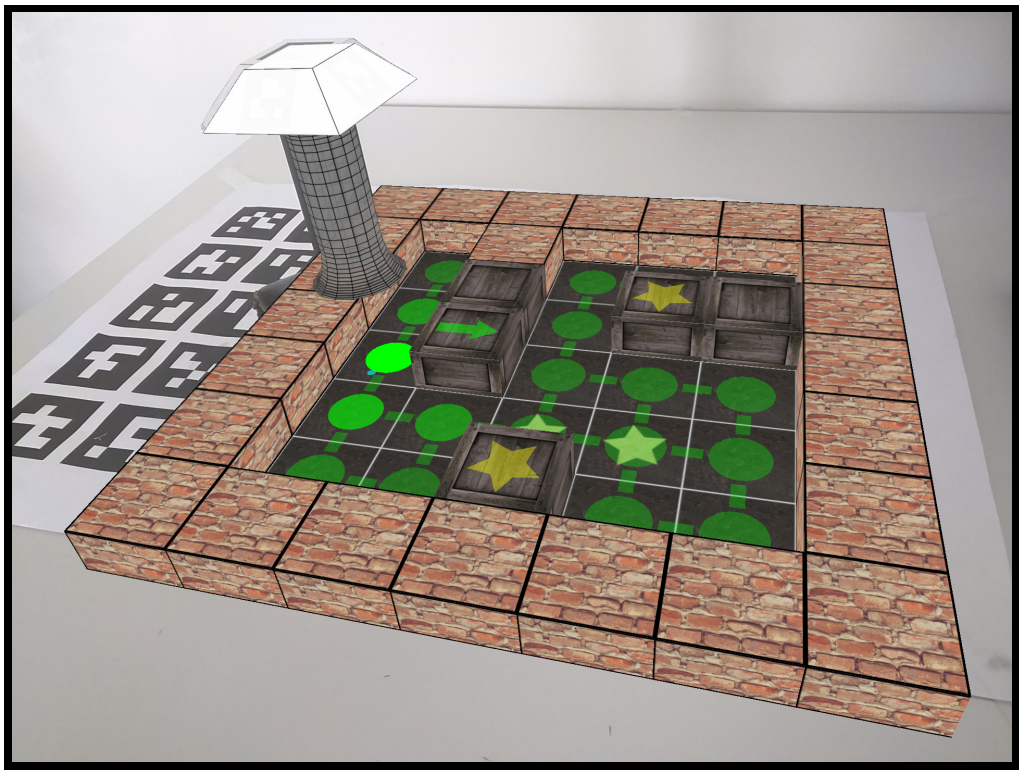


Physical Sokoban

An augmented reality game

Master's thesis in Computer Science: Algorithms, Languages and Logic

CHRISTOFFER ÖJELING

Department of Computer Science and Engineering

Chalmers University of Technology

Gothenburg, Sweden 2016

Physical Sokoban: An augmented reality game

This project was carried out at Know-Center GmbH, Graz, Austria, under supervision of Viktoria Pammer-Schindler. Know-Center is a research company focusing on data science.

At Chalmers Graham Kemp was the examiner and K V S Prasad the supervisor.

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Gothenburg
Sweden
Telephone: +46 (0)31-772 1000

Cover: The Sokoban virtual game objects overlaid on top of the camera video stream. Here the user is performing a move.

Abstract

An augmented reality version of the classical puzzle game Sokoban was developed. The user wears a virtual reality headset on their head. The game objects are virtually drawn on top of the video stream, and the game controlled entirely by moving a physical object. This thesis addresses the algorithmic and design requirements of such a game on low cost and performance constrained devices like smartphones. To this end a new algorithm for accurately tracking the position of the physical object in the video stream was developed.

While the game is designed to be enjoyable by healthy users, it is made to in the future be used in rehabilitation — to aid patients with upper spinal cord injuries training grasping with their neuroprosthesis. To this end the game records various statistics about how the user plays — which in the future can be used to track and analyze patient progress.

A user study carried out on healthy users indicates that the concept shows promise. Out of 10 participants, 9 answered yes to "I would like to play the game again", rating their overall experience a mean of 7.2 out of 10 (higher is better).

Acknowledgments

I first wish to extend my gratitude to Granit Luzhnica, whom I have worked closely with throughout the entire project, including co-authoring a paper. Eduardo Vaes for help with the marker detection benchmarks, paper writing and user studies. My supervisor Viktoria Pammer-Schindler for writing support with both the thesis and paper, the initial idea of the game and the first user study. Chalmers supervisor K V S Prasad who has given me great flexibility in my thesis work. Furthermore I wish to thank all my colleagues at Know-Center for providing a very enjoyable environment to work in. In particular Jörg Simon who has been responsible for ordering material and given Android support.

Contents

1. Introduction

The European Union funded project MoreGrasp is currently researching a brain-computer interface (BCI) to allow patients with upper spinal cord injuries to regain some ability to grasp with their hands. It uses a cap which analyses brain patterns and classifies them into commands, such as "perform grasp". A neuroprosthesis then stimulates the relevant muscles with electrical impulses.

Since each brain is unique, the neuroprosthesis must be trained to recognize each patient's individual brain pattern. This is done by grasping objects over and over. This process is very long and tedious, making it hard to motivate the patient to perform the required training. It is especially true as many patients have not only motor but also cognitive impairments [1].

In this thesis an augmented reality game which incorporates a lot of grasping is developed. Augmented reality is a mixture of the real and digital world, where virtual objects are overlaid on top the real world when viewed through a screen. The game developed is a version of the classical game Sokoban, a puzzle game where you push boxes on a 2-dimensional grid. The game is played by grasping a physical object and placing it at different locations. By using augmented reality, many different levels can be played using the same physical objects, providing variation and hopefully motivating the user to train the neuroprosthesis. At the same time data can be collected about the rehabilitation progress which might provide useful insights in the future.

1.1. Aim and limitations

The aim of the thesis is to develop a game that can later be used in rehabilitation. However, the neuroprosthesis is in the research phase, and not functional today. The game will thus be tested on users without any physical disability — with the aim that it should be enjoyable in its own regard, and not just for patients practicing grasping.

A major constraint of the game is that it should be readily available without requiring expensive hardware. It is thus developed to run on an Android mobile phone. This places significant performance constraints on the game. The second major issue is to make the game accurate enough to be enjoyable — indeed, if game does not perform the actions the users wanted it will not be enjoyable or very useful for rehabilitation. These two constraints work against each other — the more accurate the game the more computing power it requires. A large part of the thesis, and the work performed, is about the algorithms developed to satisfy both constraints sufficiently.

A secondary aim is to provide useful data about how the future patients interacts with the game. It is thus developed to measure how the user moves the object, how fast it performs certain tasks etc. While the thesis does not analyze this data, the plan is that it can be extended and analyzed in the future.

1.2. Thesis outline

The thesis is divided in two independent parts:

1. The game design from a training and enjoyment perspective
2. Software development and algorithm design used to realize the game

In the chapter that follows (chap. 2) we explore the research question regarding augmented reality physical games, why Sokoban was chosen and its related work. Subsequently in The App chapter (3), the game is explained from an end-user perspective without technical details. The algorithmic part is separated into its own self-contained chapter (4) which outlines the algorithms invented, compares them to existing algorithms, and presents performance evaluations. The User Studies chapter (5) evaluates the user experience of the game.

2. Background

2.1. MoreGrasp

Know Center is part of the European Union funded project MoreGrasp.

> The aim of the MoreGrasp project is to develop a non-invasive, multi-adaptive, multimodal user interface including a brain-computer interface (BCI) for intuitive control of a semi-autonomous motor and sensory grasp neuroprosthesis supporting individuals with high spinal cord injury in everyday activities.
>
> — MoreGrasp.eu

MoreGrasp aims to equip spinal cord injured patients with a neuroprosthesis to enable them to cooperate with their hands. The neuroprosthesis will contain a brain-computer interface which is used to analyze the patient's brain signals and recognize activity intentions like grasping. Low energy electrical impulses are then applied to the relevant muscles, which perform the intended movement. The technique, called functional electrical stimulation (FES), effectively augments a damaged nervous system.

The brain-computer interface technology translates brain activity to machine-intelligible patterns. However the assignment of brain activity patterns are user-specific. Each individual must thus perform a lot of training, which often involves repeating the same movement over and over. This process is very repetitive and tedious for the patient, especially as many patients have not only motor but also cognitive impairments [1]. Such a group is often hard to motivate to perform the required training as the tasks may seem pointless. "Gamifying" the training process has the potential to alleviate these problems and motivate the patient, while collecting the necessary data if designed appropriately [1, 2].

