from the functional requirements for sketch recognition and pose estimation, the setup as a whole is also an important factor for system usability. Tying the system to an extensive setup would deter from the uncomplicated usage scenario of only requiring pen and paper. The ease of use of pen and paper is a motivating factor for offline sketch recognition as evidenced by a quote from a test user of a system developed by Rajan et al. [43]: "One user pulled out a folded piece of paper and a pen from his pocket and said, 'This is all I need.'". Additionally a setup with a wider availability would enable a bigger number of test users as well as end users for a system.

This section focuses on setups that allow for the augmentation of an existing real-life drawing. While a similar functionality could be achieved by drawing on a tablet pc, it would lack the benefits of an augmented reality setup. A real-life dungeon drawing retains its properties without the application; it can still be viewed without the smartphone for example. In addition, this works aims to offer new ways of interaction between real and virtual scene elements and a pure virtual drawing would not offer new insights in this regard.

4.1.2 Projector based

Projector based setups are often used in a similar use case scenario to the one of this work, namely in augmented maps. Reitmayr et al. [44] for example detail a system, where a ceiling mounted projector is used to display virtual image information on top of a real world map. The setup also specifies a camera next to the projector that is used for pose estimation and capturing the data for image processing. The map is usually not moved while the program is running, so a calibration is only necessary once. This specific setup is an example of a wider range of projector based environments and other changes could be made to it, like switching top to back projection etc. For the context of this work projector based setups have the following pros and cons:

pro:

- Projected images on top of the map are visible for all users, not just a specific person and are directly on top of the actual drawing
- A fixed setup allows for a dedicated pc and thereby more system resources
- The fixed camera, projector and map locations simplify parts of the pose estimation and image processing

con:

- The setup is not mobile
- Not easily accessible by other users (due to cost and availability factors). The projector setup is very complex and would only be available in specific locations. Assembly (for development purposes as well as actual use) would be time consuming

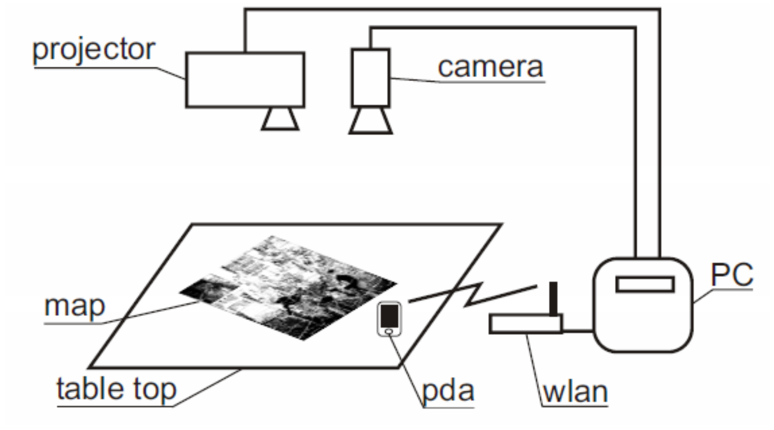

Figure 17: Projector setup for Augmented Maps [44]

4.1.3 Computer and webcam

A computer with a connected webcam would be a less specialized setup, compared to the projector and many applications make use of a webcam setup for augmented reality applications. In this setup a webcam is used to capture the sketch and a pc display is used to display information. Assets and drawbacks in this context are:

pro:

- High amount of system resources due to the connected pc
- More accessible for users than the projector setup

con:

- High amount of disconnect between the real and virtual scene elements when the information is displayed on the pc display (in contrast to the magic lens metaphor of video see-through devices or the direct projection of an augmented maps setup)
- More mobile than the projector setup, but still confined to a specific location near the pc

4.1.4 Smartphone

Modern smartphones are a valid alternative to using a dedicated fixed location pc, as they provide a high amount of system resources along with a high resolution camera. Augmented reality applications on smartphones often make use of a magic lens metaphor, where the user is looking "through" the device to perceive the mixed reality scene. This is accomplished by combining the camera feed of the front facing camera with the rendered virtual scene elements. A good example for the use of the magic lens metaphor is the Word Lens [49] application as shown in figure 18.

Depending on the operating system, the application could run on multiple device types. For example, Android smartphone applications are able to run on Android smartphones or Android tablet devices. The latter are similar in their relevant properties but would offer an increased screen size.

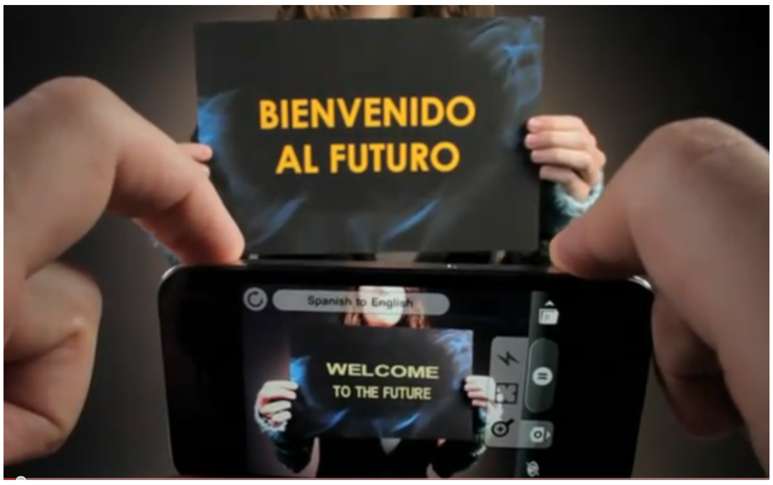

Figure 18: WordLens [49]

pro:

- Magic lens setup has a low degree of disconnect between the real and virtual scene elements
- High mobility
- High accessibility. Smartphones are widely available and increasingly popular. For example, Google announced 100 million active Android devices in May of 2011 with additional 400000 estimated daily activations [21]

con:

- Battery usage might be a limiting factor for prolonged use
- Lower system resources available than a dedicated pc

4.1.5 Conclusion

Although the direct projection aspect of the projector setup and the dedicated pc with additional system resources are good reasons for the other setups, the smartphones offers the best combination of features for this work's context. Its high mobility combines well with the idea of offline sketch recognition and its high accessibility allows for an easy development and distribution of the application. The use of the magic lens metaphor will help to heighten the authenticity of the augmented scene elements while retaining mobility, which is in contrast to the static projector based setup. In addition the question of feasibility of the system on a recent smartphone model is interesting, especially when combined with the idea of real-time sketch recognition.

4.2 Pose tracking

The pose tracking should meet the following requirements:

4.2.1 Requirements

- Must be able to provide a stable and accurate AR overlay in real-time on the target system.
- Should not obstruct the ability of the user to draw a sketch.

Although some smaller hindrances could be accepted for the drawing process, the pose estimation should cause as few obstructions as possible, this also means that it should consume as few system resources as possible. This is especially relevant due to the mobile nature of the target system, as a static client-server structure would be in contrast to the mobile aspect of the application. A high accuracy of the estimated pose is crucial for the system, as it not only affects the position of the superimposed virtual scene elements, but also the quality of the image rectification used in sketch recognition.

The following overview is based on Lepetit's and Fua's survey [31] and some additional procedures were added, that are of interest for this work.

4.2.2 Fiducial based

Fiducial based tracking is an often used method in augmented reality applications, as it offers several advantages. The fiducial elements, often called markers, assist in the detection of image correspondences and their known measurements provide information for the actual pose estimation. There are different kinds of marker based tracking systems available, of which some are relevant in the context of this work:

- ARToolkit [29] is an example of a system that uses planar fiducials that are integrated into the scene and have to be fully visible to enable correct tracking. After the image processing of the initial image the detected vertices of the rectangular marker are used as input for the pose estimation. Multimarker implementations can define transformations between individual markers allowing for extended scenes.
- Dot markers [50] are a combination of point based and planar fiducials. The system works on an plane surface that has to be prepared with a dot pattern, which is then used to similar to a planar fiducial in ARToolkit. Similar to a multi marker setup not all of the dots have to be visible at once for correct pose estimation.

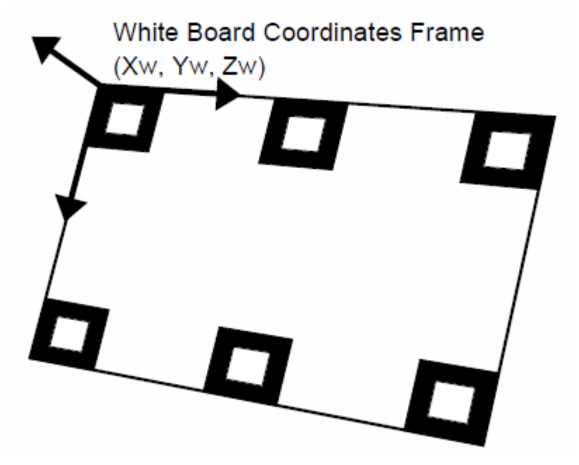

Figure 19: Example of multi marker board configuration, ARToolkit [29]

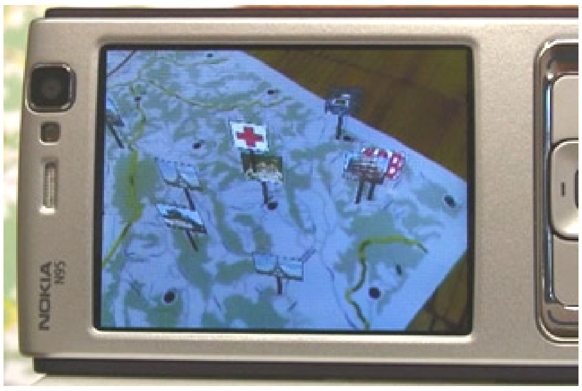
Figure 20: Dot marker [50]

Pro:

- High accuracy

- Fast pose estimation

Con

- Fiducials take up room, leaving less for the drawing
- The sheet of paper has to be prepared with fiducials, this is in contrast to the requirement of only using pen and paper, as the fiducials have to be printed.

4.2.3 Feature based

Feature based tracking methods forgo the preparation of the environment with fiducials, instead they use information from a predefined representation of the object that will be tracked. This representation can take different forms:

- Edge based tracking methods like RAPiD [26] work with a 3D model of the tracked object. From this, different properties are extracted to match with the captured image (e.g. points sampled along 3D model edges or lines sampled from an edge image of the model)
- Other systems make use of feature sets like SIFT [46] or SURF [12] and perform a feature point extraction on the object that will be tracked. Afterwards new feature points are extracted from the camera image and matched with the object's feature set. Sometimes these procedures are referred to as image marker systems [42]

Pro:

- No need for markers in the drawing
- Also offer a high accuracy

Con:

- require prior knowledge of the object that is tracked.

4.2.4 Conclusion

The conducted research has shown that there are several pose tracking options available that offer a high accuracy, which is a main priority for this work.

Because the user should not be obstructed in the way he draws, avoiding fiducials is a desirable goal, but the feature based tracking methods offer no alternative, as they require prior knowledge about the scene in a way that

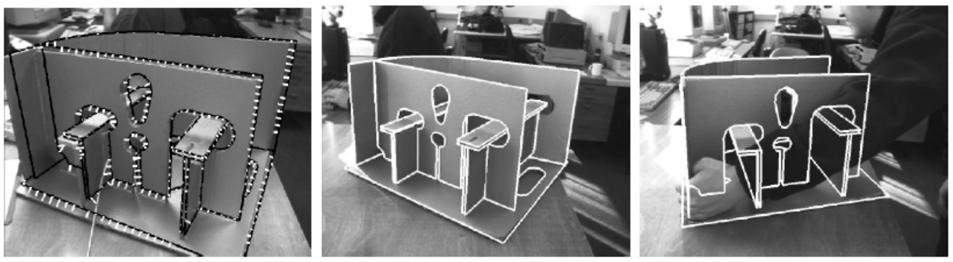

Figure 21: RAPiD implementation [16]

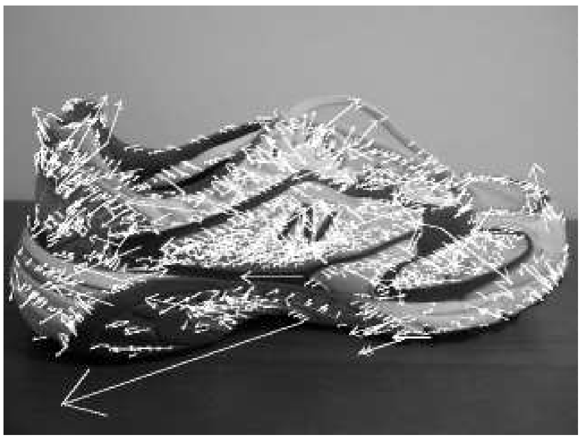

Figure 22: SIFT features detected on a 640x480 image of a sneaker [46]

is not available with the chosen setup. Large areas of the scene, in this case the sheet on which the user draws, would have to be pre-defined in order to enable tracking with SIFT [46] or SURF [12] features, as can be seen in figure 23.
While there are some limitations in place due to the chosen environment for the drawings (the dungeon context) the user would have to exactly draw certain features to the map to enable tracking. Even with this additional limitation the accuracy would suffer if the sketch itself was not accurate enough.

Using the whole sheet of paper as a fiducial would leave the sheet free of markers and allow for tracking, but would fail if only parts of the sheet were visible, which could easily happen when using a smartphone to freely look at the scene. A setup like this was used in a similar context by Hagbi et al [23].

Using several fiducial markers on the sheet reduces the available space for drawings, but the multi marker setup still provides accurate pose information when only parts of the scene are visible. This only applies to the

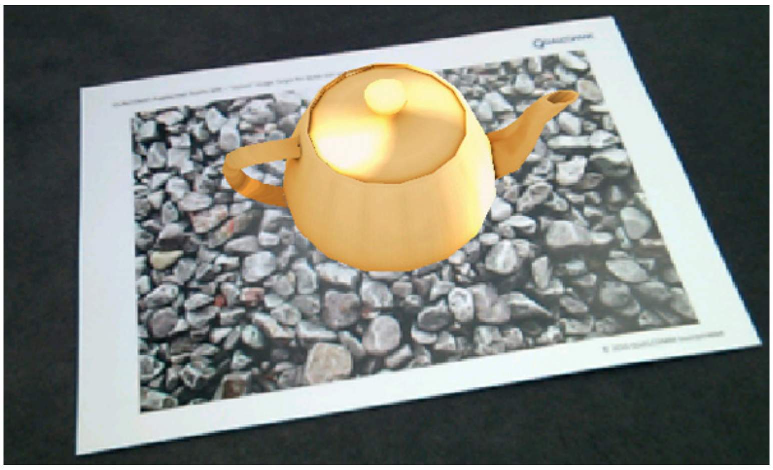
Figure 23: feature based tracking using the Qualcomm AR SDK[42]

parts of the scene in which a marker would be fully visible though, which shows that there is an important balance between the number and size of the markers and the amount of flexibility the user will have, when viewing the scene. The fiducials can also be integrated into the context of the drawing by giving them a different shape, thus making them less obtrusive than the 2d bar codes that are employed by ARToolkit. Additionally the markers can be fitted with an ID to identify them, which can be used for different purposes in the application, apart from the pose estimation.

Another way to realize pose tracking is a combination of both methods. By using a fiducial, or the whole sheet, for an initial image pose the drawing could be rectified. Afterwards the resulting image could supply features to track the rest of the image, when no marker is visible.

With the given requirements pose tracking with fiducials is the best fit among the considered options. Although the combination method mentioned above seems promising, the work would have had to focus more on the pose estimation to allow an implementation of this method in the given time frame.

The exact implementation will be detailed in the next chapter.

4.3 Sketch recognition

The sketch recognition should meet the following requirements:

4.3.1 Requirements

- Must be able to accurately detect drawn symbols.
- Must be able to correctly classify the detected symbols with good accuracy using a pre-defined list of symbol definitions.
- Must be able to detect the dungeon and create an accurate virtual representation of it that is suitable for the agent simulation.
- Optional:
  Performance should be fast enough to enable real-time sketch recognition.

Although the goal of this work is a prototypical implementation without concern for in-depth optimization, the accuracy of the sketch recognition is a central aspect of the system. The user sketches need to be recognized with every detail that holds information for the dungeon structure as a whole, in order to provide a believable augmentation.

Another interesting aspect of the system is the question of real-time capability of the sketch recognition. While real-time pose estimation procedures are available even on mobile hardware, offline sketch recognition usually pays little concerns to performance as it works on scanned-in documents nullifying the need for real-time capability.

4.3.2 Structure of the drawing area

While constructing the sketch recognition part of the system, it is important to note the two different elements that constitute the drawings: The first part is the dungeon's main structure, its walls. The only information required for the representation of a dungeon wall element, is its location. While certain additional properties could be assigned to the wall representations themselves, this will not be explored in this work.

The second part of the drawing are the symbols used to specify the additional dungeon elements. The symbol representations require location and the type of the symbol.

As there are nearly no limitations given for the drawing of the dungeon walls, a vision based approach is best suited for their recognition. No classification has to take place, thus placing the main emphasis on the correct detection of the dungeon wall's location.

For the symbol recognition several options were considered, which will be detailed below.

4.3.3 Online sketch recognition

Online sketch recognition is more common than offline sketch recognition and there are several different variants (see chapter 3.4). Among those, several work in real-time to enable direct user feedback or interaction with the system.

Most of these systems make use of trajectory information, which would not be directly available for the chosen setup of this work. There are ways to emulate trajectory information though, like Rajan proposed in [43]. Implementation complexity varies from more complex systems like LADDER [25] or Paleosketch [40] to deliberately simple ones like $1 [51] and $N [9].

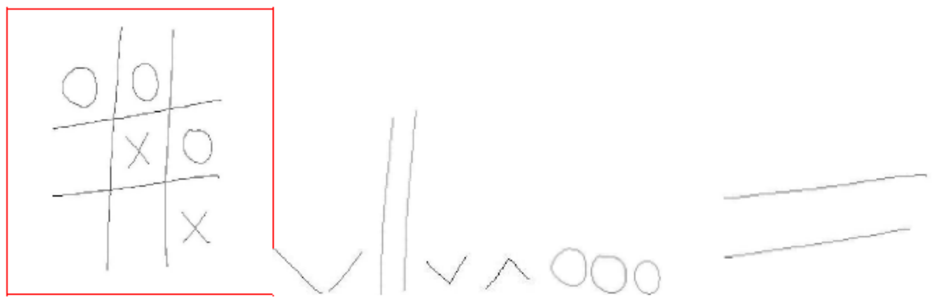

Figure 24: Paper sketch (left) and emulated strokes [43]

pro:

- Wide range of effective recognizing procedures.

con:

- Trajectory information would have to be emulated. Performance and compatibility of the emulation is untested for most recognizers.