Learning how to successfully grasp again is divided into two phases. At first, BCI training and FES training is performed separately until the patient achieve satisfactory results with the neuroprosthesis. This process uses guided and supervised training. After this the training consists of both parts combined. This game developed in this thesis targets the second phase. There are papers exploring BCI training games, like [1] and [3]. However these are digital games using BCI input directly. This project focuses on a physical game which is not directly connected to the BCI cap, but trains it indirectly.

2.2. Tangible games[1]

In the past decades, interactions in virtual environments e.g. computer screens, tablet, phones, virtual reality (VR), augmented reality (AR), have become common and replaced a lot of interactions with real physical objects especially in digital games where VR and AR are getting more popular everyday. Digital games have the possibility to stimulate the imagination of the user by presenting a virtual world of objects and characters. In addition, animations of characters and objects can be provided in order to enrich the experience which would be almost impossible to stimulate in a tangible game. Moreover, digital games can provide levels and context switching very cheap and effortless way for the user as opposed to the tangible games where changing levels, scene and objects first requires physical objects and then it requires user's effort for setting up everything and so it they are less flexible in almost every aspects.

On the other hand physical interactions and touch play a essential role in enjoyment of life [5, 6]. Moreover, physical touch provides a feedback mechanism for regulating interaction. Therefore, tangible games provide a unique experience as the touching, grasping and physical interaction becomes part of the game. In addition, as interaction requires physical activity, several researchers have have taken advantage of tangible games to perform some targeted physical activity beneficial for occupational therapy and rehabilitation [7, 8, 9].

Tangible user interfaces use direct manipulation of physical objects to interact with digital environments. They aim to empower collaboration, learning and design by taking advantage human ability for physical interaction [10].

Although, tangible interfaces are applied in different domains, in this section we will focus on tangible games. In [6], Cheock et al. presented a VR tangible game with the goal of preserving the physical and social interaction aspects of physical games. Although the game was presented through a head mounted display, users still were required to walk around a large room and interact with physical objects. In a study of 40 students, the authors found the tangible game to be more enticing to users, and to promote more human-to-human interactions when compared to the same digital only game. Other studies have researched games with tangible interfaces with similar findings, namely that games with tangible interfaces provide more engagement [11], social interaction [12] or that they accelerate learning [13] when compared to the same digital versions of games.

In occupational therapy, games with tangible interfaces are for instance after strokes to support rehabilitation of motor upper extremity movements [14, 8, 9], wrist flexion and extension[7] or postural control [8]. In most of these papers, authors use Wiimote based Controllers [7, 8, 9] to capture physical movements or a CyberGrasp device [14] to provide resistive force feedback.

[1]This section is based on a paper co-authored by me[4].

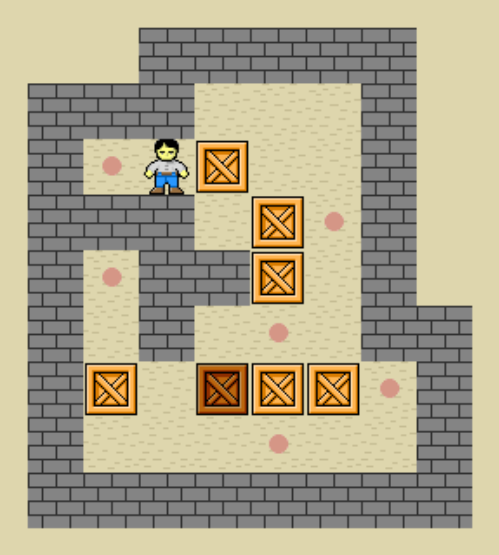


Figure 2.1.: An implementation of Sokoban. The boxes are pushed one step at a time, horizontally or vertically, to any free adjacent square. The game is completed when all boxes are at a goal position square — here indicated by a red disc in the center of the square.

2.3. Sokoban

Sokoban is a 2D puzzle game, in which the player pushes boxes on a board of squares. The goal is to push each box to any of the predefined goal positions on the game board, see fig. 2.1. The challenge lies in that the player can only push the boxes — not pull them. They must thus be careful not to create deadlock situations where the boxes can no longer be pushed to a goal position. The levels can be designed to be of varying difficulty, some trivial while others provide quite a challenge. When a level has been completed the players can further challenge themselves by trying to complete it in the minimal amount of moves. Solving a level, and solving it in the minimal amount of moves has been shown to not only be in NP-hard, but in PSPACE [15].

The version of the game designed has the same gameplay mechanics as the original version. The difference lies in how the game is controlled and displayed to the player. In the original version the player controls the game with the 4 arrow keys, deciding in which direction to move the player. The level is normally displayed from a birds eye perspective, such as fig. 2.1, allowing the player to see the full level at all times. The version developed is instead controlled by moving a physical object around in relation to a pattern on a flat surface. The level is virtually added on top of the pattern when looking at it through a screen, such as a smartphone or virtual reality glasses. It is rendered in 3D with the same perspective as the world, making it appear as if the level is naturally a part of the real world for the player.

3. The app

In this chapter the gameplay mechanics and features of the game are explained, as well as a brief overview of its implementation.

The Sokoban app consists of three programs, one to calibrate the camera, and two versions of the game. The difference between the two game versions is solely how the game is displayed to the user. The "regular screen"-version displays the game on the smartphone's screen directly, while the other version uses Google Cardboard[1], a makeshift device to turn a regular mobile phone to a virtual reality device. To play the Cardboard version, the user puts their phone into a cardboard viewer, which is then placed on their head (see fig 3.1).

3.1. Gameplay

The game is controlled entirely by moving around a physical object, hereafter *controller*, see fig. 4.9. The controller is moved around in relation to the *game board*, which is a specific pattern printed to a paper, put on a flat surface such as a table in front of the player. The game works like a state machine with state changes triggered by changes in the position of the controller in relation to the game board. See fig 3.2 for an explanation of the game mechanics. For a video, see `http://ojeling.net/thesis/sokoban.mp4`.

If the player moves the object without first lifting it up, the game will go into the same state as in the beginning of the level. This method of playing is motivated by encouraging the player to practice grasping by letting go and grasping the object to perform each move.

3.1.1. Controllers

Three different controllers are available to use for the user. The controllers are custom designed and 3D-printed. The basic design of the controller is motivated in section 4.3.4. Three variations of the controller are available to promote different grasps, see fig. 3.3.

[1] A Google Cardboard application is developed using the Cardboard SDK which takes care of the internals so the game can be displayed is a combination of a Cardboard viewer and an SDK taking care of cardboard specifics such as distorting the image for the lenses of the viewer. `https://vr.google.com/cardboard`

Figure 3.1.: A user plays the Google cardboard version of the game.

3.2. Interface

3.2.1. Regular screen specifics

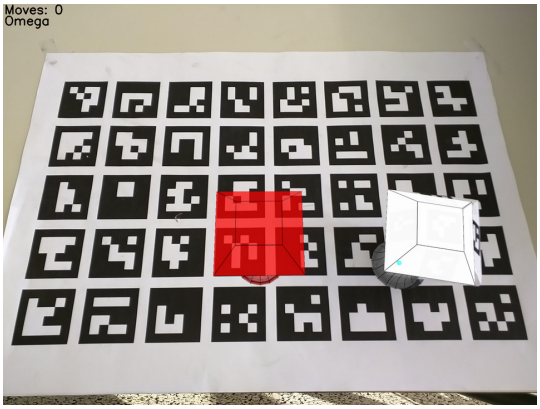
Touch operation	Action
Left swipe	Change to the next level
Right swipe	Change to the previous level
Double tap	Restart the current level

3.2.2. Cardboard specifics

Since the user can not touch the screen, the Cardboard version uses a trick where it measures the magnetic field with the device's built in magnetometer. When the field is disrupted the level changes to the next. The field can be disrupted by moving a weak magnet over the phone.

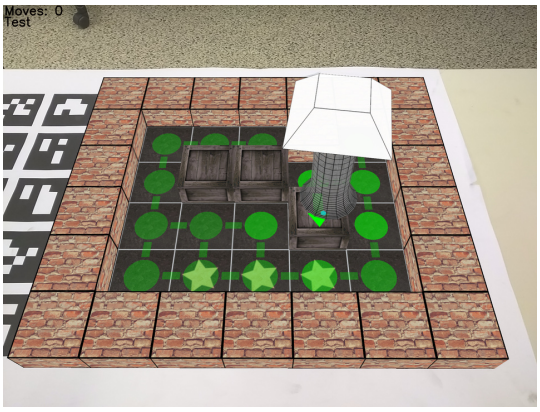
3.2.3. Bluetooth keyboard

If a Bluetooth keyboard is connected, key-presses can control some aspects of the game. This is useful for an instructor conducting experiments. An additional mode where the

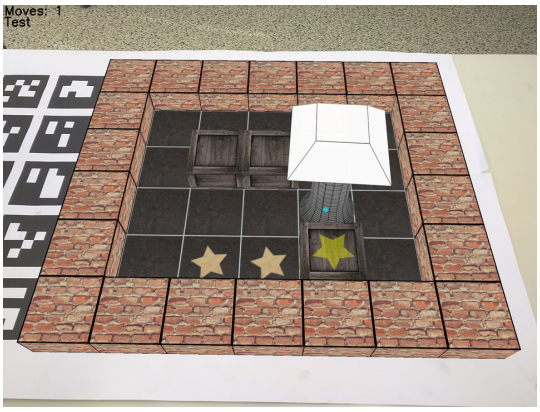


(a) The initial state of the game. The player must place the controller on the square designated as the starting position. This is indicated by the augmented red outline of the controller.

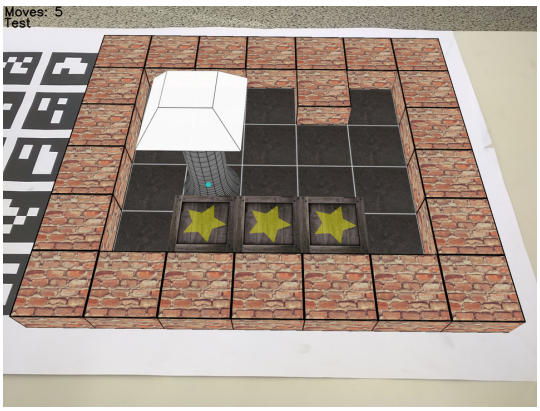
(b) The controller is positioned at the starting position. A move is initiated by lifting the controller orthogonally in relation to the game board.



(c) A move has been initiated. The squares the player can move the controller to is indicated by green circles. The boxes the player can push is indicated by green arrows. If the player places the controller on a box, it will be pushed one square further in the same direction.



(d) The move has been performed by putting the controller down on the box. Since the box was pushed to a goal position it is now has a star is on top of it.



(e) After multiple moves all boxes are at goal positions. The starts starts spinning to indicate this to the player.

Figure 3.2.: The Sokoban augmented reality game. Note that the renderings occlude the physical controller and the player's hand. Thus a virtual marker is rendered on top of the real marker.

Figure 3.3.: Three 3D-printed controllers for practicing different grasps. The keys are used for pinch grasps.

player is required to press space between each box move is included. This forces the user to let go of the controller between each move, thus ensuring grasp practice.

Key press	**Action**
1-9	Change the level
Q,W,E	Change configuration
Shift	Restart the current level
Space	Signal the game as explained above (if enabled)

The following game configurations are available:

Key press	**Action**
Q	Controller 1 used (default)
W	Controller 2 used, Space press required
E	Controller 3 used, Space press required

3.2.4. Exposure control

The exposure and ISO sensitivity of the camera must be set manually. This is done by tapping once on the regular screen version of the game, and adjusting the sliders until

it looks good. Afterwards the user can start the cardboard version, which will use the same settings. If the lighting conditions are good the exposure time is recommended to be between 1–5ms, to allow the user to move their head quickly without motion blur interfering with marker detection.

3.3. Data collection

For each level played a log file is created on the Android device's externally accessible storage. It consists of the 6 degrees of freedom of both the controller and game board over time. It also stores events for all actions performed. See appendix A for further details.

3.4. Camera calibration utility

Before the game can be played the camera has to be calibrated once for each phone. In this mode the user takes several pictures from different angles of a printed reference chessboard pattern, see fig. 3.4. It works as follows:

1. Point the camera at the printed chessboard and tap the screen.
2. After processing the frame, the corners of the chessboard pattern light up in blue if a probable pattern is found, or the application returns to step 1.
3. The user can either accept the pattern by tapping the screen once, or reject it by swiping left. The user should accept the pattern if the corners of the chessboard are all lit up in blue.
4. Repeat step 1–3 for different angles. At least 10 times are recommended.
5. Double tap the screen to compute and store calibration. Wait about 20 seconds to ensure it is stored.

3.5. Implementation

The game is mostly implemented in C++ using the Android NDK[2]. Java code is used to interface with the Android subsystems, such as fetching the camera frames and creating the OpenGL context. OpenGL ES 2.0 is used for 3D graphics, the camera2 android API is used to control the camera exposure and sensitivity manually — it is indeed important for accurate marker detection that the motion blur is minimal, thus a low exposure time is required. The application is multi-threaded and the algorithms parallel. See fig. 3.5 for an overview of the program. The marker detection part itself is described in chapter 4. The game does not use any additional frameworks, with the 3D engine being custom made.

[2] `https://developer.android.com/ndk`

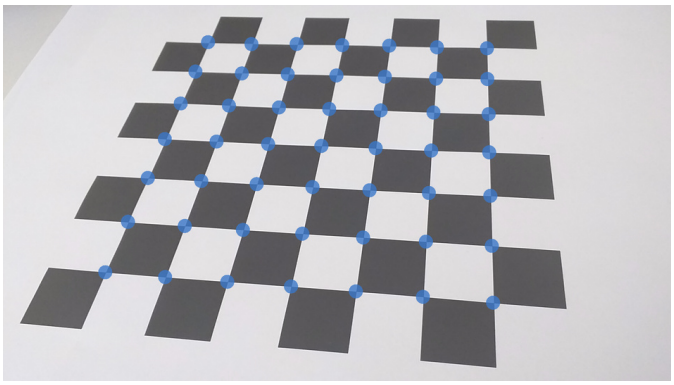


Figure 3.4.: The calibration mode as it appears from within the application. The user captures many photos of the chessboard printout from different angles. The corners of the chessboard lit up in blue.

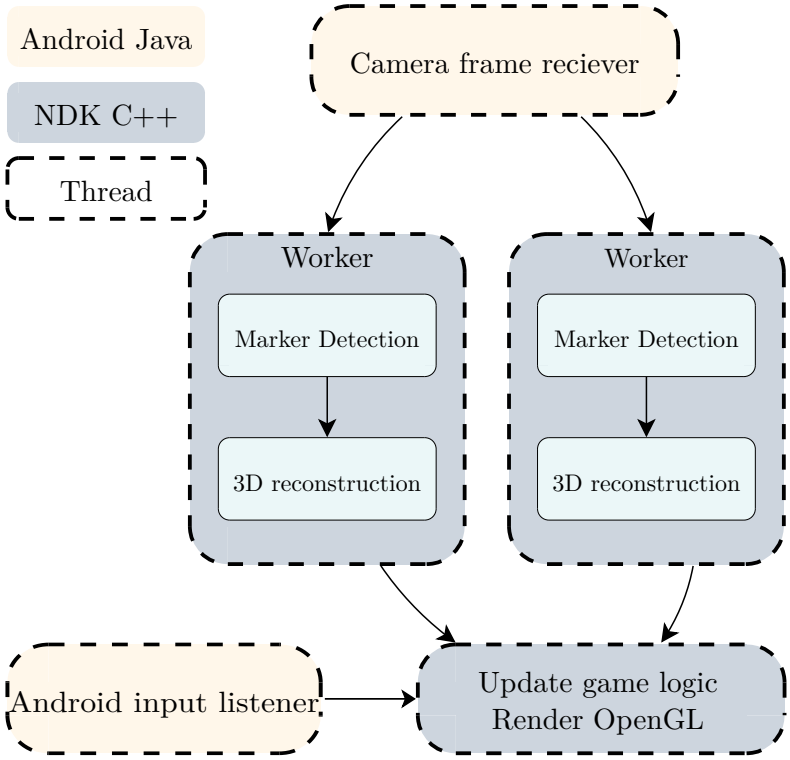


Figure 3.5.: The Sokoban program. It consists of several threads, with the marker detection subtask itself parallel.

4. Marker detection and pose estimation

In this chapter the algorithms for marker detection and pose estimation are explained and evaluated. The layout of the game board and the controller are also explained.

4.1. Related work

In computer vision there are two conceptually different approaches to find the perspective and pose of preknown patterns in a scene. Natural feature tracking (NFT) or using specifically designed markers. With NFT any image can be tracked. It works by finding high frequency features in an image, such as corners, and computing certain statistics about them. The features of a reference image are matched to the features of another image. If more than a threshold of them match and have consistent perspective, the perspective and pose of the reference image can be found in the scene. The other approach is to design special markers which are analyzed semantically. Typically these markers are black and white to provide maximum contrast. Many of these markers can be printed in a predefined grid, providing redundancy in case some are occluded.