4.3.4 Offline sketch recognition

Offline sketch recognition procedures are more suited to this work's setup due to the fact that they don't utilize trajectory information. As most offline recognition procedures work with scanned-in documents, there are usually no performance information available. Notowidigdo's [35] approach is a good example of the vision based techniques common in offline sketch recognition (see chapter 3.4).

pro:

- No trajectory information required

con:

- No performance data available
- More reliant on contextual information than online sketch recognition

4.3.5 Conclusion

Although trajectory information could be generated using a different setup (for example with real-time video capture of the drawing process) the chosen setup of this work would require the emulation of the trajectory information to use online sketch recognition. As there is little information available on this subject and because the way in which trajectory information is used, by the different online recognition procedures, varies, an offline recognition procedure seems more suited.

Most offline sketch recognition procedures make use of contextual information to compensate for the lack of trajectory information. In this regard the dungeon context does not provide as much information as the UML context. Instead the recognition process has to focus on the image information of the symbol itself.

Therefore the way sketch recognition is implemented in this work is a combination of ideas found in offline and online sketch recognizers. Protractor [32], an online sketch recognizer utilized in the Android Gesture Library, is capable of sequence sensitive recognition of gestures. It also has the option to work without trajectory information. In this case the gesture input is converted into image form and then matched against the provided templates by calculating the squared difference of the images. This is possible as the generated images are all equally aligned and sized.

To use this idea for the symbols in the drawing, an approach similar to Notowidigdo was chosen. The symbols are surrounded with another shape that can be easily picked up by a detector (rectangles or circles). Based on this information subimages are generated, which are then matched against the provided templates. The two steps necessary for this recognition process are detection and template matching. They are, depending on the exact implementation, not very demanding and could allow for a real-time sketch recognition process.

5 Implementation

This section details the prototype system that was created for this work. A basic system overview is followed by a closer look at the aspects of the system most relevant to this work, the pose estimation and the sketch recognition.

5.1 Software and hardware

This section details the selection of Hardware and Software used in this work.

5.1.1 Android OS

Google's Android operating system is available for a wide range of devices from mobile phones to tablet PCs and offers a well documented API for the various aspects of these devices. Android is used as the base of this application and provides access to the camera and the rendering environment. OpenGL ES 1.0 is used for this work, though Android supports OpenGL ES 2.0 since version 2.2. The target version for this application is 2.2 as this is the minimum version used on dual-core android smartphones.

5.1.2 Smartphone hardware

The first smartphone used for this work was the LG Optimus Speed. It provides a dual-core 1ghz cpu and 512 MB of ram and is capable of recording video in 1920x1080 resolution. The standard Android OS version of the device is 2.2.

After several problems with the LG Optimus Speed that occurred with different models of the device (spontaneous reboots, system freezes and battery drain) the development device was changed to a Samsung Galaxy S2 (GT-I9100), which provides a 1.2 ghz dual-core cpu and 1024MB of ram. Standard Android OS version of the Samsung device is 2.3.3.

5.1.3 Tracking library

The Qualcomm AR SDK (QCAR) [42] was one of the options considered to realize the fiducial based tracking of the application. The QCAR is based on the Studierstube tracker developed at the TU Graz. The Studierstube tracker itself is a continuation of the ARToolkitPlus, which was also developed at the TU Graz and is based on the original ARToolkit.

The QCAR is available for Android OS and uses the Android NDK to run native C++ code. It offers 2 different tracking options:

- Image marker tracking, which uses a feature based pose estimation approach to track predefined image targets

- Frame marker tracking, which uses planar fiducials that hold the relevant information on the border, leaving the remains of the fiducial available for other purposes (for example a graphic could be inserted to make the marker human readable)

The QCAR SDK is closed source and the API does not provide deeper access to the actual algorithms used. The use of the LG Optimus Speed provided an example where this is problematic: It was not possible to change the video recording resolution on the LG Optimus Speed. Although the API offers a parameter to select a quality setting, which is supposed to change the resolution, the parameter had no effect on the video feed of the LG Optimus Speed that was used. This was determined as the cause for the initial bad tracking results with the LG Optimus Speed. The detection of a frame marker with a width of 5cm stopped at about 30-40 cm, which could be explained by the lower resolution used (320x240). The same test setup offered good results on a HTC Desire where the quality settings did take effect and changed the resolution to 640x480.

In addition the Image marker tracking provided by the QCAR is not suitable for this work, as the image markers would require a large amount of space on the drawing area (as seen in figure 25).

While the QCAR SDK offers many advantages over its predecessors, it has some problems and limitations that led to the selection of a different tracking solution.

The tracking library used in this work is ARToolkitPlus (ARTK+). It provides tracking of planar fiducials and offers multimarker support. Originally available as an open source release with a C++ based API, it is used in this work via a Java wrapper (JavaCV) [11] to directly work in Android. ARTK+ was optimized to work on older mobile hardware (the latest official release is from 2006) and provides fast pose estimation.

The open source nature of ARTK+ might also help in improving performance for the application. As the pose estimation and the sketch recognition share some similarities in their work order, it might be possible to combine certain computations. For example both procedures use image thresholding and rectification.

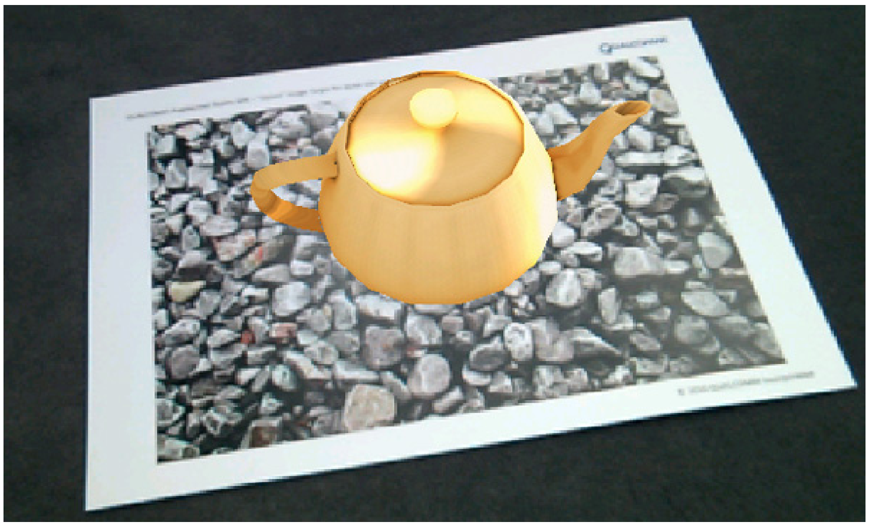

Figure 25: QCAR Image marker example [42]

5.1.4 Image processing library

Image processing for the sketch recognition part of this work is done via OpenCV, an open source image processing library originally developed by Intel [6]. It is used via the same Java wrapper as ARTK+, to use the C++ based API on Android.

5.2 Basic system overview

The prototype system combines pose estimation, sketch recognition and a simple renderer to augment user created drawings with virtual information. The exact nature of the virtual information is not specified in the system, nor is the dungeon context of the drawings. As an example application scenario the data generated from the sketch recognition is passed to an agent simulation, running in a separate thread, that is used to control the behavior of the dungeon inhabitants. Further information about the agent simulation can be found in Anica Eiting's diploma thesis [18].

5.2.1 Usage

This section details the basic usage of the application. An activity diagram, figure 26, is provided to illustrate the basic procedure and the activities are explained below in more detail:

1. The user is handed a pen and a sheet of paper, on which the marker and borders where printed (the exact layout specifications of the drawing area will be detailed in the Sketch Recognition chapter).

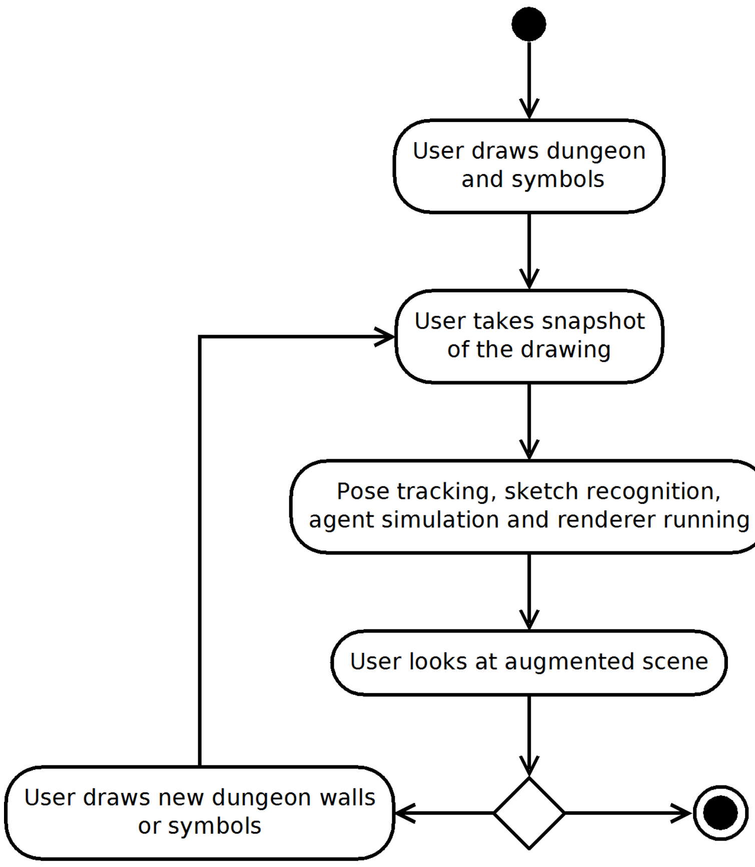
User draws dungeon and symbols
User takes snapshot of the drawing
Pose tracking, sketch recognition, agent simulation and renderer running
User looks at augmented scene
User draws new dungeon walls or symbols

Figure 26: basic activity diagram of the application usage

2. The user draws the dungeon walls and adds symbols to specify the location and type of nodes used in the agent simulation. The dungeon drawing doesn't have to be final and can be extended later on. Drawn lines are recognized as dungeon walls and symbols are marked with a rectangle or a circle.

3. The user takes a snapshot of the dungeon by pressing a button on the smartphone. This starts pose estimation, sketch recognition, the agent simulation and the renderer (the exact order is detailed later in the sequence diagram, see figure 29)

4. The user can look at the augmented scene by moving the smartphone around and using it as per the magic lens metaphor. The agents are rendered on the smartphone display and appear to be on the drawing plane. They respect dungeon walls and interact with the symbols. A screenshot of the running application can be seen in figure 27.

5. New dungeon walls and symbols can be added by drawing them and

taking another snapshot of the respective areas.

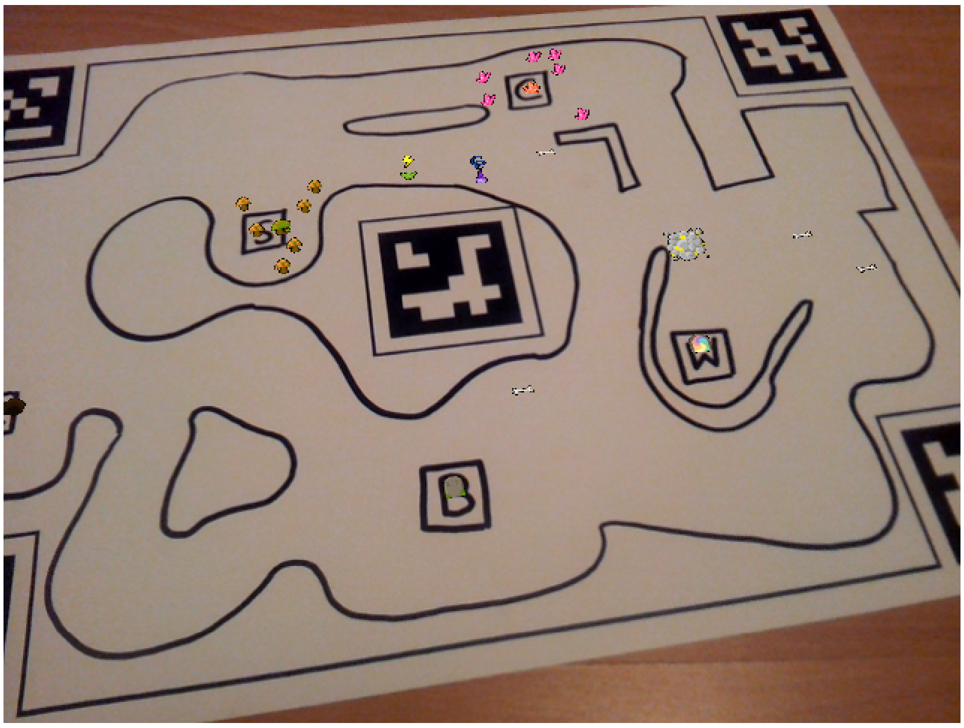

Figure 27: Screenshot of the application with running agent simulation

5.2.2 System structure

This section aims to provide an overview of the basic structure of the developed system. It focuses on the parts relevant to this work, i.e. the pose estimation, sketch recognition and rendering parts. Information about the agent simulation working in concert with the rest of the application can be found in [18].

Among the priorities for the system are:

- Loose coupling of the components to allow for easy change of the connected components.

In this case connected components are the pose estimation library, sketch recognition procedures and the actual application that makes use of the drawing's representation (here: the Agent simulation). This is achieved by reducing the communication between the components to the required minimum. For example the pose estimation receives a byte array of video data and provides pose and projection matrices to the sketch recognition and the renderer. Similarly the only the connection between the sketch recognition and the agent simulation is a method that is called when submitting new sketch recognition data and an observer that is used to pause the agent simulation during the update process of the sketch recognition.

- Support for different setups (hardware and application wise).

This refers to the different hardware specifications of the device that the application should support, as well as the actual setup components, like the size of the paper sheet and the position of the fiducials.

The first is achieved by following the Android design paradigm, where no specific device is targeted. Instead Android only defines a range of compatibility parameters for supported devices (like minimum and maximum screen size). In accordance to this, no final parameters that refer to hardware specifications are used in the application. One exception to this is the video format used to record the camera data, for ease of development and evaluation a resolution of 640x480 was used. But even this parameter could theoretically be changed to allow for different camera resolutions. For comparison, the maximum supported video recording resolution of the Samsung Galaxy S2 is 1920x1080 at 30 frames per second.

Figure 28 shows a reduced UML class diagram of the system. It shows the main components of the system, without the parts of the agent simulation. Implementation details (getter and setter methods, sub methods, auxiliary variables) were left out for the sake of readability.

The basic system structure is defined by the Android API. DungeonMastar is the Activity class, whose onCreateMethod is called at program launch. It holds the main objects for the system tasks.

The display is realized by combining two SurfaceView objects into a Layout. The first SurfaceView object is the renderer itself which outputs the virtual scene elements (named SkullRenderer because of the initial appearance of the agents). The second is the surface used to display the camera images. The layout is set to display the rendered parts of the scene on top of the camera image. The SurfaceView objects share no connection other than being combined in the layout . This fits the nature of the Android system, as generating a final image, with fixed parameters, would violate the requirement of support for variable hardware specifications. This is why a screenshot method had to be implemented manually to generate the mixed reality screenshots, again combining the image output of both renderer and

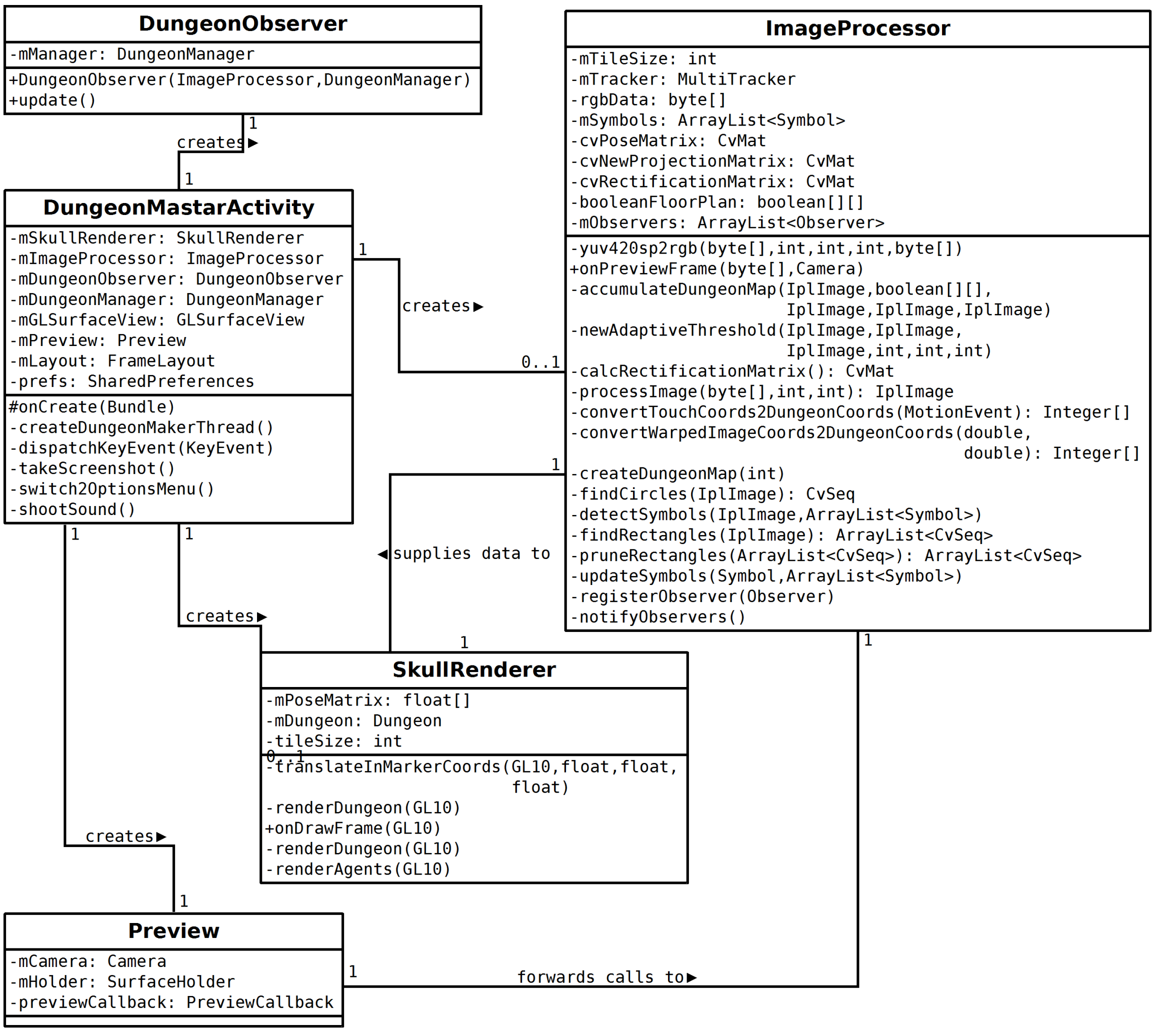
DungeonObserver
-mManager: DungeonManager
+DungeonObserver(ImageProcessor,DungeonManager)
+update()
creates ▶
1
1
DungeonMastarActivity
-mSkullRenderer: SkullRenderer
-mImageProcessor: ImageProcessor
-mDungeonObserver: DungeonObserver
-mDungeonManager: DungeonManager
-mGLSurfaceView: GLSurfaceView
-mPreview: Preview
-mLayout: FrameLayout
-prefs: SharedPreferences
#onCreate(Bundle)
-createDungeonMakerThread()
-dispatchKeyEvent(KeyEvent)
-takeScreenshot()
-switch2OptionsMenu()
-shootSound()
ImageProcessor
-mTileSize: int
-mTracker: MultiTracker
-rgbData: byte[]
-mSymbols: ArrayList<Symbol>
-cvPoseMatrix: CvMat
-cvNewProjectionMatrix: CvMat
-cvRectificationMatrix: CvMat
-booleanFloorPlan: boolean[][]
-mObservers: ArrayList<Observer>
-yuv420sp2rgb(byte[],int,int,int,byte[])
+onPreviewFrame(byte[],Camera)
-accumulateDungeonMap(IplImage,boolean[][], IplImage,IplImage,IplImage)
-newAdaptiveThreshold(IplImage,IplImage, IplImage,int,int,int)
-calcRectificationMatrix(): CvMat
-processImage(byte[],int,int): IplImage
-convertTouchCoords2DungeonCoords(MotionEvent): Integer[]
-convertWarpedImageCoords2DungeonCoords(double, double): Integer[]
-createDungeonMap(int)
-findCircles(IplImage): CvSeq
-detectSymbols(IplImage,ArrayList<Symbol>)
-findRectangles(IplImage): ArrayList<CvSeq>
-pruneRectangles(ArrayList<CvSeq>): ArrayList<CvSeq>
-updateSymbols(Symbol,ArrayList<Symbol>)
-registerObserver(Observer)
-notifyObservers()
1
creates ▶
0..1
1
◀ supplies data to
1
1
1
creates ▶
1
SkullRenderer
-mPoseMatrix: float[]
-mDungeon: Dungeon
-tileSize: int
0..1
-translateInMarkerCoords(GL10,float,float, float)
-renderDungeon(GL10)
+onDrawFrame(GL10)
-renderDungeon(GL10)
-renderAgents(GL10)
creates ▶
1
1
Preview
-mCamera: Camera
-mHolder: SurfaceHolder
-previewCallback: PreviewCallback
1
forwards calls to ▶

Figure 28: Class diagram

camera view.