The advantages with the latter approach is lower processing power required, better resistance to different lighting conditions, motion blur and different distances of the camera. The advantage of NFT is that any image can be used. The current de facto standard for NFT is Vuforia, a proprietary framework developed by Qualcomm. For marker tracking two common open-source libraries are ARToolKit and Aruco. While NFT is considered to be theoretically slower, Vuforia has a large commercial backing, is heavily optimized and can process camera frames in real time on modern phones.

In this chapter a new algorithm for marker detection is developed, specialized to work well for the use case of the Sokoban game.

4.2. Theoretical Framework

In this chapter existing technologies and frameworks, and how they relate to the thesis, are explained.

4.2.1. OpenCV

OpenCV[1] is an open source library originally developed by Intel. Currently maintained by the non profit foundation OpenCV.org, it is designed for computational efficiently and aims mainly at real-time computer vision applications. The library provides the basic building blocks for the marker detection algorithm.

4.2.2. Projective geometry

A projective transform maps a set of coordinates from one space to another, with the important property that lines maps to lines. Augmented reality is based heavily on the mathematical properties of those transforms, from reconstructing the 3D scene from a 2D image, and then render virtual 3D objects on top of the 2D image.

Homography transform

A homography transform establishes a mapping between two projective planes $\mathsf{P}^2 \to \mathsf{P}^2$ [16]. A 2D point (x, y) in an image can be represented as (x_1, x_2, x_3) where $x = \frac{x_1}{x_3}$ and $y = \frac{x_2}{x_3}$. This is called a homogeneous coordinate. Given a 3×3 matrix H, we can map a set homogeneous coordinates from one plane to another:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} u \cdot s \\ v \cdot s \\ s \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

H can be computed unambiguously from 4 point-to-point correspondences between the two projective planes. It is important to note that the homography transform only transforms from one plane to another — it does not give out any information about the 3-dimensional properties of the scene, such as the axis orthogonal to the plane.

A homography transform is computed once per marker in the image to transform the projected marker to a top-view perspective. It is then possible to efficiently compute how much of each cell in the marker is colored, as the borders are axis parallel (see fig. 4.1).

Perspective projection

When the eye or a camera sensor views a scene, objects further away appear smaller — this is know as perspective. A perspective projection is a mapping from $\mathsf{P}^3 \to \mathsf{P}^2$. Given a projection matrix K, and a coordinate in the world $[x, y, z]^T$, its R^2 projection $[u, v]^T$ is defined as

[1] `http://opencv.org`

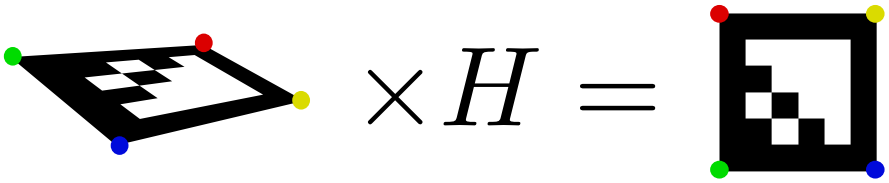


Figure 4.1.: The homography transform maps a projective plane to another, $P^2 \rightarrow P^2$.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} u \cdot s \\ v \cdot s \\ s \end{bmatrix} = K \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Where s is a scaling factor, also know as the perspective divide, such that the third row is 1. It is the division of s that makes objects further away seem smaller on their 2D projection. In section 4.2.3 we will define K for the camera sensor.

4.2.3. Camera Model

A good mathematical model for the camera sensor is necessary to accurately find the perspective and pose of the controller and game board.

Intrinsic parameters

Each camera has different parameters inherent to its lens and sensor, called the intrinsic parameters. Those parameters determine how a 3D coordinate in the world will be projected on the camera's sensor. There are numerous ways to model this. In this application we use the pinhole camera model, a simplification where the camera aperture is described as a point with an infinitely small aperture, and no lenses are used to focus light (see fig. 4.2). A good estimation of the parameters are required to successfully estimate the 3D pose of objects captured, and to relate the camera sensor's native unit, pixels, to world units, such as millimeters.

We define the camera matrix, K, for the linear parameters, where f is the focal length and c the optical center. Both expressed in pixel units.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

Given a coordinate in the world $[x, y, z]^T$, with origin at the camera lens and the Z axis forward, see fig. 4.2, the projection on the camera sensor is:

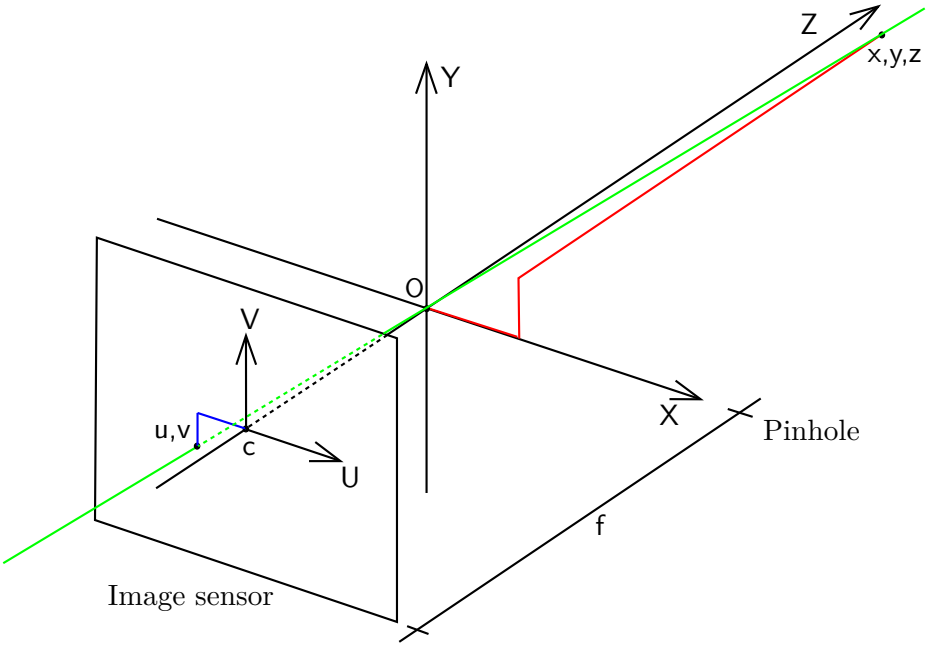


Figure 4.2.: The pinhole camera model. *Modified graphics. Original license public domain.*

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Where u, v is the coordinate in pixel units on the sensor, s is a scaling factor, also know as the perspective divide, such that the third row is 1. It is the division of s that makes objects further away seem smaller on their 2D projection[17].

Lens distortion

The pinhole camera model does not take into account the distortions from imperfections of the camera lens. This problem is especially significant on cheap camera lenses such as those found on mobile phones. Another model is used to compensate for several distortion parameters, such as barrel and sheer distortion (see fig. 4.3).

Extrinsic parameters

The extrinsic parameters define a transformation $P^3 \rightarrow P^3$. It consists of a rotation and a translation, which transforms from the object coordinate system to the camera coordinate system. Intuitively it describes the 6 degrees of freedom of an object in the

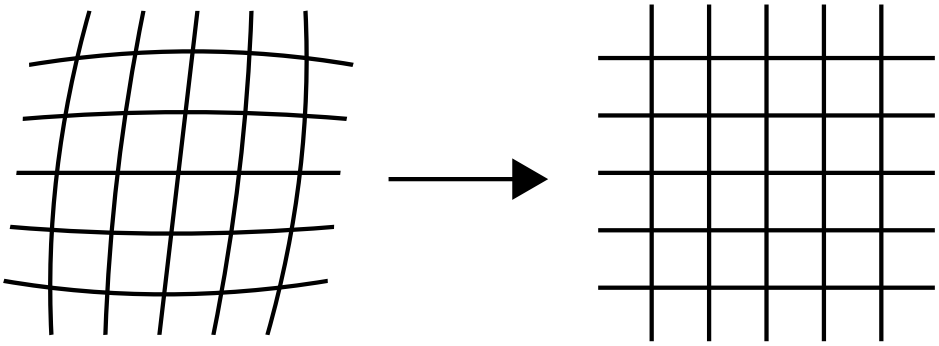


Figure 4.3.: Cheap lenses such as the ones found in mobile phones often have significant barrel distortion. An image can be undistorted if the distortion coefficients for the lens are known. On the left an exaggerated distorted image and on the right its undistorted version.

scene. While the intrinsic parameters are calibrated once per sensor, the extrinsic are computed for each camera frame.

4.2.4. Thresholding

Thresholding classifies lightness intensities of a gray-scale image to to a binary image (black and white).

Global thresholding A global threshold is fast and simple, each pixel in the binarized image $b_{i,j}$ is set according to the following rule, where $g_{i,j}$ is the pixel value for a greyscale image:

$$b_{i,j} = \begin{cases} 1 & \text{if } g_{i,j} > \text{value} \\ 0 & \text{otherwise} \end{cases}$$

Otsu thresholding

Determining a good threshold value can be hard since the lightness of the image changes depending on dynamic factors such as the amount of light in the room. If it is possible to make the assumption that the image contains two distinct brightness levels — an optimal threshold value can be computed so that the intra-class variance is minimized. This is called an Otsu threshold. The single threshold value is first computed and then used as the threshold for the entire image. For thresholding a part of the image containing a single marker, an Otsu threshold provides good results in practice with minimal computing power.

Adaptive thresholding

A camera image has some properties that makes it difficult to use a single threshold value for the entire image. Highlights and shadows make the absolute brightness throughout the image very uneven. An adaptive threshold uses a different threshold value for each pixel. The threshold value for a pixel is computed by taking the mean of the pixels of a square window of some size centered on that pixel. It is thus very good at detecting edges in the image. However, if the window size is too small it may result in a lot of noise, while if it is too big might miss detail and suffer from the same problem as a global threshold. Adaptive thresholding is significantly slower than global thresholding, as the image is convoluted to find the mean value for each window.

$$b_{i,j} = \begin{cases} 1 & \text{if } g_{i,j} > \text{mean}_{i,j} \\ 0 & \text{otherwise} \end{cases}$$

4.2.5. RANSAC

Random sample consensus — RANSAC — is an iterative method to estimate parameters from data that may contain a relatively large amount of outliers [18]. For the purpose of this application it works as follows:

1. A subset of the data is randomly selected and used to compute a 3D→2D transformation.
2. The transformation is used to re-project the entire data-set 3D→2D.
3. For each point the Euclidean distance between its original coordinate and its re-projection is computed. If it is lower than some threshold it is counted as an inlier
4. The above steps are repeated many times for different subsets of the data.
5. The points from transformation which yielded the largest amount of inliers are designated as good inlier points.
6. All good inliers points are used to compute the final transformation.

RANSAC produces non-deterministic results, however with a sufficiently large amount of iterations the result works very well in practice.

4.2.6. Disjoint set

A disjoint set is a data structure that has a number of non-overlapping sets that partition the elements. Finding which set an element belongs to is $O(1)$. Merging two sets is approximately[2] $O(1)$ [19].

[2]Merging two sets is amortized $O(\alpha(n))$ where $\alpha(\cdot)$ is the inverse Ackermann function. It grows so slowly it is approximately $O(1)$

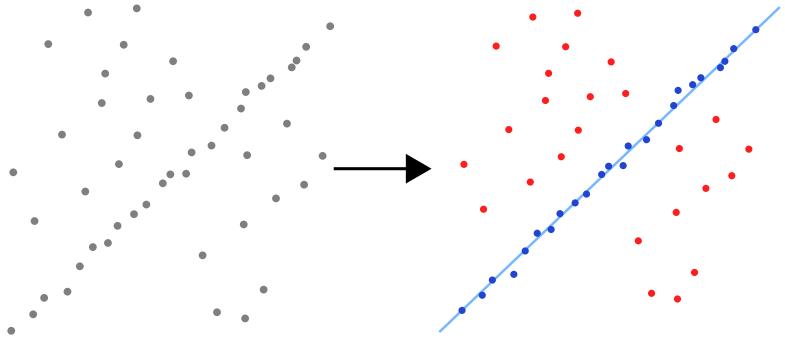


Figure 4.4.: RANSAC being used to fit a line to a set of points with a large amount of outliers. Only the points designated as inliers (blue) are used to compute the best fit. *Modified graphics. Original author wikipedia.org user "Msm". License: CC BY-SA 3.0.*

4.2.7. NEON Instruction set

NEON[3] is the ARM processor extension for SIMD — Simple Instruction Multiple Data. It performs simple arithmetic operations on multiple data in parallel. Contemporary ARM processors in smartphones have 128bit NEON instructions, allowing for example 16x8bit data to be operated at once. A few performance critical functions have been vectorized to take advantage of this. See section 4.4.3 for performance evaluations.

4.3. Implementation

For the controller and the game board, we wish to find their position in relation to the camera sensor for each frame — in other words to find the extrinsic parameters (see section 4.2.3). This is conceptually a two stage process:

1. The corners of the markers in the frame and their id:s are extracted.
2. A point to point correspondence between the observed corners and their supposed position in relation to each other are used to compute the perspective and pose of the object.

4.3.1. Marker Detection

The marker detection algorithm finds and identifies corners of specific markers in an image. The markers consists of a black border with a 4x4 grid of cells inside. Square cells are used as they can be identified efficiently with high accuracy, even in unfavorable angles and poor lightning conditions. An overview of the algorithm can be see in fig. 4.5.