The Preview class holds the camera implementation and the camera data is send to the ImageProcessor's onPreviewFrame method by means of a call-back interface.

The sequence diagram in figure 29 shows the communication between the components in the application. The Pose Estimation and the UI callback are running in the main thread, the renderer, the agent simulation and the sketch recognition all run in their own threads. The main thread has the highest priority to allow for stable pose tracking and the sketch recognition is only running for a short time after the user takes a snapshot. Afterwards the agent simulation thread and the renderer thread both run at the same time. Further performance information and profiling information of the smartphone's dual-core system can be found in the evaluation section.

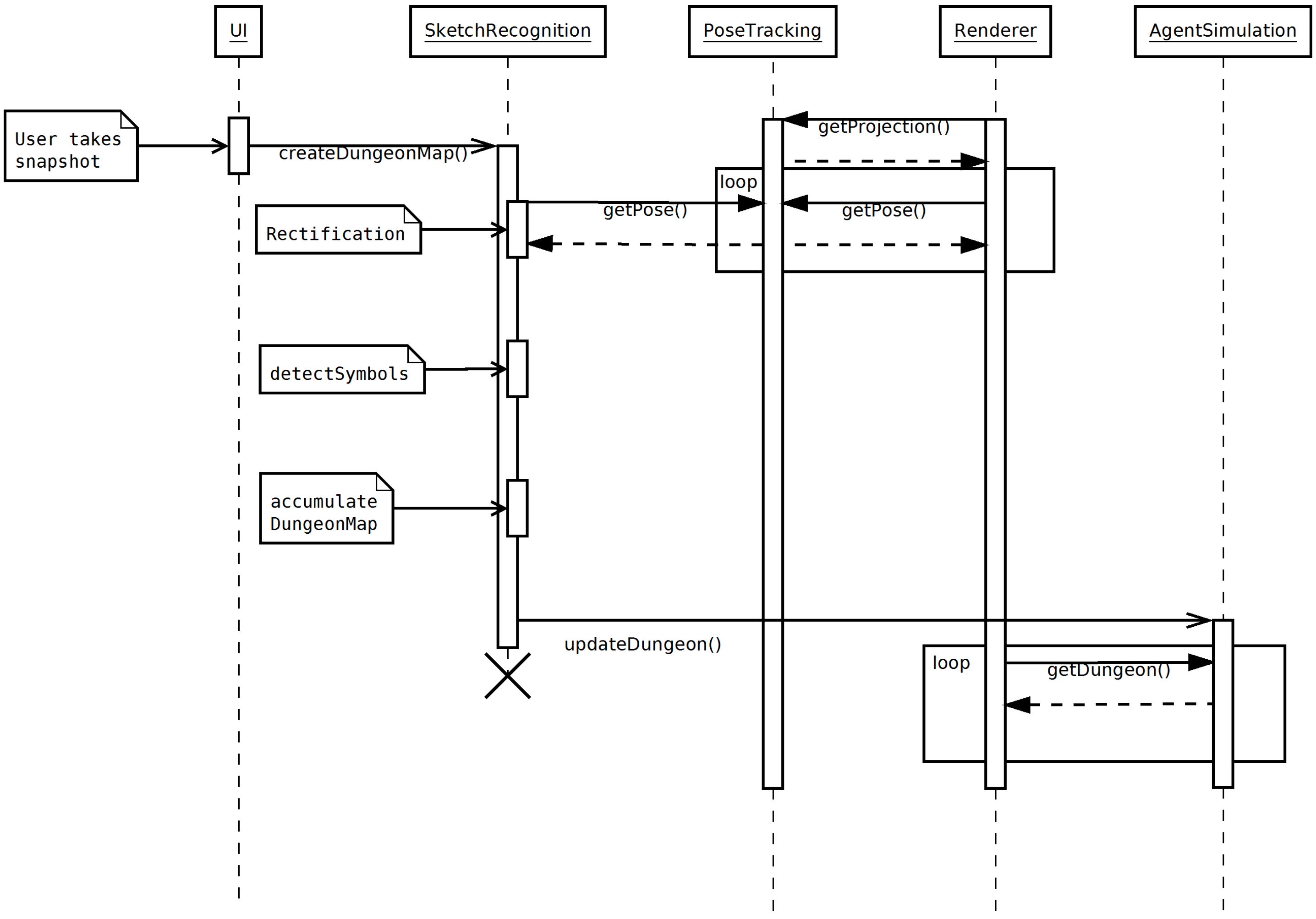

Figure 29: Sequence diagram

5.3 Pose tracking

ARToolkitPlus is used for pose estimation in this process. As the C++ based API is directly available under Android through the JavaCV wrapper, the implementation is similar to other implementations of ARTK+. A Tracker object is created and initialized with the specified parameters (image size, video format, pose estimation algorithm that should be used etc.). Afterwards the video images are supplied to the calc method and a pose matrix is calculated for every frame. ARTK+ also provides a projection matrix which is used by the renderer and the sketch recognition.

As camera calibration data is needed for the pose estimation, ARTK+ provides a default calibration file, which provides good results on most devices. Hence, no device-specific calibration file was created.

Of note is the conversion of the image format that had to be done in order to create a compatible byte stream input for ARTK+. As the Android camera implementation only supports the YCbCr video format (at least on the smartphones available for testing), the data is converted into RGB format. For performance reasons a native function is used to convert the data.

To use a multimarker setup another file has to be supplied to ARTK+ which contains the relative locations of the markers. A Multimarker board configuration file was created according for the designed multimarker board (see Sketch Recognition for details) with values accurate to 1mm. In the configuration file every marker's position is noted in form of a transformation matrix, relative to a separate coordinate system, so markers can be translated and rotated relative to each other (though only relative translation was used in this configuration). The pose matrix generated by ARTK+ in multimarker mode denotes a position that lies in the origin of this separate coordinate system. See figure 30 for an example.

5.4 Renderer

Rendering is done via the OpenGL ES 1.0 implementation supplied by Android and is used to render the virtual scene elements (in this case mainly the agents populating the recognized dungeon). Details about the visual representation of the agents can be found in [18]. Their position is relative to the detected markers in the camera image. That means if no marker is visible, the location of the agents is no longer relative to the sheet of paper until the marker is visible again and a new pose matrix is calculated.

Pose and projection matrices are given to the renderer in standard OpenGL column major order, so for example the translation vector in the pose ma-

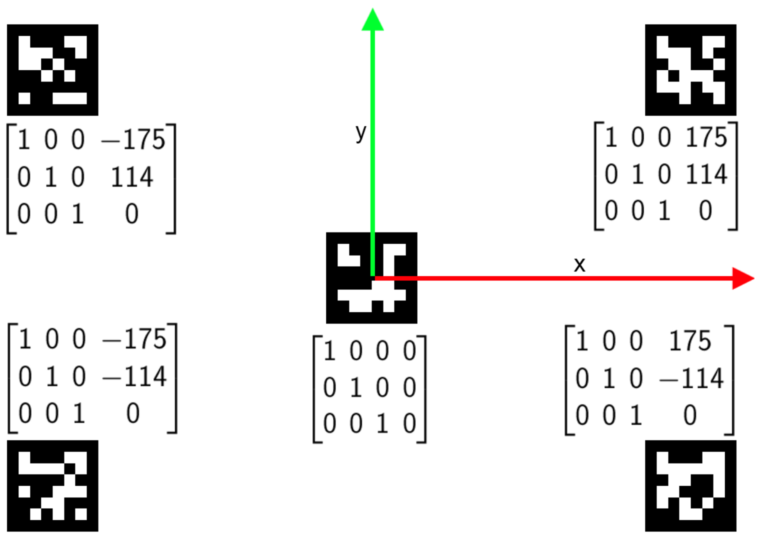
$\begin{bmatrix}1&0&0&-175\\0&1&0&114\\0&0&1&0\end{bmatrix}$
y
$\begin{bmatrix}1&0&0&175\\0&1&0&114\\0&0&1&0\end{bmatrix}$
x
$\begin{bmatrix}1&0&0&-175\\0&1&0&-114\\0&0&1&0\end{bmatrix}$
$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix}$
$\begin{bmatrix}1&0&0&175\\0&1&0&-114\\0&0&1&0\end{bmatrix}$

Figure 30: Multimarker board with relative transformations and ARTK+ marker coordinate axes. Z-axis points toward the viewer

trix is in position 13, 14, 15 and 16 of the array. (See figure 31) Afterwards the matrices are loaded by calling glLoadMatrix() to replace the current modelview or projection matrix.

$$\begin{bmatrix}R_{11}&R_{12}&R_{13}&t_x\\R_{21}&R_{22}&R_{23}&t_y\\R_{31}&R_{32}&R_{33}&t_z\\0&0&0&1\end{bmatrix}\longrightarrow\begin{bmatrix}1&5&9&13\\2&6&10&14\\3&7&11&15\\4&8&12&16\end{bmatrix}$$

Figure 31: Pose matrix and OpenGL array positions

For debug and evaluation purposes an additional method to translate agents relative to the bottom left of the detected dungeon as well as the option to render the detected dungeon walls were implemented. When the option from the menu is selected the walls are rendered as square tiles, relative to the marker position to appears as if they were on the sheet of paper. See figure 32 for an example.

5.5 Sketch recognition

This section details the sketch recognition part of the system in four parts. First some specifications for the materials used for drawing the sketch are given. The rectification, dungeon creation and symbol detection parts de-

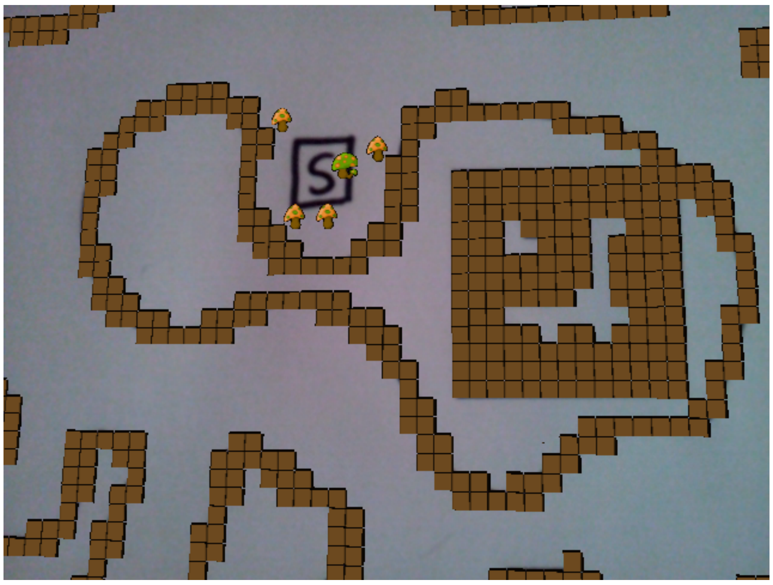

Figure 32: Screenshot with enabled wall rendering

tail the actual procedures of the sketch recognition. The rectification eliminates the perspective warp in the camera image. The dungeon creation discretizes the image to create a suitable representation that is usable by the agent simulation. The symbol detection recognizes the user drawn symbols and also sends this information to the agent simulation.

Images and matrices mentioned in this chapter are used via the corresponding OpenCV formats (IplImage and CvMat).

5.5.1 Material specification

In order to use marker tracking and provide better results for sketch recognition some specifications for the drawing area were made. Figure 33 shows a picture of the prepared sheet.

The size of the paper sheet is A3, to provide enough room for drawings in the presence of the markers. Marker size is an important factor for the maximum distance for stable tracking. The marker size was set to 5cm in this application. Different paper formats and marker sizes are possible. An A4 sheet of paper could be used with only one central marker for example. As the evaluation should incorporate multi marker tracking other paper formats were not evaluated further in this work. All printing is done in black and white and the pen provided for drawing is a permanent marker with 3mm line strength.

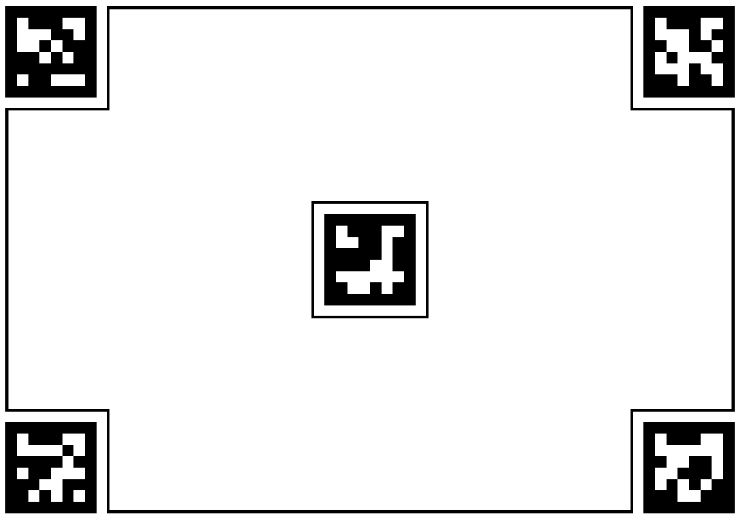

Figure 33: Prepared sheet used for drawing

According to the multi marker setup five markers were placed in the drawing area. More and bigger markers would have a positive effect on tracking accuracy and stability, but would leave less room for the actual drawing. The chosen marker arrangement leaves most of the central area of the sheet free and still provides marker tracking for the outer regions. Arrangements with 4 or less markers gave unsatisfactory results, as the markers had to be positioned in a more obstructive way or would not allow for stable marker tracking when viewing the outer regions of the drawing area. The marker size of 5cm allows for stable tracking up to about 70cm distance with a video resolution of 640x480.

As the markers used in ARTK+ require a white border around the black marker interior an additional border line was added to the sheet. It separates the marker area from the drawing area and assists the user in identifying the drawing area. The border lines also have a line strength of 3mm.

5.5.2 Rectification

As shown in section 2, the rectification process allows for a correction of the perspective warping that is present in the camera image. This is done by applying a homography to the image. This means that every pixel of the input image is transformed to a new location in the result image, according to the information in the homography matrix.

Several steps are necessary to calculate the final homography matrix which is then used with the OpenCV method cvWarpPerspective:

To calculate the homography matrix the camera matrix (the product of camera calibration matrix K and and projection matrix P) as well as the external parameter matrix D have to be known. The external parameter matrix D is available in the form of the pose matrix and is provided by ARTK+ in the form of a OpenGL modelview matrix (in column major order).

Instead of a camera matrix ARTK+ only provides an OpenGL projection matrix, which is not equivalent to the projection matrix P, as it is created specifically for OpenGL pipeline use. The OpenGL projection matrix represents a transformation between the camera coordinates and the normalized view volume and is in this way similar to the camera matrix. ARTK+ makes use of other matrices but does not give access to these. As a result a camera matrix usable for the homography had to be reconstructed from the projection matrix ARTK+ provides.

With the information given by one of the developers of ARTK at [3] the transformation ARTK (and ARTK+) uses to calculate the OpenGL specific parameters could be inverted. The resulting matrix transforms camera coordinates to screen or pixel coordinates. The calculations are shown below, P denotes the elements of the OpenGL Projection matrix with their respective row and column

$$\begin{bmatrix} P_{00} * mWidth/2 & P_{01} * mWidth/2 & mWidth/2 & 0 \\ P_{10} & -P_{11} * mHeight/2 & mHeight/2 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The next step is to multiply the camera matrix with the external parameter matrix D, which is available in form of the pose matrix. From the resulting 3x4 matrix a 3x3 matrix is formed by omitting the third column of the rotation matrix as shown in section 3.3.2. This preliminary homography H_{pre} holds the information of the perspective warp present in the image. It is then inverted in order to be usable for rectification.

Applying the inverted homography still would not yield the desired results though and additional modifications have to be made to the matrix for a correct rectification. The OpenCV method cvWarpPerspective works by applying a transformation to every pixel of the input image and writing the result into the output image. To achieve the desired results the coordinate systems of the input and output images have to be correctly aligned. OpenCV uses a standard image format for the IplImage class, whose origin is in the upper left corner. The ARTK camera coordinate system has its origin in the image center though. Applying the homography as is, would result in a partially visible image, because the resulting image would still be drawn around the origin of the result image. Pixel, whose position would result in negative values after the warp, would not be visible. Figure 34 shows an example of the partially visible image, a black border was added to show the image boundaries.