[3] `http://www.arm.com/products/processors/technologies/neon.php`

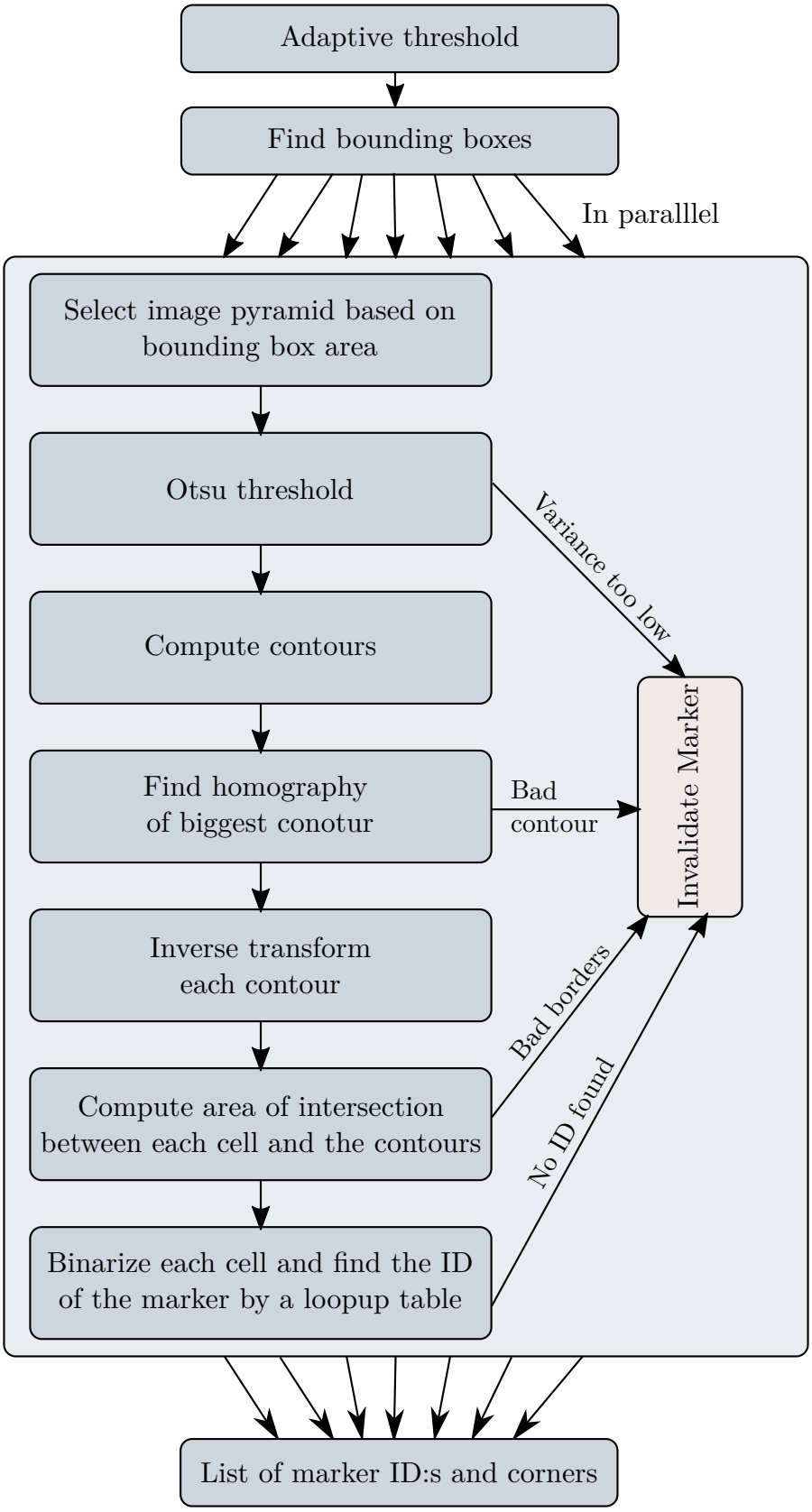


Figure 4.5.: The parallel marker detection algorithm

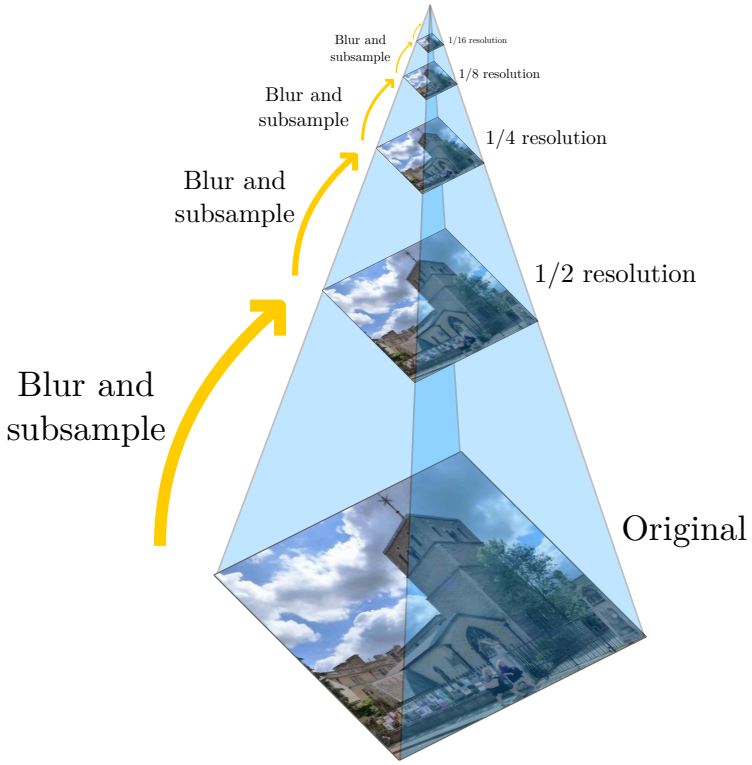


Figure 4.6.: Image pyramids are used to speed up computation for the marker detection. *Modified graphics. Original author wikipedia.org user "Cmglee". License: CC BY-SA 3.0.*

Initialization

The algorithm starts by computing image pyramids. Image pyramids down-sample the image by halving both the width and height (see fig. 4.6). A specialized version using the NEON instruction set is used where the mean value of 4 pixels is used for each new pixel. Some image pyramid methods, including the one in OpenCV, uses a Gaussian window instead. This yields higher quality at the cost of more computing power. Testing showed that the quality degradation is not a major concern for the algorithm.

Adaptive threshold

An adaptive threshold (see section 4.2.4) is performed on an appropriate image pyramid. The result is a binary image with the markers disjoint. The corners of the markers might be slightly offset from their real position. However this is not important as seen later.

Find bounding boxes for the connected components

A connected component analysis is performed on the binarized image, returning the bounding boxes of each connected component. Bounding boxes that are too small or have odd shapes are discarded. Typically hundreds of valid bounding boxes are found in a camera frame. Each bounding box contains a potential marker. The bounding box algorithm is described in detail in section 4.3.3.

Parallel marker identification

Each bounding box is checked for a valid marker. After this stage the corners of all valid markers, and their ID:s are returned. The bounding boxes are checked in parallel, making the algorithm scale well on multi-core systems.

Pyramid selection and Otsu thresholding At first a suitable image pyramid is selected based on the height and width of the bounding box. An Otsu threshold is computed on the selected image-pyramid inside the bounding box. The use of image pyramids are important as the next step, contour finding, is computationally expensive. If the variance between the two classes of lightness in the image is too low — meaning the result is just noise — the marker is discarded.

Contour finding A contour finding algorithm is run on the binarized image. The largest contour is selected. If that contour is too small, does not have 4 corners or is not convex the marker is discarded. A hierarchy of the contours are established, returning a tree of which contours are inside each other.

Homography transform Given the largest contour found in the previous step, the homography transform is computed, see section 4.2.2. Each contour inside the largest is transformed to a top-eye perspective. This is done so each cell in the marker will be in a 1x1 wide space of arbitrary units. It is easy to compute the intersection of a cell with straight edges and an arbitrary contour.

Contour intersection and area computation The marker consists of 6x6 cells, of which the outer border is 1 cell wide, and the actual ID of the marker is encoded with 4x4 cells. At first the area of intersection for all contours is computed for each cell. For the edge cells it is required that they are filled above a certain threshold, or the marker will be discarded. For the inner cells a majority vote is used. The area used for computation is offset by a small amount on each side of each cell, giving margin for image artifacts and motion blur.

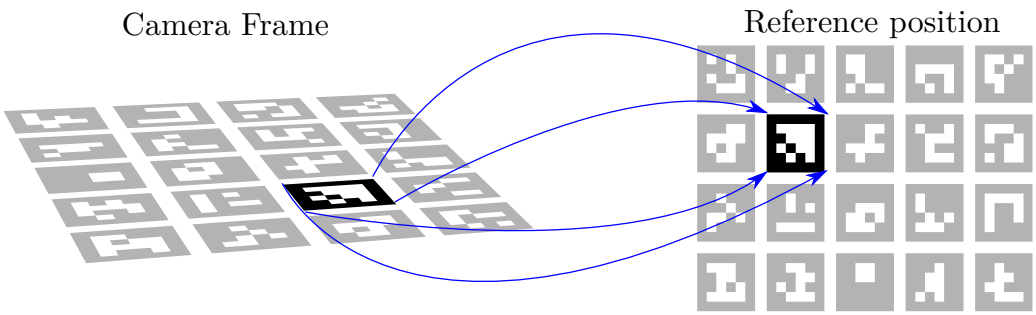


Figure 4.7.: The marker coordinates and their respective ID:s have been identified in the camera frame. The corner for each marker is associated with its reference position, which is used to find the perspective and pose of markers in the camera frame.

Determination of Marker ID Since each marker consists of 4x4 cells, it can represents 16 bits of information. A precomputed lookup table is used associate each pattern to one of 256 ID:s. To recover the rotation of the marker, each ID is represented 4 times, giving 1024 valid bit-patterns. The markers are selected in such way that the Hamming distance from each other is at least 2 bits. Thus one bit error is allowed per marker, meaning that 16K bit-patterns out of the 64K possible, 15%, will return a valid ID. The markers are identical to those of the Aruco library. The generation and motivation of the markers are described in [20, 21].

Precision and recall

The parameters of the algorithm are tuned to have a high recall rate, with a lower precision as a consequence. There are often some parts of the image being incorrectly recognized as markers. This is however not a great concern for the purpose of the game, as the pose estimation step uses robust inlier detection.

4.3.2. 3D pose estimation

Finding the transformation from a set of 2D-projected points to their original 3D coordinates is a well understood problem. First a mapping between the marker corners in the image their known relative reference position in the 3D space is established using the marker IDs (see fig. 4.7). Next the rotation and the translation of the object is reconstructed using RANSAC as explained in section 4.2.5. The extrinsic parameters have been recovered.

4.3.3. Bounding boxes for connected components

Connected component analysis is used to detect regions of connected "filled pixels" in a binary image. 8-connectivity is used, meaning two pixels are connected if their edges or corners touch. A minimal bounding box is the smallest axis parallel rectangular box containing all pixels of a component. For the marker detection algorithm the bounding boxes are used to find areas of interest which are to be checked for markers.

The conventional algorithm to find bounding boxes first labels each pixel in the image, then goes through the labels pixel by pixel to update the left-, right-, top-, and bottom-most coordinate for each label [22]. In addition to requiring two passes, it also requires $O(nm)$ extra memory to store the labels of each pixel, where n, m is the height and width of the image.

A custom algorithm was developed exploiting that only the bounding boxes and not the labels themselves are needed. This eliminates the need for the extra memory and manages to compute the bounding boxes in one pass instead of two. In practice it is about 3-5 times faster for a typical camera frame. See section 4.4.2 for performance measurements.

The algorithm scans the pixels in memory (row-major) order. It uses 2 data structures to keep track of the bounding boxes:

1. The labels of the pixels in the row above the current.
2. A disjoint-set (see section 4.2.6) like data structure to efficiently keep track of and merge bounding boxes.