To correct this, a translation has to be applied that corresponds to the different locations of the image origins. The translation offset matrix T_o is then applied to the homography.

$$T_o = \begin{pmatrix} 1 & 0 & mWidth/2 \\ 0 & 1 & mHeight/2 \\ 0 & 0 & 1 \end{pmatrix}$$

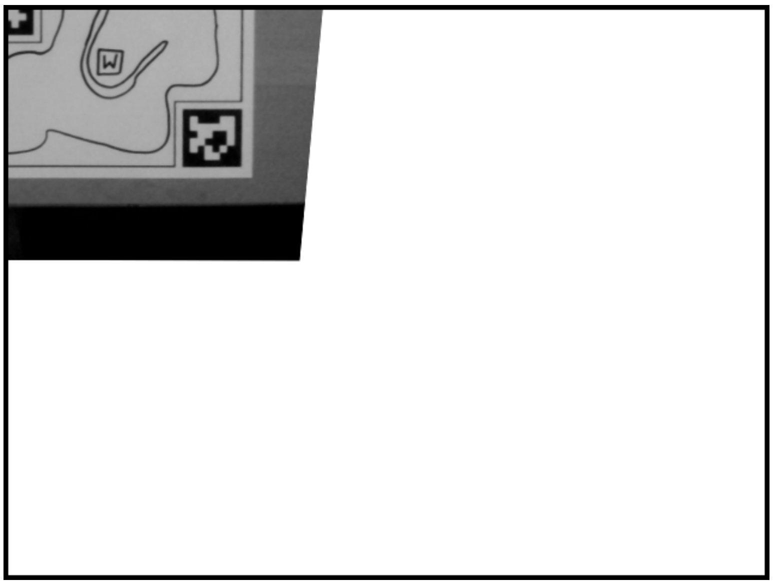

Figure 34: Result of image warp without translation offset

The last modification necessary for a correct rectification is an image flip around the y-axis. The coordinate systems used by ARTK+ can be seen in figure 35. The pose matrix supplied by ARTK+ transforms from camera to marker coordinates. As can be seen in figure 35, the marker coordinate system is right handed, with the x-axis pointing to the right and the y-axis pointing up. Compared to the image coordinate system of the IplImage the y-axis is flipped. To compensate for this a y-flip matrix F_y is applied to the homography as well.

$$F_y = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

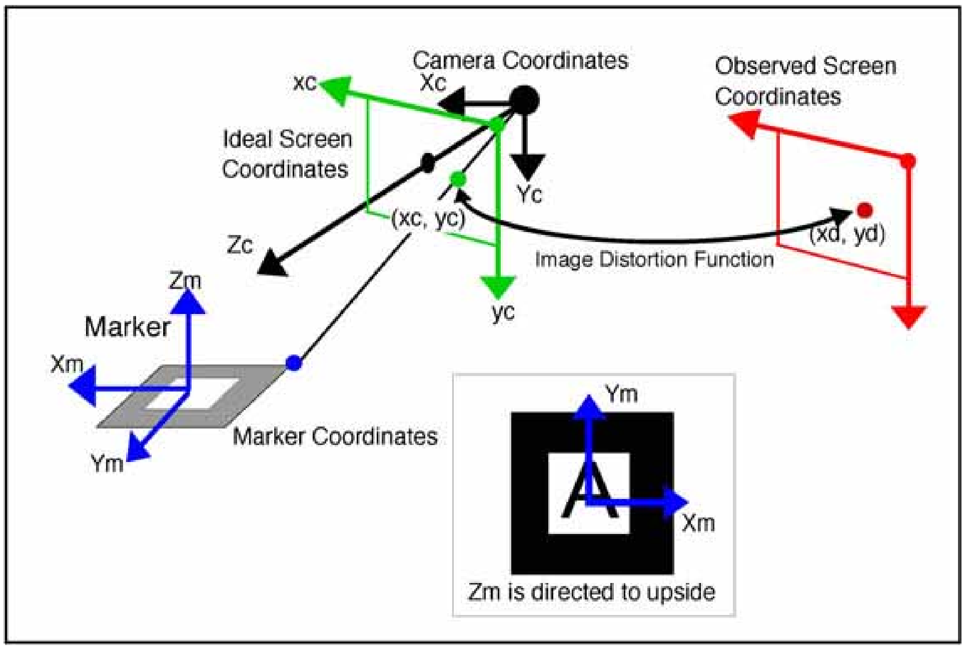

Figure 35: ARToolkit coordinate systems

The complete process for the construction of the rectifying homography H_{rec} is:

$$T_o \cdot F_y \cdot H_{pre}^{-1} = H_{rec}$$

The rectification matrix is then applied to the input image, in IplImage format, with the OpenCV function cvWarpPerspective. To save system resources the camera image is converted to grayscale, as the warp has to be applied to every channel of the input image. As the sketch itself is in black and white no relevant information is lost in this way.

The accuracy of the warp is directly dependent on the quality of the estimated pose. Further information about rectification accuracy can be found in the evaluation chapter. Figure 36 shows an example of the rectification process.

As can be seen in the examples images, some pixel in the resulting image of the cvWarpPerspective function are left white by the warp. This is due to a

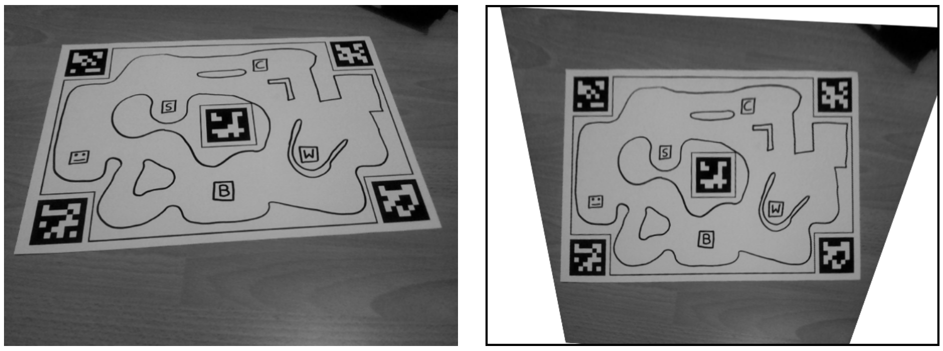

Figure 36: Rectification example

parameter set in the function, that controls the color of outlier pixel. Outlier pixel are pixel of the result image that were not mapped with a result of the warp. This occurs because the rectification also encompasses a scaling factor. When the correct values for marker size are supplied to ARTK+ the warp with the rectifying homography creates a scale of 1mm to 1 pixel in the result image. This information can be used to crop the result image and to extract only the region containing the drawing area, as the dimensions of the sheet are known. The cropped region image always has the position of the central marker in its middle. Figure 37 shows the cropped result image.

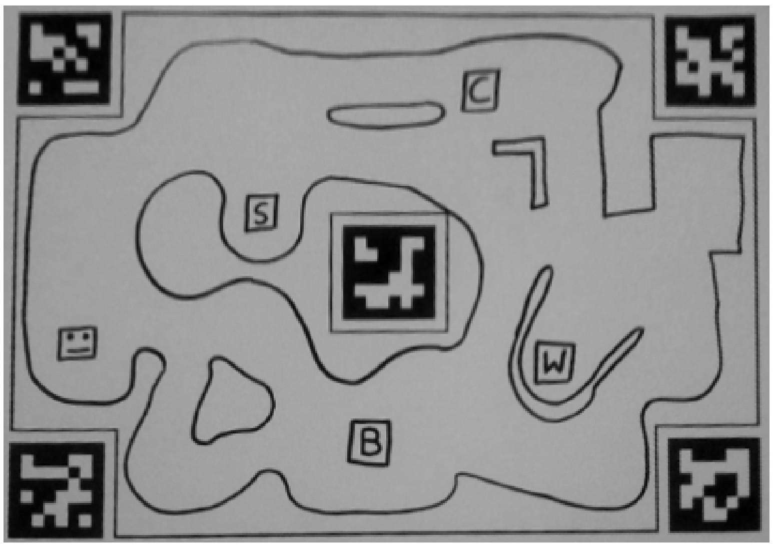

Figure 37: Rectified, cropped image

The outlier information is also saved in a separate image containing only the outlier information. The outlier image is created by writing the outlier information into a separate channel of the result image. It is then split into the outlier image and the region image containing the rectified camera image. The outlier image is cropped along with the region image. Figure 38 shows an example of the outlier image.

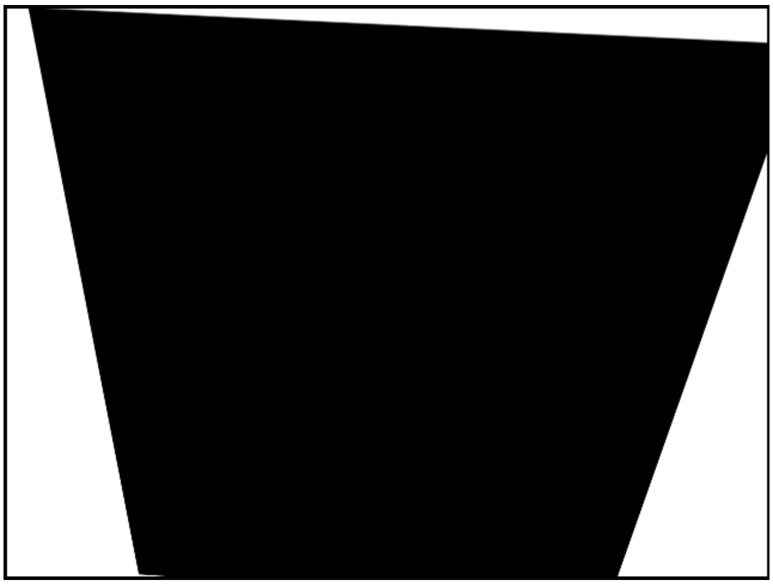
Figure 38: Outlier image (prior to cropping)

The outlier image is used later to assist during the accumulation of the dungeon (see section 5.5.3), in case the camera image contains only parts of the drawing area. Figures 39 to 40 show an example of the images generated from a partial capture of the drawing area:

5.5.3 Dungeon creation

This section details how the representation of the user sketch, which is later supplied to the agent simulation, is created. In the application the symbol detection is executed before the dungeon creation, but the order given here, was chosen for ease of understanding.

The dungeon creation works by discretizing the rectified region image until the desired level of information is reached. In this case binary information (wall or not wall) is the desired format and a two dimensional boolean array is used to hold the representation. According to the specifications of the agent simulation a tile, a single element of the dungeon floor, should have a size of 5mm. That means the smallest unit of discretization, one entry in the boolean array, should represent 5mm of space in the drawing area. Note that this representation only encompasses the walls found in the user sketch and does not contain the symbols that are used to convey additional

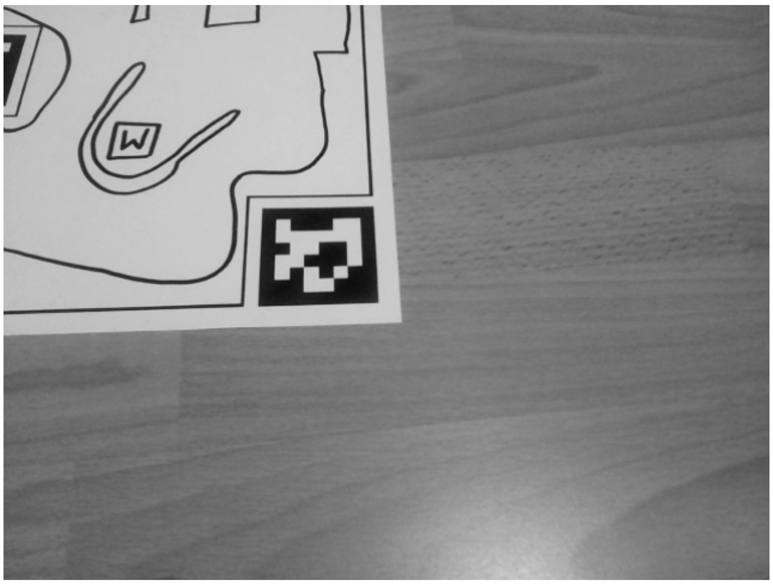
Figure 39: Camera image with partial dungeon

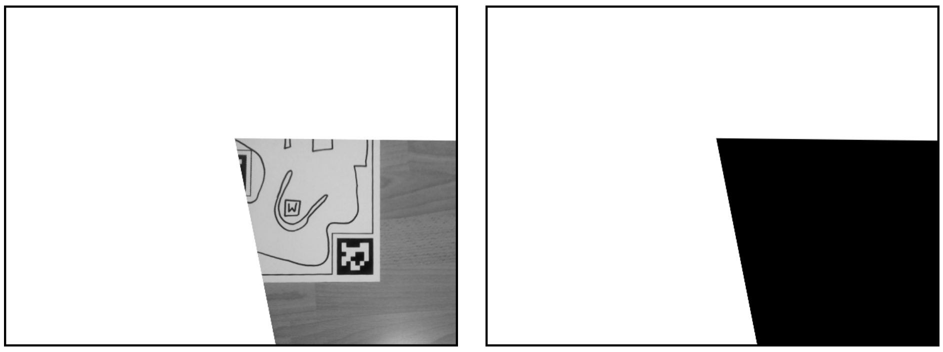
Figure 40: Partial image examples

information.

As a first step the region image is eroded, which broadens the black lines drawn by the user. This helps to maintain the sketch information over the next steps and also serves to close small gaps that might be present in the drawing. Figure 41 shows the eroded region image.

Afterwards the eroded region image is resized to achieve a 1 to 1 ratio of pixel and tiles. Due to the rectification the scale is 1mm to 1pixel, so the resize operation has to reduce the image size by the factor 5 for a tilesize of 5mm width and height.

The image decimation is an important step in the dungeon creation, as it is

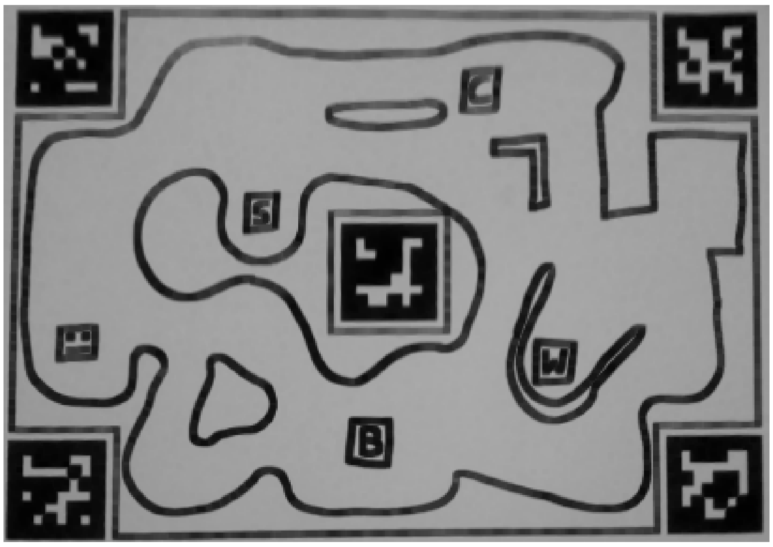

Figure 41: Eroded region image

responsible for a strong loss of image information. To preserve as much information as possible an interpolation is used during the resize operation. OpenCV offers several interpolation methods, of which linear interpolation offered the best results for the given image context. Bicubic interpolation offered no increase in image quality and nearest neighbor interpolation and area based re-sampling offered slightly worse results. Figure 42 show the resized image.

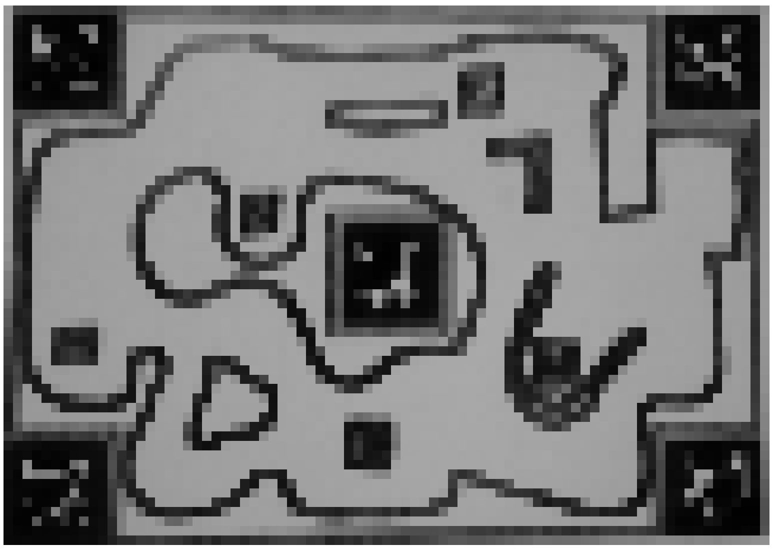

Figure 42: Resized eroded region image

After the resize operation the image has the correct size, but still contains grayscale color values. As a binary discretization is desired, a thresholding operation is applied. Several variants are available for this:

- Thresholding with fixed value

This thresholds the image according to a pre-defined value. All pixel with a color value brighter than the value are mapped to white, all pixel darker or equal than the value are mapped to black. This method was not chosen: while the threshold could be set manually, the varying lighting conditions common in the mobile use scenario of the application require a more flexible approach.

- Automatic global threshold generation

OpenCV offers the option to threshold an image with an automatically generated global threshold, using Otsu's Method [38]. This method offers the ability to work under varying lightning conditions but is still sensitive to lighting differences in the image. Even though the threshold is usually not changed dramatically in these scenarios, even small changes could result in a loss of relevant information for the dungeon creation process.

- Adaptive thresholding

The adpativeThresholding function of OpenCV offers the option to use separate thresholds for each pixel in the image. These thresholds are generated for a given neighborhood size. In this way local lighting differences can be compensated without affecting the image as a whole.

Even though the adaptive thresholding method offers good results, it still generates problematic images for the special case of images that only show parts of the drawing area. These partial images contain two parts: the partially captured dungeon sketch and the outlier area, which is marked in white. As the adaptive threshold cannot differentiate between these areas, running the algorithm results in a black line between them. This happens because the neighborhood of a pixel on the line, contains both grayscale pixel from the camera image and white pixel from the outlier areas. Figure 43 shows an example of adaptive thresholding.

To solve this problem a new adaptive thresholding algorithm was implemented as a modification of the OpenCV algorithm. It still generates a threshold over the pixel's neighborhood, but only adds those pixel to the

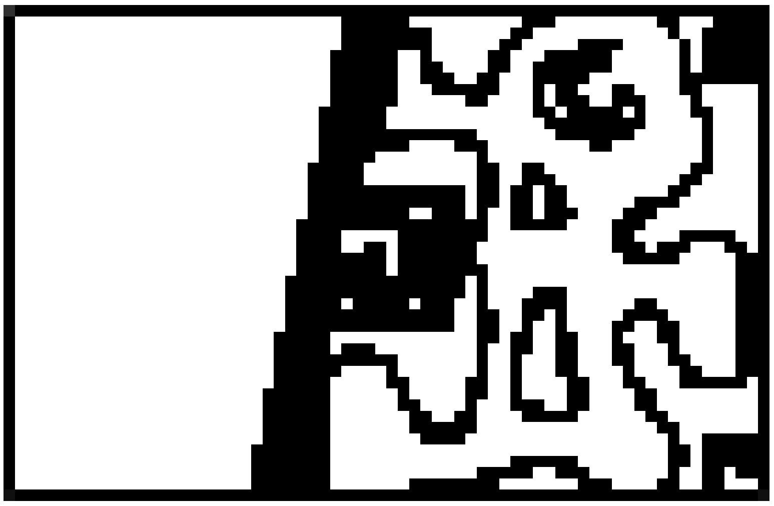

Figure 43: binary image, generated with standard adaptive thresholding algorithm (different drawing)

neighborhood that belong to the camera image, i.e. those that are not outliers. The outlier image is used to supply the necessary information.

The modified adaptive thresholding method generates satisfactory results and is used to create the binary image (see figure 44 for an example).

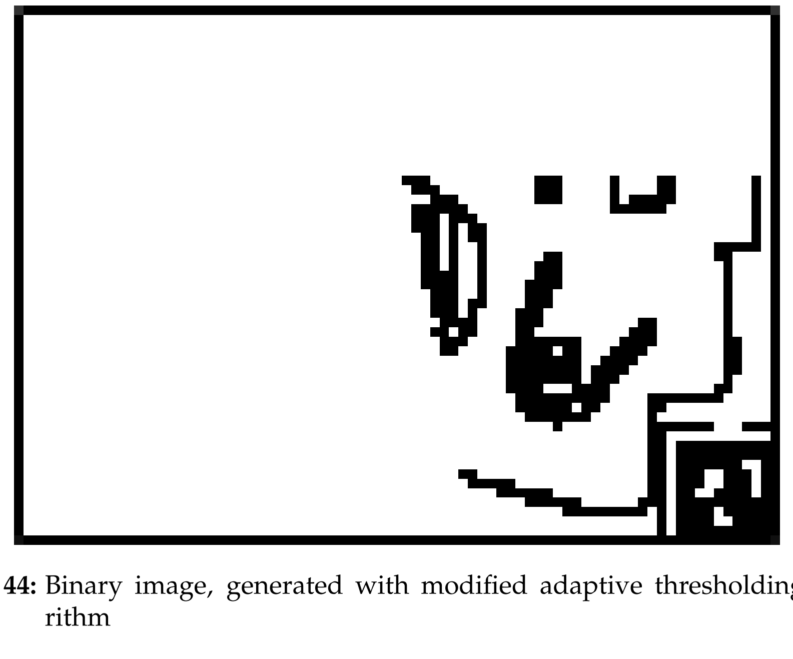

Figure 44: Binary image, generated with modified adaptive thresholding algorithm

From this binary image a two dimensional boolean array is created to represent the wall structure of the dungeon (true represents a wall tile). The array is then passed to the agent simulation along with the information about the detected symbols. Figure 45 shows another example of a binary image used to create the dungeon array.

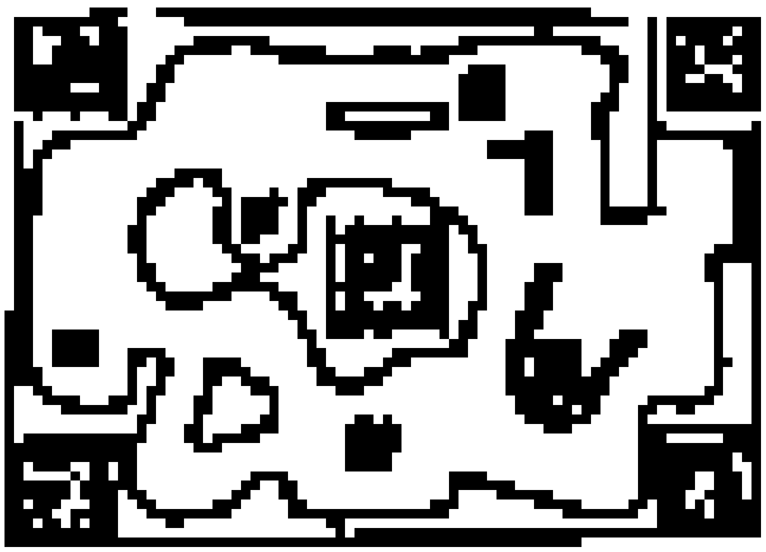
Figure 45: Binary image of complete drawing area

5.5.4 Symbol detection

The symbol detection deals with processing the user drawn symbols. This encompasses two steps, detection of the symbol area and classification of the symbol. Two methods were implemented for the first part. Both share the concept of searching for a specific user created shape. Afterwards the position information of the detected shape is used to generate a sub image for template matching, which represents the classification. The following details the two detection methods and continues with the classification part.

- Circle based

Circle based detection uses a hough transform based approach to detect circles in the image.

The first step is an image smoothing with a gauss kernel, to improve the results of the circle detection. Conducted tests have shown an increase in false positives with a reduction in false negatives. That means more shapes were wrongly detected as circles but less correct circles were missed. As the classification can be used to eliminate false positives, but there is no way to correct false negatives, using a smoothed image provides better results for the overall process.

After the smoothing operation, the OpenCV function cvHoughCircles is used to detect circles in the image. The function implements the algorithm by Kimme [30]. It works on an edge image (created using the canny algorithm [14]) and creates hypotheses for every potential circle center. A hypothesis is strengthened, if more pixel that constitute a circle are found around its center. Several parameters are available, among those: The minimum and maximum radius of a circle and a threshold that specifies a minimum number of supporting pixel for a circle hypothesis. Detected circles are returned with their radius and position in the image. A full description of the available parameters and the function itself can be found in [27].

Additional modifications had to be made to the algorithm, after initial tests had shown erroneous results of the circle positions and radii. While small uncertainties are to be expected in the detection of hand drawn circles, the deviations were significant and also present when using perfect circles (as opposed to hand-drawn circles) created with an image editing program. After some research, a bug was found in OpenCV's bug tracker that specified a problem with detection of small circles in low resolution context [5]. With the implemented changes that were suggested for the problem, the results of the circle detection improved to the estimated accuracy level. Figure 46 shows an example of the circle detection.

- Rectangle based

The second implemented detection method uses rectangles to specify the symbol area. It works on an edge image, generated with the cvCanny function of OpenCV. This is the input for the contour finding algorithm used by cvFindContours. The algorithm analyzes the edge image to find connected curves in the image. Several different options are available for the function and further information can be found in [27]. In this work the function is used to return the detected curves as CvSeq objects, each containing mainly position and next element in the curve, if available.

To reduce the number of detected points for processing, the curves are approximated with polygons using the cvApproxPoly function. This creates a polygonal approximation of the curves based on the Douglas-Peucker (DP) approximation [15]. Figure 47 shows a visualization of the algorithm. It works by first selecting the two vertices with the biggest distance from

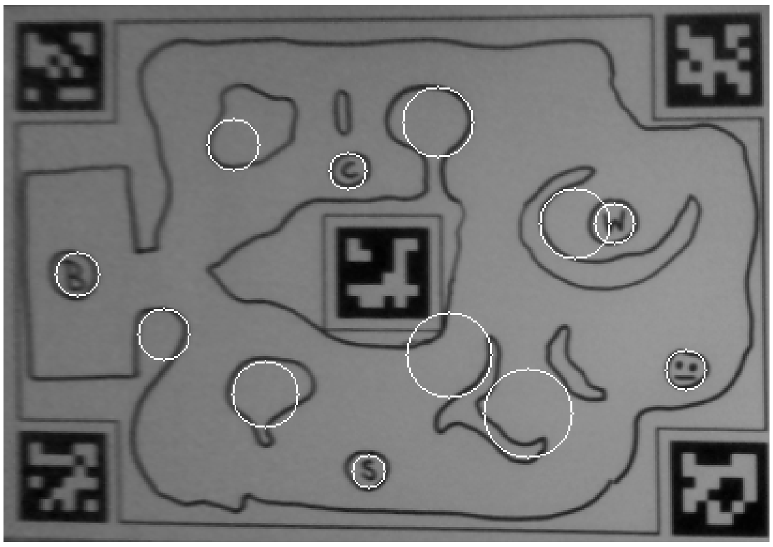
Figure 46: region image with detected circles

each other, and then iteratively selects the vertex with the next highest distance until a specified threshold is reached. Figure 48 shows an example of the detected contours and their approximation.

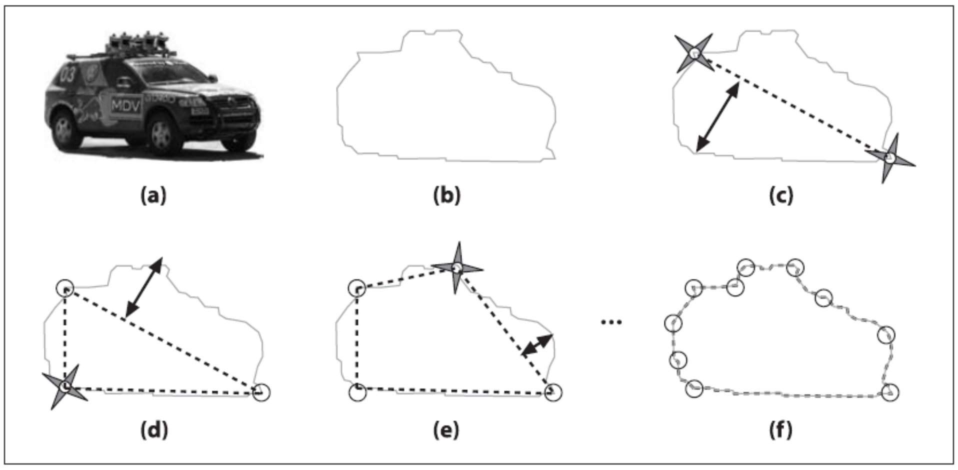

Figure 47: Polygon approximation of curves [27]

Afterwards the polygons are examined to see if they constitute a rectangle. The approximated polygon is tested for convexity and the number of its elements must equal four. A minimum and maximum area size is set to prevent the fiducial markers from being detected, as they are rectangles themselves and can contain small rectangular structures inside their area. In a final step the angle between the sequence points is analyzed, if it is

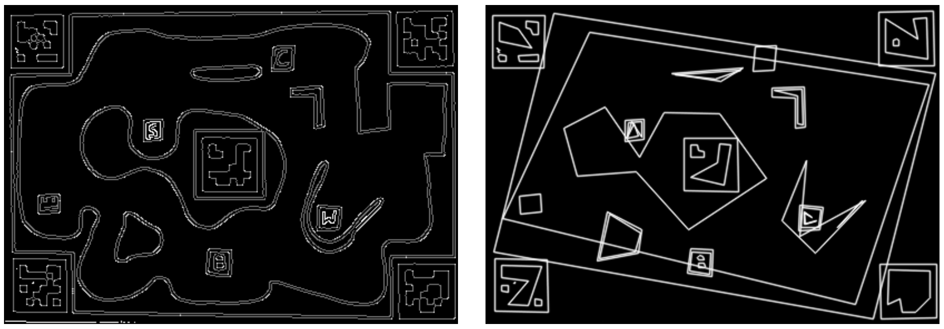
Figure 48: Contour image and approximated contour image

close to 90 degree, the polygon approximation is returned as a rectangle.

Similar to the circle detection, parameters were set to eliminate false negatives. This also produces more false positives, as some rectangles are detected twice by the cvFindContours algorithm. In a correction step rectangles that share a center with slightly larger rectangles are eliminated. Figure 49 shows the detected Rectangles.

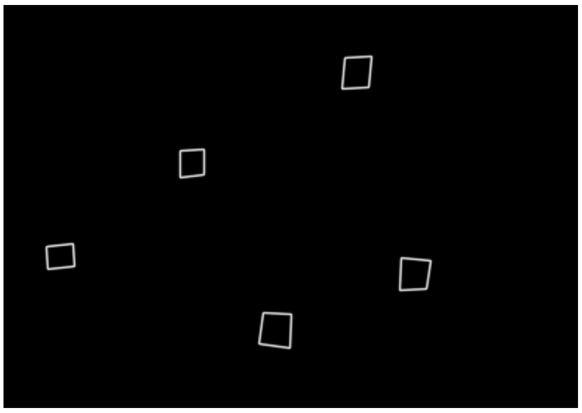
Figure 49: example image with detected rectangles

After the circles or rectangles were detected in the image, sub images are generated based on the detected shape's position and size. This image is then thresholded (using Otsu's method [38] to generate the threshold value) and supplied to the template matching.

The template matching is done with the generated sub image and a template image. The latter is generated from a pre-defined list of template images and resized to fit the sub image's dimensions. The actual template matching is done via OpenCV's cvMatchTemplate. The function is often used to detect a template inside of a main image. It works by moving the template over the main image and calculating a similarity value for each position. Figure 50 shows an example of template and main image.

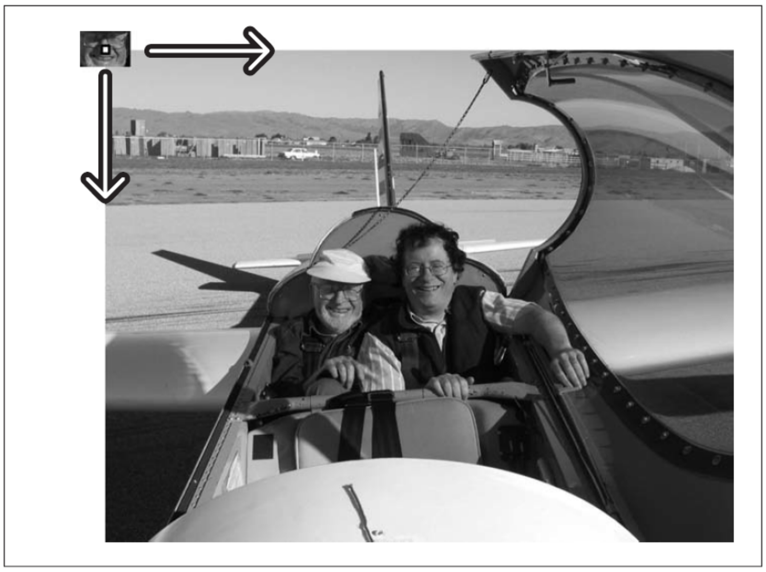
Figure 50: cvMatchTemplate visualization [27]

The same functionality is used in this work to classify the sub image, containing the user drawn symbol. Similar to the way the gesture recognizer Protractor [32] works (as mentioned in chapter 4.3.5), the calculation done by the algorithm provides a similarity measure between template and sub image. Unlike Protractor's scenario the location of the user drawn symbol inside the sub image is not fixed. Depending on the accuracy of the detectors, the exact position can vary. To compensate for this, cvMatchTemplate is used to generate a similarity value for each valid pixel in the sub image. The position with the highest value is then used as the measure of similarity between the template and the user drawn symbol. Figure 51 shows an example of a template matching process between a sub image and a resized template (original size of the sub image 22x24 pixel, original size of the template 15x15 pixel). The size of the sub image is marked with a dotted line. Note that the outer white pixel in the sub image belong to the user drawn rectangle used to mark the symbol.

Several options are given for the calculation of the similarity value. The options are: Squared difference (used by Protractor), cross correlation and correlation coefficient. While a squared difference calculation provided good results in Protractor, an additional complication is introduced in this work's setup: The sub image, containing the user drawn symbol, can contain other

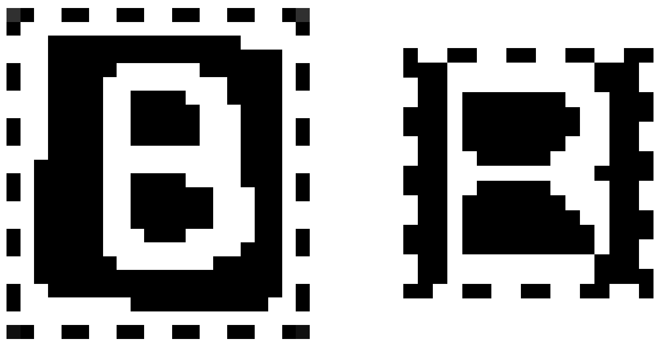
Figure 51: Sub image with drawn symbol(left) and resized template(right). Borders were added manually

elements of the drawing, like elements of the circle or rectangle used to mark the symbol, due to inaccuracy of the detectors. If pixel belonging to these structures are used to calculate the squared difference value, the results are unusable for classification.

The squared difference R_{sq_diff} of the sub image I and the template T at position x, y is calculated by:

$$R_{sq\_diff}(x, y) = \sum_{x',y'} [T(x', y') - I(x + x', y + y')]^2$$

For this reason cross correlation was chosen to calculate the similarity value. Because the value is calculated as a product between sub image and template image and not as a difference, it is possible to eliminate pixel from the calculation, if the template image contains black pixel at the appropriate positions. If, for example, the template and sub image look like the one in figure 51, only those positions, where white pixel are matched against each other, are influencing the similarity value. All other matches are eliminated, as one of the factors for their calculation is zero.

The cross correlation R_{ccorr} of the sub image I and the template T at position x, y is calculated by:

$$R_{ccorr}(x, y) = \sum_{x',y'} [T(x', y') \cdot I(x + x', y + y')]^2$$

Using standard cross correlation would produce higher values for template images with a high amount of white pixel though. As a consequence results would no longer be comparable between different templates. To compensate for this, the normalized cross correlation method is used. The normalized method is typically used to compensate for brightness differences between sub image and template. Likewise it provides normed values for this context, as the cross correlation value is divided by the number of white pixel, when using the normalized method. R_{ccorr_normed} is the normalized cross correlation, with $Z(x,y)$ being the normalization coefficient used.

$$Z(x,y) = \sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}$$

$$R_{ccorr\_normed}(x,y) = \frac{R_{ccorr}(x,y)}{Z(x,y)}$$

The results of all template matches are gathered. If the highest value is above a threshold, the symbol is classified according to the highest ranking template and a symbol object is generated that contains position and type of the symbol.

The list of detected symbols is then send to the updateSymbols method, which uses a list of all previously detected symbols and makes sure that no symbol is send twice to the agent simulation.

Up to this points the detected and classified symbols are still part of the original image and would register as walls, when the dungeon creation runs on the rectified image. To eliminate the symbols in the rectified image, another IplImage is created that represents the area covered by the symbols and their corresponding marking shape. The necessary information (like bounding box size and position) is contained in the symbol object. This image is then used similar to the outlier image, to check if a position in the rectified image should be considered, when creating the dungeon wall structure. The information could also be written into the outlier image, though they were kept separate for debug purposes in this work.

Figure 52 shows an example of the dungeon map image with and without the symbol shapes.