The algorithm works as a state machine, it scans each row left to right until a filled pixel is occurs. Once it occurs it needs to be assigned a label. For the 2^3 combinations of pixels in the row above, there are 3 cases for deciding which label it will be assigned:

Legend:
□ Empty pixel
■ Filled pixel
◪ It does not matter if the pixel is filled or not

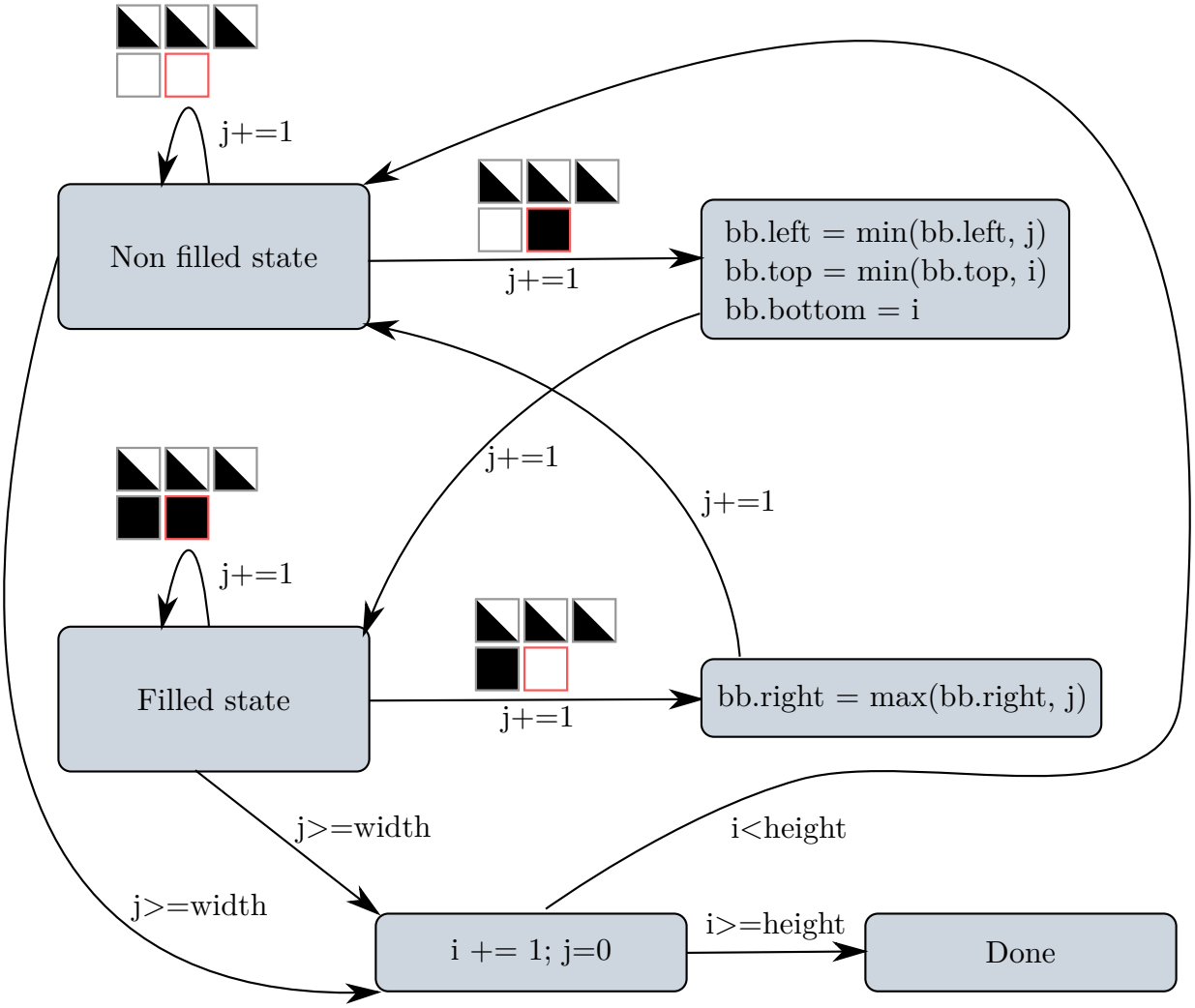
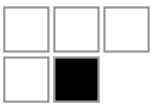
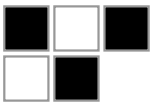


Figure 4.8.: The state machine for the bounding box algorithm. i and j are the current row and column. bb is a reference to the bounding box associated with the current pixel.



No pixels above are filled. A new label is generated and the pixel assigned it.

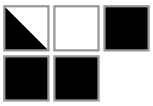
One, two or three continuous pixels above are filled. The pixel will be assigned the label of the above pixels.



Two non-adjacent pixels above are filled. There might potentially be a conflict since the new pixel may connect the labels for the two pixels above for the first time. If that is the case the labels above and their bounding boxes need to be merged. The label for the pixel will be the smallest of the two above.

State filled

When the state machine is in the "filled" state the current pixel will always be assigned the label of the pixel west of it. Two out of the 2^3 cases may cause a label conflict and thus require a label and bounding box merging:



The value of the north-west pixel is not important, it was handled previously. Likewise the north pixel was handled while processing the west pixel if it was filled. If both the north and the north-east pixel are filled, they will be the same label and will have been handled by the west pixel. However if the north pixel is not filled and the north-east is filled, this may be the first time the labels of those pixels are connected. A merge is thus required.

Complexity analysis

Giving a binary image with n rows and m columns. Each pixel will be visited once, resulting in $O(nm)$. For each pixel there might be a merge of labels, which is approx-

imately $O(1)$ (see section 4.2.6). The final run-time is thus $O(nm)$. This is the same time complexity as the preexisting algorithm in OpenCV. However the advantage of the new algorithm is that the bounding boxes are computed in one pass instead of two.

The space requirement is lower since the labels for each pixel are not stored. However as there might be $O(nm)$ bounding boxes it has the same upper bound too. In practice the number of bounding boxes is a lot lower than $O(nm)$ though. With k bounding boxes, the space complexity improves to $O(m + k)$, where m is the storage required to save the previous row of pixels.

4.3.4. Controller

The design of the controller went through 3 prototypes. Initially a single flat marker was placed on top of a cuboid made out of cardboard. This suffered from two problems:

Noise

The recovered perspective and pose of the object relative to the game board was very noisy from certain angles, meaning that sometimes incorrect moves would be performed. It turned out that the axis orthogonal to the camera sensor, see the Z axis in fig. 4.2, was the worst. The noise is a natural consequence of the resolution of the image. This is especially true for narrow angles, which could be very unstable.

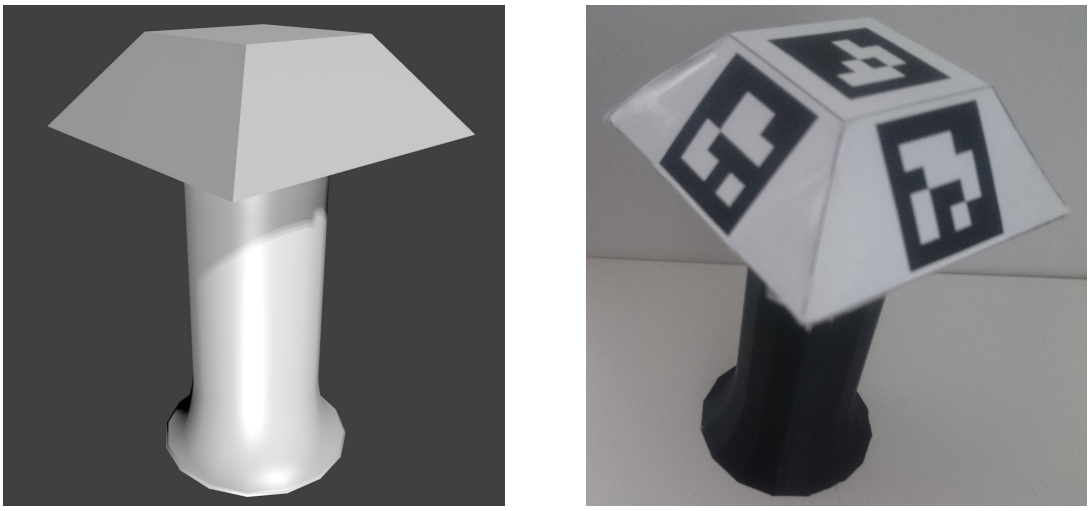
Theoretical limitations

The 3D pose of an object cannot be recovered unambiguously from only co-linear points — there are in fact 4 valid solutions [23]. By empirical testing it turned out that the pose would oscillate between two of these valid solutions, posing a serious challenge to the playability of the game.