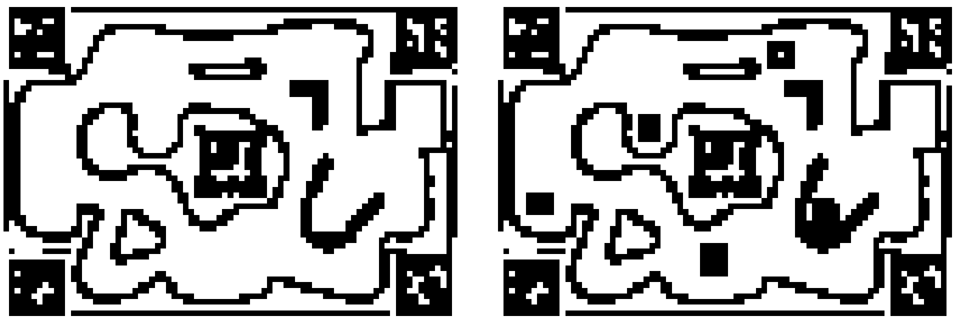

Figure 52: Dungeon map with and without eliminated symbols

6 Evaluation

This section evaluates the system components and explains the used test scenarios. A performance evaluation of the system on the target hardware is discussed in its own subsection.

6.1 Pose tracking

The evaluation of the pose tracking part of the system focuses on the accuracy of the used solution. In addition encountered problems are discussed.

6.1.1 Accuracy

A high degree of accuracy is the main requirement for the pose tracking. As expected ARToolkitPlus shows good results in the conducted tests, with one exception (see multimarker tracking below).

To test pose estimation accuracy, test objects in the form of colored cubes were rendered at the position of the markers specified in the multimarker configuration file. If a correct pose is used, the cubes are rendered exactly on top of the marker centers. This was tested from different positions and angles with a sheet containing a dungeon drawing and the markers according to the specification in section 5.5.1. See figure 53 for an example.

In addition, the warped images that are created during the sketch recognition process, were evaluated. If a correct pose is used, the middle marker is in the center of the rectified image. This was also tested from different positions and angles with the test drawing mentioned above.

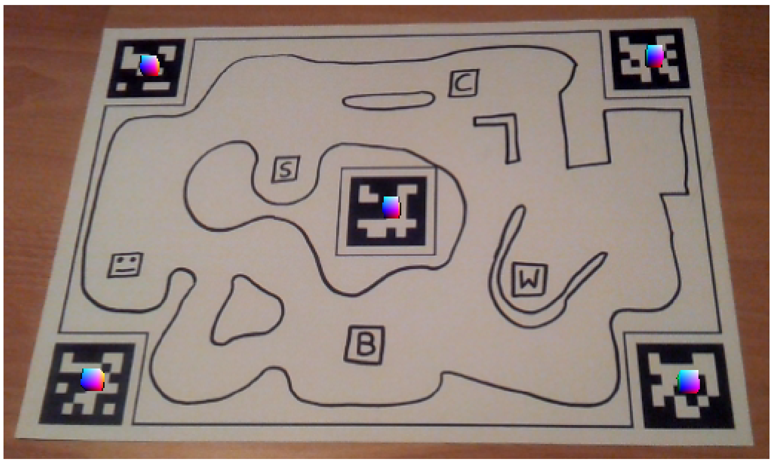

Figure 53: Pose tracking test

The following is valid for two cases: When only one marker is used, or when all five markers of the setup are visible in the camera image. This is because a problem with the multimarker tracking of ARTK+ was encountered during the evaluation. This will be explained below.

Even though no calibration data for the specific camera of the Samsung Galaxy S2 was provided the results are satisfactory. This suggests that the default calibration file of ARTK+ is suitable for use with smartphone cameras, though further tests with additional camera models would have to be conducted for a final result in this regard.

The conducted tests have shown that the rendering based on the pose data provided by ARTK+ is sufficiently accurate. The rendered cubes are on the marker centers and no deviation was visible . This is valid for a distance of less than ca. 70cm and an angle of more than ca. 30 degree between sheet and camera. Depending on the distance and the angle used the accuracy decreases as expected but remains stable for parameters inside the given values.

The camera images were warped with a maximum deviation of about two pixel during the test with 20 program runs. The test included images from different distances and angles, still inside the parameters mentioned above.

6.1.2 Multimarker tracking

A problem with the multi marker tracking of ARTK+ was encountered during the tests. This problem results in erroneous pose data.

Starting with the prepared sheet and all markers visible, the pose is accurate, i.e. the created dungeon rendering is correctly placed on top of the drawing. When markers of the multi marker setup are occluded the pose data contains errors and the dungeon rendering is no longer located exactly on top of the drawing.

This error takes the form of a translation of the occluded marker position in the multi marker setup. For example, occluding the bottom left marker of the setup, results in a translation of the rendered walls close to the occluded marker. Figure 54 shows an example of the multimarker setup with enabled wall rendering and with all markers visible. Figure 55 shows an example of the same scene, where the bottom left marker is occluded. The test series was conducted with a tripod to ensure an identical camera position.

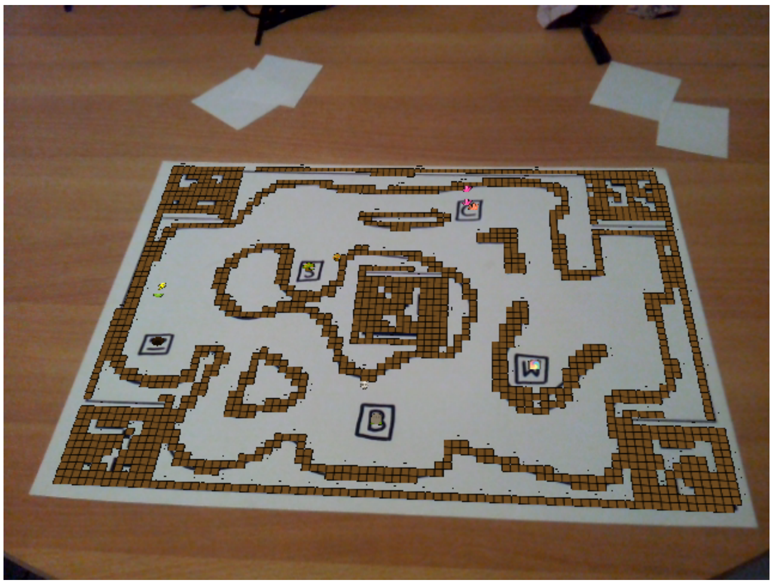

Figure 54: Dungeon wall rendering, created with all 5 markers visible

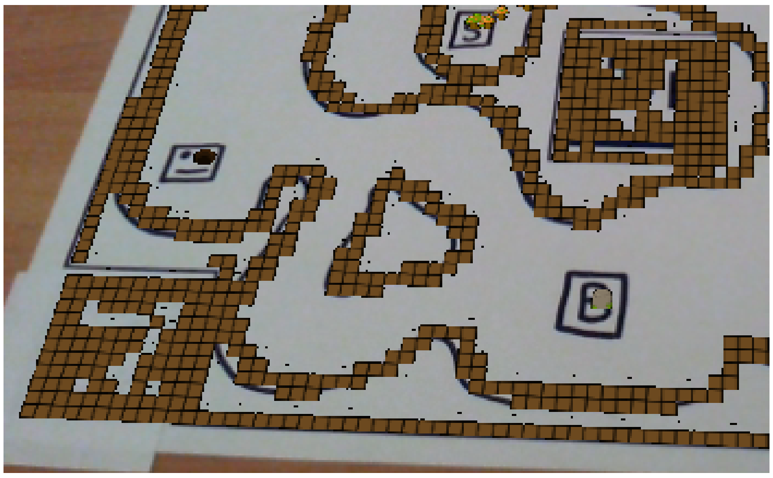

Figure 55: Dungeon wall rendering, bottom left marker occluded

This error occurs for every occluded marker. Consider, for example, the case that the dungeon structure is generated for a camera image that contains all five markers. The camera is moved closer to the sheet afterwards and some markers are no longer visible. This will result in erroneous pose data for the new camera position. Figure 56 shows an example of a case where the initial dungeon structure was generated with all five markers visible and the camera is moved closer afterwards.

The error does not occur when only a single marker is used as figure 57

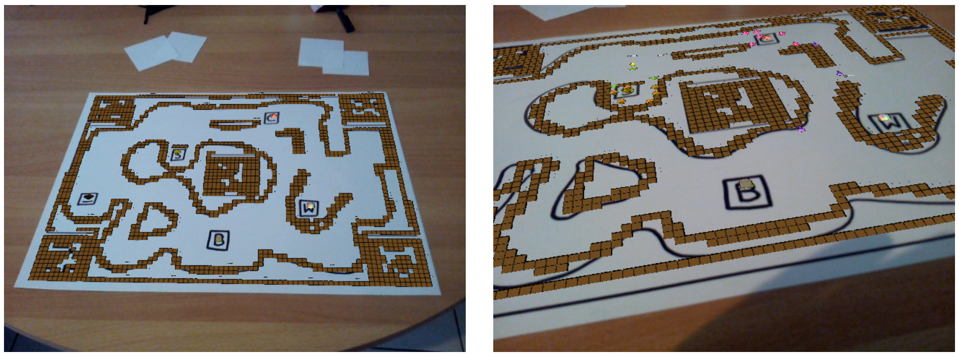

Figure 56: Dungeon structure generated with all five markers visible and closeup from new camera position with only 1 marker visible

shows. In both cases, the initial images were taken with a tripod to ensure an identical camera position for the pose calculation.

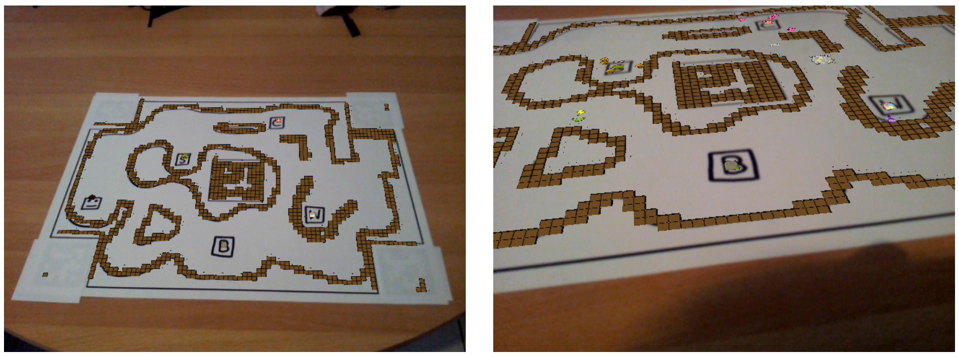

Figure 57: Dungeon structure generated with only one marker visible and closeup from new camera position

Erroneous parameters for the radial distortion parameters in the ARTK+ calibration file might be the cause of the problem. The observed deviations in the dungeon rendering are appropriate for a radial distortion (figure 55). See figure 58 for an illustration of a radial distortion. The plus signs show the position of image coordinates after a radial distortion is applied.

The exact cause for the problem was not determined in the course of this work, due to time constraints.

6.1.3 Optimization of pose tracking and sketch recognition

The combination of fiducial based pose tracking and offline sketch recognition offers a lot of potential for optimization as some of the necessary operations are shared among the two procedures. Image rectification, for

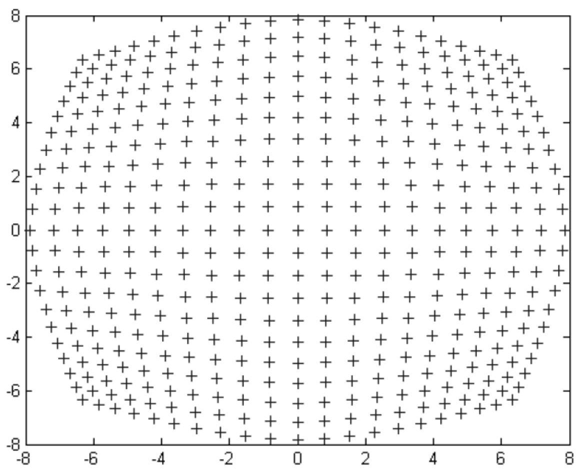

Figure 58: Illustration of radial distortion [41]

example, is done in both cases.

While the pose data generated from ARTK+ was successfully used for the sketch recognition, the idea of using the rectification process of ARTK+ for the sketch recognition was not realizable. The reason for this, is the fact that ARTK+ does not generate a complete rectified image, but only warps the relevant parts of the image (e.g. the corners of the marker). For this reason, the image rectification had to be done in full for the sketch recognition.

6.1.4 Summary

This section summarizes the above and evaluates if the given requirements are met by the pose tracking.

Pose tracking requirements:

- Must be able to provide a stable and accurate AR overlay in real-time on the target system

Result: Satisfied, for single marker tracking. If multi marker tracking is used the rendered scene elements can be slightly misaligned, depending on the number of visible markers.

- Should not obstruct the ability of the user to draw a sketch

Result: Partially satisfied. The fiducials occupy room on the sheet.

6.2 Sketch recognition

The evaluation of the sketch recognition uses a qualitative approach to analyze the system's ability to fulfill the given requirements. It focuses on problem and important boundary cases to show the strong and weak points of the respective techniques.

6.2.1 Dungeon creation

The dungeon creation process was evaluated by analyzing the user sketches drawn at a public demonstration of an early version of the application. While the symbol detection was not in place at that time, the dungeon creation was already in its final stage. The users were given instructions about the way the dungeon has to be drawn, in the form of an example dungeon with added notations. Figure 59 shows the example dungeon. 23 user sketches were evaluated. The main aspects that were pointed out to the users were:

- Minimum distance between dungeon walls had to be about 2cm in order to be passable by agents
- They were told not to draw over the borders on the prepared sheet

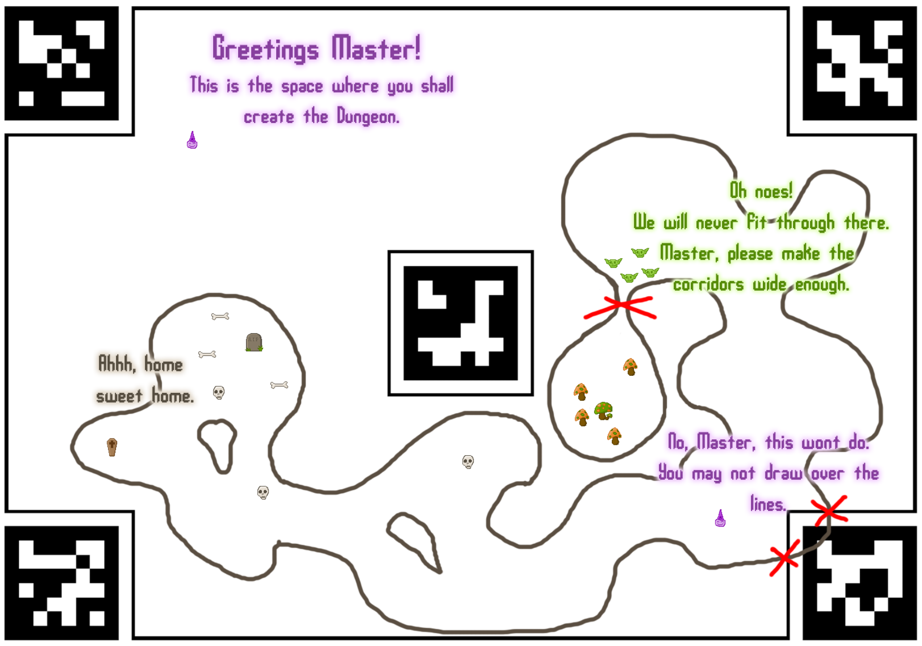
Greetings Master!
This is the space where you shall create the Dungeon.
Oh noes!
We will never fit through there.
Master, please make the corridors wide enough.
Ahhh, home sweet home.
No, Master, this wont do.
You may not draw over the lines.

Figure 59: Example dungeon

The accuracy of the representation is dependent on the image resolution and the tilesize used for the dungeon creation. The values used in the evaluation are: camera image 640x480 pixel, warped to 408x286 and resized to 81x57 for a scale of 25 square mm in the drawing to 1 tile in the dungeon structure.

For a believable augmentation of the drawing, the created dungeon representation has to satisfy several criteria:

- The virtual scene elements representing the dungeon inhabitants must be able to traverse narrow spaces, as expected for their size
- They must not walk on top of or through the drawn lines

The system used for dungeon creation is able to satisfy these criteria, when an accurate pose is provided by the pose estimation. The agents do not cross or walk on top of the drawn lines. This is because of the erosion applied to the camera image. The operation closes gaps in the structure, which is important for the pathfinding of the agent simulation and results in the agents not walking through lines. The increase in line strength is strong enough to also prevent the virtual scene elements from being displayed on top of the lines, even when slight inaccuracies are present in the supplied pose. A similar principle is used in robotics pathfinding by applying the minkowski sum of robot body and environment.

The erosion still allows for the agents to walk through narrow spaces as expected for their displayed size. Even though the room available for agent movement is reduced by the erosion, the reduction is not sufficient to close off spaces equal to or smaller than the agent size. They only take up one tile in the dungeon representation which equals 5mm width and height. Figure 60 shows an example of the created dungeon representation. Note that agents are able to walk diagonally. More information about the agent behavior can be found in [18].

To register as two separate lines the minimum distance between two drawn lines has to be 5 mm, which equals the tilsize. Lines that are closer than 5mm are interpreted as a single line. This is accurate for the augmentation though, as the agents are not supposed to be able to walk between these lines, as they would appear to walk on top of the lines. A change in tilesize and agent size would allow for a smaller minimum line distance.

Like the dungeon creation as a whole, the generation of dungeon parts from partial images is reliant on pose estimation accuracy. If no errors from multimarker tracking occur, the dungeon parts generated from the partial images fit into the overall structure without causing gaps or dead ends. If

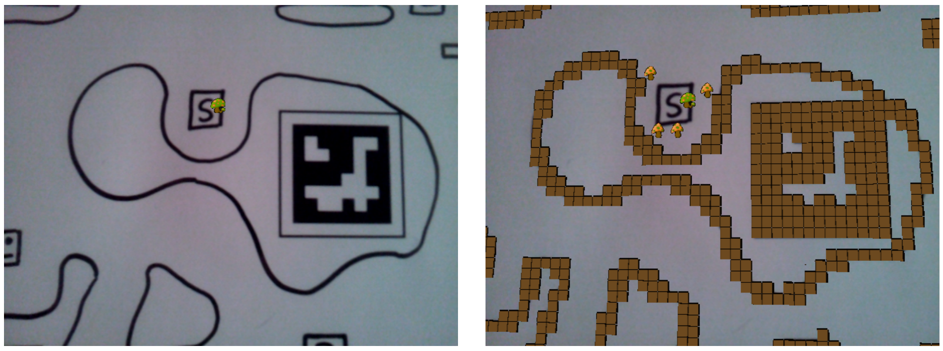
Figure 60: Dungeon representation example

errors are present, the insertion of the dungeon part will cause the drawing and the dungeon representation to be out of alignment in that particular area. The degree of misalignment is dependent on the error of the pose estimation.

A problem that was encountered during the evaluation is connected to the prepared sheet used for drawing. The markers are made up of comparatively big black areas, which influences the adaptive threshold technique used to create the binary image. As a result the border lines used to separate the drawing area from the markers, are not fully represented in the binary image. Figure 61 shows an example. As a fix for this problem, the line strength of the border lines could be increased. This would make sure that they are not filtered out by the adaptive thresholding.

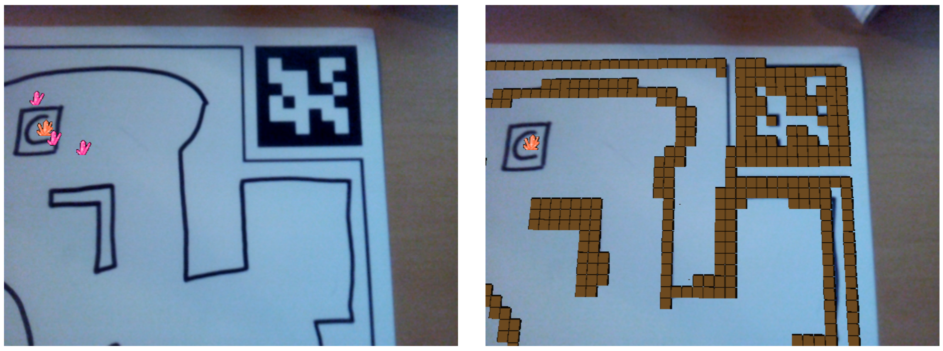
Figure 61: Dungeon representation with hole in wall near marker

6.2.2 Circle based symbol processing

The circle based approach to detect and classify the user drawn symbols has proven to be problematic. The main problem stems from the inaccuracies introduced during the detection of hand drawn circles. The algorithm provides satisfactory results for perfect circles, but hand drawn circles are often more ellipsoid, resulting in incorrect position and radius results.

Correct positions and radii are of great importance for the template matching part, for several reasons: If parts of the drawn symbol are missing due to a wrong position, the template matching is not able to provide correct results. Depending on the sub image generated even a false match could be generated.

To compensate for this, the region of interest used to created the sub image has to be increased in size, which results in additional processing and more pixel in the sub image that do not belong to the user drawn symbol.

Unlike a wrongly reported position, an erroneous radius cannot be corrected and leads to a wrong template size, which in turn leads to incorrect classification. This makes a wrongly reported radius the highest contributor to incorrect classification results.

Providing exact numbers for the cause of a singular problem is difficult as additional elements, like the shape of the provided templates and the quality of the user drawing, have to be considered. For evaluation purposes test drawings for 5 template types were created, each with multiple hand drawn circles, ranging from more regular to ellipsoid structure. Figure 62 shows examples from a circle test drawing.

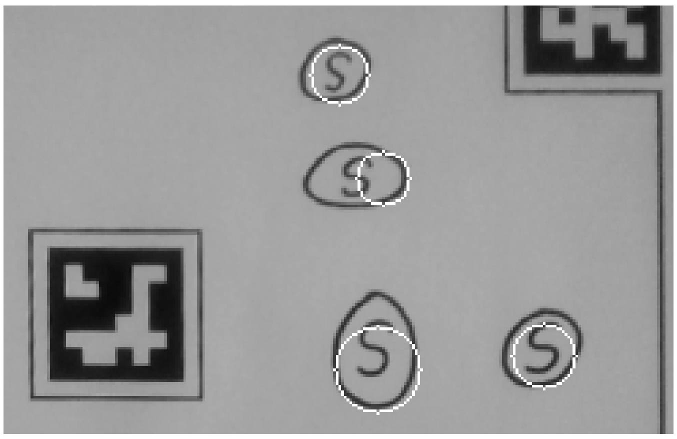

Figure 62: Test circle drawings for the S template, detected circles shown in white

Tests with this fixed setup have shown that a deviation of more than 15% in radius makes a correct classification nearly impossible. The only true positives in this context resulted from erroneous circle data: In these cases high similarity values resulted from pixel that did not belong to the symbol itself, but were part of the sub image due to the increased radius of the region of interest. Figure 63 shows an example of a sub image generated from wrong circle data.

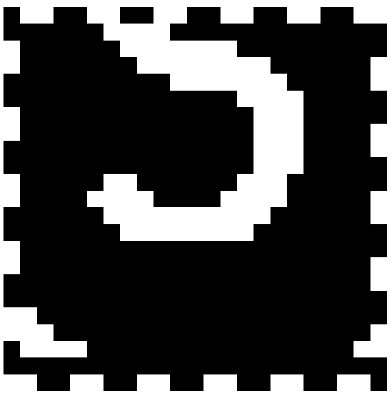
Figure 63: Sub image with S symbol from erroneous circle data

Another problem arises from the irregular nature of the hand drawn circles: The parameters for cvHoughCircles have to be more lenient in order to not generate false negatives. This also results in more false positives, as the detector picks up ellipsoid or circle-like parts of the drawing. Figure 64 shows an example. The sub images generated from these false positives have to be filtered out by the template matching. A threshold value of 0.6 for the normalized cross correlation was determined empirically, by analyzing the values of test drawings that contained ellipsoid and circle like elements, in addition to the symbols. With this threshold no false positives were detected as symbols and the user drawn symbols were still correctly classified, if the circle detection provided sufficiently accurate data.

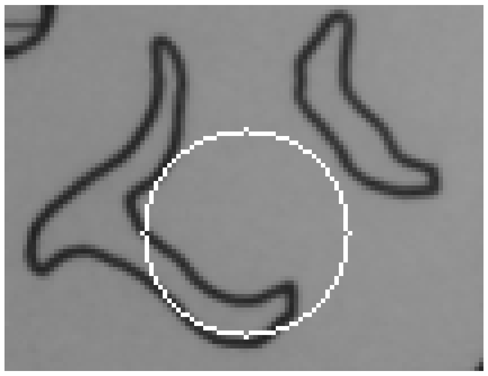
Figure 64: False positive from circle detection

The inconsistency of the results of cvHoughCircles, between different takes of the same drawing, is also problematic. Even if the pose estimation provides satisfactory data, the algorithm can detect different circle positions and radii for the same structure. This problem is more common for irregular circles. The position and radius of perfect circles varied little between images and was not strong enough to influence the classification. Figure 65 shows another example of the circle test drawing. To compensate for the inconsistency, the circle detection is run a second time(with a new image and pose matrix pair). This improves the final results, as wrong circle detection data is filtered out by the classification process. Additional runs of the circle detection would further improve the results at a cost of additional processing time. Letting the detection run twice was found to increase the chance of detecting all user drawn circles to about 85%. (Determined in tests using the circle test drawings over a series of 20 program runs.)

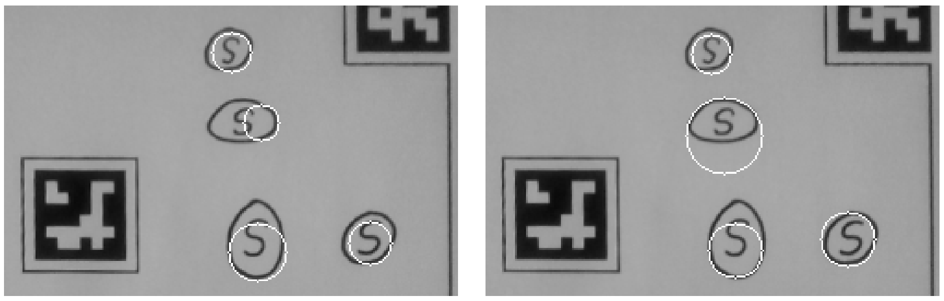

Figure 65: Different results of cvHoughCircles

6.2.3 Rectangle based symbol processing

Similar to the previous method, irregular structures are a boundary case for the rectangle based approach. Non-square rectangles are one of the irregular structures that can be found in the user drawings. Classification of structures in a non square rectangle works well though, as the chosen approach searches the whole rectangle for a match. In this case the template image is resized based on the biggest enclosed square. Figure 66 shows an example of a non square symbol area.

The rectangle detection works well for axis aligned rectangles. Skewed rectangles are problematic in that the application generates the sub image based on an axis aligned bounding box. OpenCV does not support the creation of sub images based on non axis aligned bounding boxes, so a manual rotation and masking of the original rectangle would have to be implemented for this feature. If the user would draw non axis aligned rectangles, the symbol classification of the current system could fail. This only

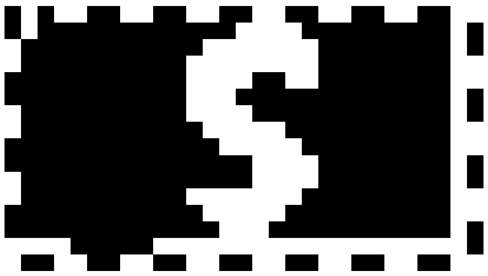
Figure 66: Non square symbol area

applies to a skew of more than approximately 10-15 degree, as smaller deviations are still processed correctly by the classification.

Due to the tests that are conducted to verify a curve as a rectangle, the detection only supports correct rectangles. That means no overdrawn lines are permitted as they would introduce another element in the cvSeq object. Figure 67 shows an example of a boundary case that is still identified as a rectangle.Further increasing the length of the overdrawn line would result in a not-detected rectangle. Non-closed rectangles are not detected as well, but this is not considered an error and can be easily corrected by the user.

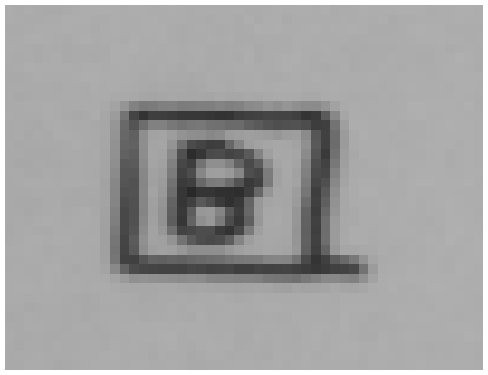
Figure 67: Boundary case for rectangle detection

The filtering of other rectangular structures, also works well. For example the used parameters prevent the markers on the prepared sheet from generating positives. Other rectangles in the drawing that pass the necessary tests and do not mark a symbol, are filtered by the classification process. As the rectangle detection offers precise location and size information, only a small amount of boundary pixel are supplied to the template matching. The term boundary pixel refers to pixel that do not belong to the actual symbol, but to other shapes in the surrounding area, like the one used to mark the symbol's location.

A special case that causes problems is a small rectangular structure in the drawing (like a small room of the dungeon) that is detected as a proper rectangle and that itself contains a new symbol marked with a rectangle. Figure 68 shows an example. Here, the outer rectangle could be picked up as the symbol's rectangle, resulting in slightly different values. Note that the relevance of this special case depends on the exact ratio of the outer and inner rectangles' dimensions. If the sizes are similar, the classification would still be correct. If the sizes are too far apart the classification would reject the outer rectangle.

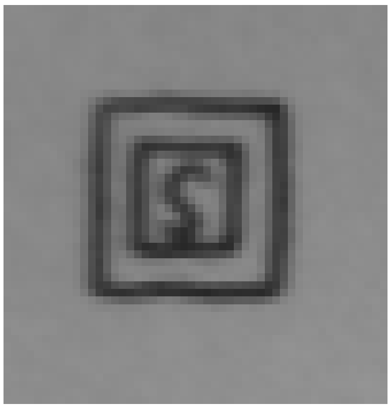

Figure 68: Boundary case for rectangle detection

Detection can fail, if the rectangles are drawn close to other lines in the drawing. In an earlier version of the system, a dilation was applied to the edge image, strengthening and merging the edge lines. This was done, because the lines of a rectangle could otherwise be picked up as two separate curves. In some cases, this also merged the rectangle lines with the shape of a neighboring wall line. As a consequence the rectangle was not detected, as the resulting merged curve had more than four elements. The dilation was cut from the rectangle detection process, which resulted in an increase of false positives (this was compensated with a filtering step, as mentioned in chapter 5.5.4). The minimum distance between rectangles and other lines is about 3mm (corresponds to the kernel size of the Canny algorithm used, which is 3 pixel).

The minimum rectangle area is about 1 square cm, which is close to the minimum drawing size for the given pen thickness.

6.2.4 Rectangle based vs Circle based

The rectangle detection provides good sub images for the template matching and better results than the circle based approach. The sub images hold little boundary pixel as the rectangle detection offers precise results. This is in contrast to the circle detection, where a larger region of interest had to be selected to accommodate for the position error.

Correct rectangles are easier to draw by hand than correct circles, thus the detection of the structure is more accurate. The rectangle detection also produces less false positives (none at all with the current settings and the test drawings) than the circle detection. This results in less template matching procedures and allows for a lower threshold value for the classification, making it less likely that a false negative will occur.

The rectangle based approach clearly offers superior results and is chosen for the final application. The circle based approach suffers from the difficulty of drawing correct circles, from which most of its problems arise. The circle based detection could be easily be used with pre-printed circles though. This offers an alternative for a different scenario, if a drawing contains too many rectangular structures. In this case the circle detection would return accurate information.

6.2.5 Template matching and classification

To evaluate the template matching and classification process, different problem sources that were identified during the tests, were analyzed. In addition different templates types were evaluated, to determine the influence template appearance has on the classification result.

Results have shown that the classification is strongly reliant on the quality of the generated sub image. Symbols that are only partially visible in the sub image, or sub images that contain too many boundary pixel, have adverse effects on the classification and usually lead to incorrect results. This problem occurs more often when using the circle based detection, because of the inaccuracy of reported circle position and radius mentioned above. Figure 69 shows an example of a sub image that was generated from incorrect circle data and the resized template used for matching.

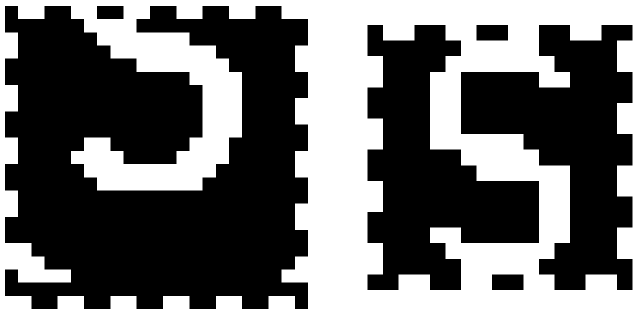

Figure 69: Problematic sub image, generated from erroneous circle data and resized template

If the rectangle based approach is used, the problem arises less often, as it gives more accurate results. In fact the sub images generated from the rectangle detection are correctly resized unless a special case ocurres: A big difference in size of marking rectangle and actual symbol drawing. Figure 70 shows an example.

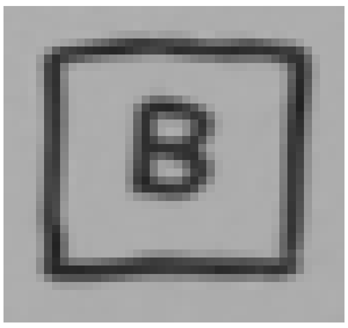

Figure 70: Example of deviating rectangle and drawn symbol size, classified as W-template

The sub image supplied to the template matching is based on the marking rectangle's dimensions. According to this sub image the predefined template is resized. The process assumes a fixed ratio of sub image size and drawn symbol size (A ratio of 1.2:1 was determined empirically). A difference in size between drawn symbol and resized template, larger than 15% can result in incorrect classification. Small deviations from the default ratio still result in a correct classification and large deviations (more than ca. 50%), are rejected as they don't pass the threshold value. The example in figure 70 shows a difference that leads to a wrong classification.

In the conducted test drawings, this was usually not a problem as only small deviations occurred, though more data from a user test would be needed to evaluate the significance of this problem. A solution for this would be to create multiple template resizes and run the template matching with each one.

Test templates were generated from uppercase letters of the Sans true type font, a sans-serif font, and saved as 15x15 pixel images. In addition non-letter templates were tested.

The chosen similarity calculation with normalized cross correlation works well for templates with different numbers of pixel. Figures 71 and 72 show examples of a correct template match. The B and Face templates have a big difference in pixel quantity, the image shows the generated sub image (left) and the resized template (right) along with the respective highest normalized cross correlation values.

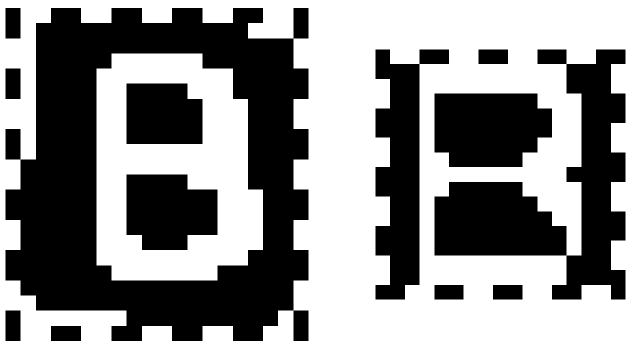

Figure 71: Classifcation example, highest $R_{ccorr\_normed}(x, y)$: 0.76 (B template)

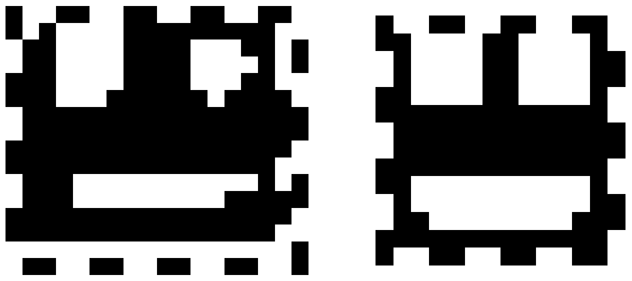

Figure 72: Classifcation example, highest $R_{ccorr\_normed}(x, y)$: 0.73 (Face template)

The calculation method also helps to eliminate false positives. Figure 73 shows an example of a false positive and its value. The example is taken from the circle based approach, as the rectangle based approach doesn't produce false positives apart from the case mentioned above. When the special case occurs, the classification will likely return an erroneous result, though the exact outcome is dependent on the difference in ratio and the list of supplied templates. A difference in size between sub image and resized template of more than 50% for example, will have too many black pixel in the sub image, resulting in a normalized cross correlation value that does not exceed the threshold.

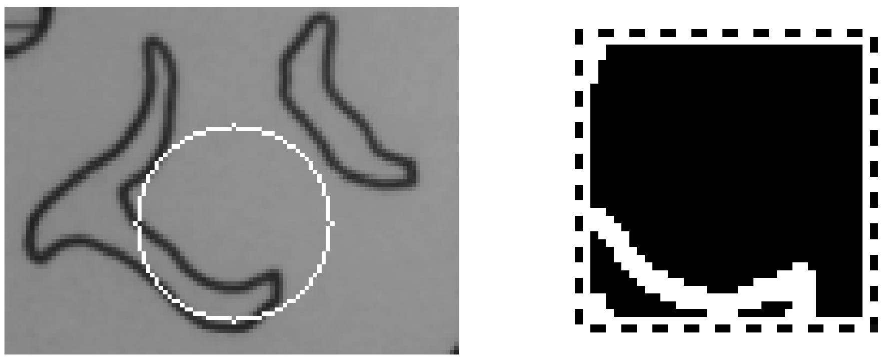

Figure 73: Rejected false positive from circle detection with generated sub image. Highest value, S template: 0.48

To improve classification results, the templates need to be as distinct from each other as possible. Templates that are similar in appearance will produce similar values, making a correct classification more difficult. Another way to improve classification results is to utilize additional templates for a single symbol type. In this way different writing styles can be taken into account. Figure 74 shows an example of two different S templates.

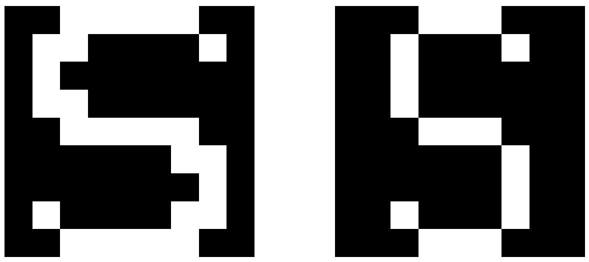

Figure 74: Example of different S templates

Image blur is a problem originating from the chosen setup. If images contain blur from a shaking motion during image capture the classification will fail, as the template matching relies on details that will be lost if an image blur is applied.

6.2.6 Sketch recognition summary

This section evaluates if the given requirements are met by the sketch recognition.

Sketch recognition requirements:

- Must be able to detect the dungeon and create an accurate virtual representation of it that is suitable for the agent simulation

Result: Satisfied. The discretized representation is suitable for the agent simulation. The level of accuracy is satisfactory for the given tilesize and agent size.

- Must be able to accurately detect drawn symbols.

Result: Satisfied for rectangle based approach, when no strongly skewed rectangles are used.

- Must be able to correctly classify the detected symbols with good accuracy using a pre-defined list of symbol definitions.

Result: Satisfied under conditions. The Sketch recognition generates a correct symbol classification, when:

- Rectangles are accurately detected
- Captured image is not strongly blurred
- No big difference in size between rectangle and drawn symbol
- User drawing is close in appearance to the intended template

6.3 Performance

The performance evaluation focuses on the general feasibility of the system on a smartphone platform. Special consideration is given to the question of real-time capability for the sketch recognition part.

Performance was measured with the Traceview program, the standard profiling tool for the Android platform. Due to the additional work done during profiling, the reported timing data is not perfectly accurate. The actual performance should be slightly faster, though not to a significant degree.

All timings are reported for the prototype system and further optimizations could likely speed up the system.

The evaluation was done on a Samsung Galaxy S2 with a 1.2ghz dual-core ARM Cortex-A9 processor.

6.3.1 Pose tracking performance

The pose tracking consists of two major parts: the color space conversion to RGB format and the actual pose estimation of ARToolkitPlus.

The conversion from YCbCr to RGB is done via a native method and takes about 15ms. The pose estimation via ARTK+ takes 14 ms. With an overall time of about 29ms the pose tracking part is able to match the 30 frames per second provided by the camera. Hence, the system is able to provide real time pose tracking with 30 fps. The pose tracking thread is set to high priority and takes up most of the resources of the first core of the system.

6.3.2 Sketch recognition performance

The evaluation examines the different parts of the sketch recognition process and later gives a summary of the overall process.

Dungeon creation
The dungeon creation process consists of several parts: The image rectification and the construction of the actual dungeon representation.

The image rectification takes about 170ms. The method cvWarpPerspective takes up 80ms and is by far the most expensive operation during the process. The method applies the homography and creates the rectified image and the outlier image. The method is highly resolution dependent and the supplied images are sized 640x480 pixel.

The creation of the dungeon representation takes about 816ms. This is because of the non-native implementations for the new adaptive threshold method and the updateDungeon method. Both frequently interact with the native IplImage structures and in this way accumulate a high workload(550ms for the adaptiveThreshold and 266ms for updateDungeon). For comparison: The native implementation of the original cvAdaptiveThreshold method takes about 2ms for the same image. A native implementation of the new adaptive Threshold algorithm should be as fast as the cvAdaptiveThreshold in the worst case or slightly faster otherwise. With native implementations the creation of the dungeon representation should take between 2 and 10ms.

Overall the dungeon creation process takes about 1 second which could be reduced to about 180ms by implementing the native methods.

Circle based symbol processing
Circle based symbol processing consist of the circle detection and the template matching process.

The circle detection takes about 56ms with cvHoughCircles being the most expensive operation with 53ms. The template matching takes about 86ms. The most expensive operation of the template matching process is cvMatchTemplates with 21ms (a much higher value than in the rectangle based approach, due to the higher number of false positives).

Overall the processing time for the circle based symbol processing is about 142ms for a single run with 6 circles and a dungeon structure visible in the camera image. 7 pre-defined templates were used in the template matching. Note that the circle based approach requires a second run of the circle detection part and additional template matching, which increases the overall time to more than 200ms.

Rectangle based symbol processing
Similar to the circle based approach the main processes of the rectangle based symbol processing are made up of rectangle detection and template matching.

The rectangle detection takes about 34ms with cvCanny being the most expensive operation with 12ms, followed by cvFindContours with 6ms. The template matching takes about 32ms. The method cvMatchTemplates only takes about 4ms in this approach.

Overall the rectangle based symbol processing takes about 66ms for a camera image with 6 rectangles and a dungeon structure. Again, 7 pre-defined templates were used for template matching.

Sketch recognition performance summary
The processing time of the complete sketch recognition process is about 882ms which could be reduced to about 246ms when implementing the

native methods. This applies to the use of the rectangle based approach. Results have shown that the circle based approach is slower in addition to offering less precise results.

6.4 Performance summary

The prototype system offers real-time pose tracking with 30 fps and and can create a representation of the user drawn sketch in about 1 second. Native implementations could reduce the time for the latter step to about 246ms.

With the current implementation the dungeon creation is triggered manually, which leaves the second core of the smartphone available for other purposes like the agent simulation. With the added native implementations a real-time sketch recognition would be possible. A sketch recognition running with 1 frame per second would still leave ample resources for other processes. Pose tracking running with 30fps and sketch recognition with 1fps would leave about 754ms of processing time available per second on one core of the smartphone.

The system was not tested on single core hardware, though the available data allows some conclusions. The pose estimation would take up most if not all of the processing time if it were to run with 30 fps to match the camera framerate. A lower framerate for the pose tracking would leave more resources for other applications. A user-triggered sketch recognition would still be possible on a single core device if the native methods are implemented. Real-time sketch recognition would not be available on a single core smartphone, without accepting a lower pose tracking framerate. With 15 fps the pose tracking would take up 450ms. Together with a sketch recognition process running at 1 fps (246ms) this would leave about 304ms per second available for other processes.

6.5 Setup

The question of technical feasibility of the smartphone setup was already examined in the previous sections. This section summarizes the above and examines if the requirements were met. In addition some observations related to the setup's usability are given.

6.5.1 Setup requirements summary

Setup requirements:

- Must provide sufficient information to enable accurate sketch recognition and pose estimation

Result: Satisfied, the camera of the Samsungs Galaxy S2 is sufficient for the given requirement.

- Should be on a mobile platform

Result: Satisfied, users can look at the sheet from different positions using the smartphone and the device itself is mobile.

- Should only require pen and paper to draw the dungeon map

Result: Partly satisfied, the sheet has to have preprinted markers with the current setup, though other options are possible (see prospects, section 7.2)

6.5.2 Usability observations

Even though a dedicated evaluation of the system's usability as a whole is beyond the scope of this work, some observations were made during a public demonstration of the program at the university. The users were asked to draw dungeons and view them through the smartphone. The following observations were made:

- The users had no problem using the smartphone as a ”magic lens”
- Some users had problems with the fact that the marker tracking requires a marker to be fully present in the camera image at all times

When no valid pose is available, the rendering is no longer accurate and a disconnect between real and virtual scene elements happens. This highlights a limitation of the chosen pose tracking method. It mostly occurred, when the user moved the smartphone closer to inspect details of the virtual scene elements. When viewing the sheet as a whole or from a distance of more than 30-40cm this is usually not a problem as it is more likely that a marker will be present in the camera image.

7 Conclusion and prospects

7.1 Conclusion

Augmented reality applications often suffer from a disconnect between real and virtual scene elements because the virtual components lack knowledge about their real-world surroundings.

The topic of this work was to explore new ways in which real and virtual elements of an augmented reality environment can interact with each other, to allow for a higher degree of authenticity of the virtual content.

For this purpose a system was designed and implemented that uses techniques of sketch recognition and pose tracking to recognize user-created drawings and display virtual content on top of the drawings via augmented reality. The virtual scene elements are able to interact with the drawing by means of a representation that is created by the sketch recognition.

The pose tracking component was realized by implementing ARToolkit-Plus in the Android environment. For the sketch recognition component a solution was developed and implemented that combines techniques from online and offline sketch recognition. The application runs on an Android smartphone.

The goal of augmented drawings was successfully realized for the dungeon context that was chosen for the drawings. The agents move in the mixed reality environment without violating the authenticity of the scene by moving through drawn lines etc. The dungeon context offers an interesting environment, with many interactions options, especially when combined with the additional functionality for the virtual scene elements, introduced by the agent simulation by Eiting [17] in this work.

As expected, the accuracy of the pose tracking proved to be crucial to the system as a whole. Apart from a problem with multimarker tracking, mentioned in section 6.1.2, the pose tracking part of the system is precise and thus allows for correct rendering and sketch recognition.

The sketch recognition works successfully in combination with the pose tracking and the rectangle based approach gives correct results for most cases. With additional implementation of native methods the application could run with real-time sketch recognition on the smartphone hardware.

Tracking a sheet of paper with unknown content while allowing partial images of the sheet, is a challenging task and some limitations have to be made to enable accurate tracking. The required level of accuracy clearly demands an optical tracking system and while ARTK+ is accurate enough, the fiducial based tracking still presents some usability concerns, as mentioned in section 6.5.2.

Due to the loose coupling of the components, different options for pose tracking or sketch recognition could easily be added to the system to allow for further research. Additional ideas for this are given in the prospects.

7.2 Prospects

Future work could focus on testing different options for pose tracking. For example a new version of the Qualcomm AR SDK could be tested, which might offer more accurate and faster multimarker tracking than ARTK+. Alternatively, different approaches that focus on tracking the sheet as a whole could be explored. Depending on the context, the limitation of keeping the whole sheet in the camera image might be acceptable for stable pose tracking without fiducials. The combination method mentioned in section 4.2.4 could not be implemented in the time frame of this work, but might be a good solution to the problem.

Many different options could be explored for the sketch recognition part, ranging from different drawing contexts to more advanced sketch recognition procedures. The approach chosen for this work is still basic and could be improved with techniques of machine learning for example. Techniques similar to Oltmans [37] could help to improve the classification process. Further exploration of real-time sketch recognition would be interesting as well, though the combination with the smartphone setup might create usability concerns.

New functionality could be added to the current system to provide more interaction options or new use case scenarios. The system could be modified to use a single marker that is placed on a dungeon drawing without preprinted markers, to enable augmentation without the need for prepared sheets. Support for different paper sizes could easily be added. A pinch to zoom metaphor similar to a camera zoom on smartphones could be used to define the format dynamically and to locate the sheet in the marker plane. The setup could also be expanded to include multiple sheets for drawings. In the dungeon context this could be used to implement multiple height levels or dungeon parts that are linked by the augmentation.

Additional works could focus on the user interaction component of the system. Additional user tests would help judge the authenticity of the augmented drawings and could be used to evaluate different setups. The interaction and usability of augmented reality and drawings is a recent topic [23, 24] and the system created in this work could provide an interesting framework for future research and applications in this field.

References

[1] Wikitude, www.wikitude.org, online September 2011.

[2] Anoto pen system. http://www.anoto.com, online September 2011.

[3] Hitlab washington, artoolkit forum. http://www.hitl.washington.edu/artoolkit/mail-archive/message-thread-00654-Re–Questions-concering-.html, online 2011.

[4] Moby games. http://www.mobygames.com, online September 2011.

[5] Opencv bug ticket #951. https://code.ros.org/trac/opencv/ticket/951, online September 2011.

[6] Opencv, computer-vision programming library. http://opencv.willowgarage.com, online September 2011.

[7] Wizards of the coast, dungens & dragons homepage. http://www.wizards.com/DND, online September 2011.

[8] ALVARADO, C., AND DAVIS, R. SketchREAD : A Multi-Domain Sketch Recognition Engine. *Knowledge: Creation, Diffusion, Utilization 6*, 2 (2004).

[9] ANTHONY, L., AND WOBBROCK, J. O. A Lightweight Multistroke Recognizer for User Interface Prototypes. *Proceedings of the 20th annual ACM symposium on User interface software and technology - UIST '07* (2010), 245–252.

[10] APTE, A., VO, V., DAN, T., AND SCIENCE, P. T. Recognizing Multistroke Geometric Shapes. *Computing* (1993), 121–128.

[11] AUDET, S. Javacv. http://code.google.com/p/javacv/, online September 2011.

[12] BAY, H., TUYTELAARS, T., AND GOOL, L. V. SURF : Speeded Up Robust Features. *9th European Conference on Computer Vision Volume:110, Issue:3* (2008).

[13] BULLFROG. Dungeon Keeper, mobygames archive page, online September 2011.

[14] CANNY, J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 8* (1986), 679–714.

[15] DOUGLAS, D., AND PEUCKER, T. Algorithms for the reduction of the number of points required for represent a digitized line or its caricature. *Canadian Cartographer 10* (1973), 112–122.

[16] DRUMMOND, T., AND CIPOLLA, R. Real-time visual tracking of complex structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence vol. 27* (July 2002), 932–946.

[17] DYSON. A character for every rpg, rpg blog, online September 2011. http://rpgcharacters.wordpress.com/maps.

[18] EITING, A. Poplulating a dungeon - autonomous agents in a dynamic user created enviroment, 2011. University Koblenz-Landau.

[19] FAUGERAS, O. *Three-dimensional computer vision: a geometric viewpoint*. MIT Press, Cambridge, MA, USA, 1993.

[20] GOOGLE. Android Gesture Library, http://developer.android.com/resources/articles/gestures.html, online September 2011.

[21] GOOGLE. official google blog, online September 2011. http://googleblog.blogspot.com/2011/05/android-momentum-mobile-and-more-at.html.

[22] GYGAX, G. Ad&d adventure module: Tomb of horrors, 1978.

[23] HAGBI, N., BERGIG, O., EL-SANA, J., AND BILLINGHURST, M. Shape Recognition and Pose Estimation for Mobile Augmented Reality. *IEEE transactions on visualization and computer graphics* (Oct. 2010), 65–71.

[24] HAGBI, N., GRASSET, R., BERGIG, O., BILLINGHURST, M., AND EL-SANA, J. In-Place Sketching for content authoring in Augmented Reality games. *2010 IEEE Virtual Reality Conference (VR)* (Mar. 2010), 91–94.

[25] HAMMOND, T. A. LADDER : A Perceptually-based Language to Simplify Sketch Recognition User Interface Development, MIT PhD Thesis, 2007.

[26] HARRIS, C. Tracking with Rigid Objects. *Active Vision* (1993).

[27] KAEHLER, G. B. . A. Learning opencv.

[28] KARA, L. B., AND STAHOVICH, T. F. An Image-Based Trainable Symbol Recognizer for Sketch-Based Interfaces. *AAAI Fall Symposium* (2004).

[29] KATO, H., AND BILLINGHURST, M. Marker tracking and HMD calibration for a video-based augmented reality conferencing system. *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, 85–94.

[30] KIMME, BALLARD, S. Finding circles by an array of accumulators. *Communications of the Association for Computing Machinery 18* (1975), 120–122.

[31] LEPETIT, V., AND FUA, P. Monocular Model-Based 3D Tracking of Rigid Objects : A Survey. *Foundations and Trends in Computer Graphics and Vision 1*, 1 (2005), 1–89.

[32] LI, Y., AND VIEW, M. Protractor : A Fast and Accurate Gesture Recognizer. *Design* (2010), 2169–2172.

[33] LILJEQUIST, B. Planes , Homographies and Augmented Reality, 2003.

[34] MILGRAM, P. Augmented Reality: A class of displays on the reality-virtuality continuum. *Telemanipulator and Telepresence Technologies 2351* (1994), 282–292.

[35] NOTOWIDIGDO, M., AND MILLER, R. C. Off-line Sketch Interpretation. *Artificial Intelligence* (2004).

[36] NOVARAMA. Invizimals. www.invizimals.com, online September 2011.

[37] OLTMANS, M. Envisioning Sketch Recognition : A Local Feature Based Approach to Recognizing Informal Sketches, Doctoral Dissertation at MIT, 2007.

[38] OTSU, N. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man and Cybernetics 9*, 1 (Jan. 1979), 62–66.

[39] PAGE, G. F. Multiple view geometry in computer vision, by richard hartley and andrew zisserman, cup, cambridge, uk, 2003, vi+560 pp., isbn 0-521-54051-8. (paperback £44.95). *Robotica 23* (March 2005), 271–271.

[40] PAULSON, B., AND HAMMOND, T. PaleoSketch : Accurate Primitive Sketch Recognition and Beautification. *Architecture* (2008), 1–10.

[41] PAULUS, D. lecture notes "Struktur aus Bewegung", 2007.

[42] QUALCOMM. Qualcomm ar sdk. https://ar.qualcomm.at/qdevnet/sdk, online September 2011.

[43] RAJAN, P., AND HAMMOND, T. From Paper to Machine : Extracting Strokes from Images for use in Sketch Recognition. *Interfaces* (2008).

[44] REITMAYR, G., EADE, E., AND DRUMMOND, T. Localisation and Interaction for Augmented Maps. *Fourth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'05)* (2005), 120–129.

[45] RUBINE, D. Specifying gestures by example. *ACM SIGGRAPH Computer Graphics 25*, 4 (July 1991), 329–337.

[46] SKRYPNYK, I., AND LOWE, D. Scene Modelling, Recognition and Tracking with Invariant Image Features. *Third IEEE and ACM International Symposium on Mixed and Augmented Reality* (2004), 110–119.

[47] SUTHERLAND, I. E. SKETCHPAD A MAN-MACHINE GRAPHICAL COMMUNICATION SYSTEM *. *Computer 23* (1963).

[48] TSAI, R. Y. Radiometry. Jones and Bartlett Publishers, Inc., 1992, ch. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses, pp. 221–244.

[49] VISUAL, Q. Word lens, 2010. www.questvisual.com.

[50] WAGNER, D., LANGLOTZ, T., AND SCHMALSTIEG, D. Robust and unobtrusive marker tracking on mobile phones. *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality* (Sept. 2008), 121–124.

[51] WOBBROCK, J. O., WILSON, A. D., AND LI, Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. *Proceedings of the 20th annual ACM symposium on User interface software and technology - UIST '07* (2007), 159.