Solution

A hypothesis to solve the first two problem by making the marker 3-dimensional was tested by adding an additional marker to the side of the cuboid controller. This constrained the axis perpendicular to the flat marker and solve the theoretical issue of the ambiguous pose. The test proved successful. However as the controller is typically viewed from above while playing the additional marker would rarely be visible in practice.

A custom 3D-printed controller was created, see fig. 4.9. The pattern on top of the controller is shaped as a frustum with 5 sides, each with a marker. From every position typically viewed in a gaming session there will be at least 3 markers visible. This eliminates both the noise- and theoretical problem completely. See section 4.4.1 for a quantitative evaluation.



(a) A rendering of the 3D model created in blender.

(b) The 3D printed physical controller with a pattern of Aruco markers added.

Figure 4.9.: The physical controller.

4.4. Evaluation

All tests are run on the Motorola Nexus 6, released in 2014.

4.4.1. Controller

In section 4.3.4 it was conjectured that the 3D configuration of markers improve the accuracy of the perspective and pose recovery. The conclusion was motivated by iterative testing based on moving the controller around and looking at the accuracy in real-time. A quantitative test is performed to verify the assumption and to measure how much better the frustum configuration version is. A level of Sokoban is played as usual without taking any extra consideration to the test itself. The translation vector of the two recovered poses of the controller are used. The rotation is not compared as it only affects the visual experience and not the gameplay directly.

The first important metric is if the controller is found at all. Using the frustum configuration the controller is recovered for 99% of all frames, while for the single marker it is recovered 94% of the frames. This is a clear advantage for the frustum marker. However as the game is often viewed top-down the top marker is in view most of the time, yielding a decent performance for it too. Note that the missing 1% for the frustum configuration is not necessarily the algorithm failing — the controller could simply be out of view.

The second metric is the accuracy of the controller. Using the frustum configuration as

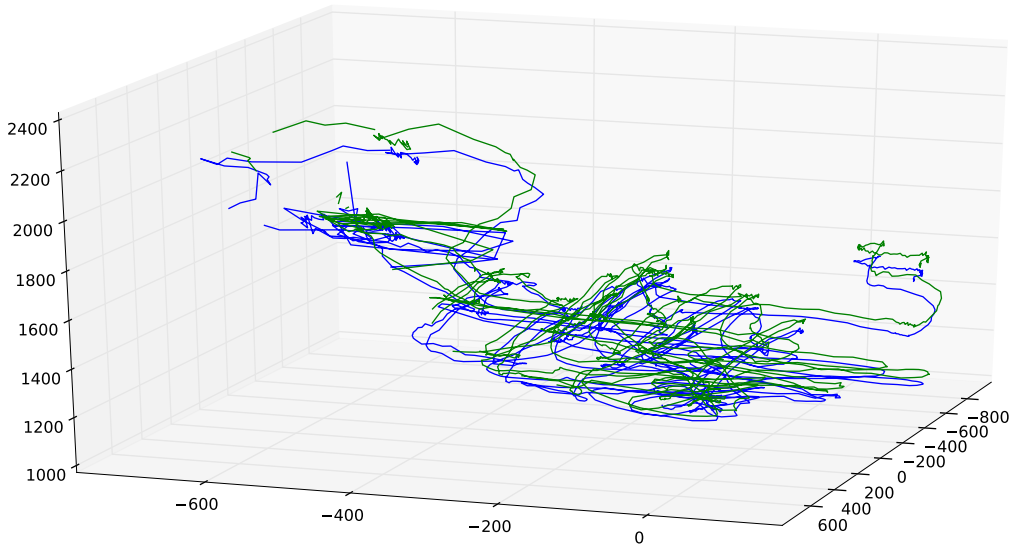


Figure 4.10.: The two reconstructed controller paths using the frustum configuration (blue) and the single marker configuration (green). It can be seen that the single marker drifts compared to the frustum. Pixel units are used.

reference, we note that the average distance between the frustum and the single configuration is 157mm with a std. dev. of 44mm. This is a clear improvement of the accuracy which can be felt while playing the game. Due to the properties of 3D reconstruction, the more markers used to reconstruct the pose the better the result. However we can not be sure if the frustum configuration is the true reference position of the controller. The test should thus be seen as indicative and not an absolute measurement (see fig. 4.10).

4.4.2. Bounding boxes

This section compares the improved algorithm to find bounding boxes to the one included with OpenCV. Both algorithms return exactly the same results, so only the computation time is measured. The test is performed by playing a level and running both versions on the same camera frame. It is thus ensured that a typical image is used and the performance for the particular use case is relevant. 1000 camera frames are used to compute the statistics. The algorithm is run on an image of size 960x540.

Algorithm	**average**	**std. dev.**	**median**	**min**	**max**
OpenCV	23ms	3.8ms	27ms	17ms	39ms
Custom	6.9ms	2.0ms	6.0ms	3.9ms	18ms

The custom algorithm outperforms the one included in OpenCV by a factor of 3.3. It has also a lower standard deviation, which is an important as a spike in computation time can cause noticeable lag for the player. Given the approximate 66ms available to

compute each frame for a smooth 30 frames per second frame rate[4], the old version spends 35% of the time computing the bounding boxes, while the new only spends 10% — making it a crucial part of the performance of the new marker detection algorithm.

Tested on a Sony Xperia Z1, with a 640x360 image size, the results are even better for the custom algorithm, improving the time by a factor of 4.6 (11ms down to 2.4ms)

4.4.3. Neon Optimizations

Like for the bounding boxes benchmark, the performance measurements are performed both for the stock OpenCV version and the NEON optimized version on typical camera frames. The NEON optimized threshold functions return exactly the same results as the OpenCV versions. However the OpenCV version of the pyramid generation returns a slightly different result explained in section 4.3.1.

Function	**average**	**std. dev.**	**median**	**min**	**max**
Pyramid generation	8.8ms	2.0ms	7.8ms	7.3ms	39.0ms
Threshold	3.8ms	1.8ms	7.8ms	0.7ms	13.4ms
Adaptive threshold	7.1ms	1.5ms	6.0ms	5.9ms	18.3ms
NEON pyramid	0.91ms	0.35ms	0.73ms	0.63ms	3.8ms
NEON threshold	0.56ms	0.44ms	0.66ms	0.16ms	5.0ms
NEON adaptive	4.2ms	1.3ms	3.4ms	3.3ms	15.7ms

The NEON optimized versions provide a clear performance boost — pyramid generation is 9.7 times faster, thresholding 6.8 times and adaptive threshold 1.7 times faster.

4.5. Competing algorithms benchmark

In this section we compare the performance of the custom algorithm compared to two existing, commonly used algorithms for similar patterns. There are two important metrics — how long does it take to compute a frame and how accurate is it.

4.5.1. Algorithms

Artoolkit

ARToolKit[5] is a commercial, open source library. It supports custom black and white markers, natural feature tracking and 2D barcode tracking very similar to the algorithm developed and Aruco. For the test 3x3 2D barcode tracking is used as it has

[4]The limit for one frame is 1/30=33ms, however as the algorithm is double buffered roughly the double time is available.

[5]`https://artoolkit.org`

the best performance. The test is based on the example project "ARMulti" included with the default ArToolKit 5.3.2 distribution. The time is measured by seeing how long the `NativeInterface.arwUpdateAR`() function call takes. The Artoolkit SDK is single buffered.

Aruco

Aruco[6][20] is a minimal library included with OpenCV used for tracking 2D markers. OpenCV 3.1 is compiled with support for NEON and multi-threading. Given that the custom algorithm is a drop in replacement for Aruco, and each frame is independent of previously computed frames, the code for double buffering designed for the custom algorithm is also used for Aruco. The time for the function calls `detectMarkers` and `estimatePoseBoard` are measured.

Custom algorithm

The same pattern as Aruco is used. The algorithm is designed to be double-buffered and parallel in itself — like Aruco each frame is independent of previously computed frames.

4.5.2. Performance test: marker detection & pose estimation

The marker detection algorithms are tested for three different number of markers, 12, 24 and 48. The setup is as follows:

- A stream of 1920x1080 image frames
- Camera set to automatic expose
- Only the functions computing markers & pose measured - color conversion, OpenGL etc are excluded
- Hover the phone 30-40cm above the pattern of markers while moving the phone slightly, always allowing all markers to be seen.
- Run the computation for 1000 frames, 300 for Aruco due to overheating of the phone causing a throttling of the CPU.
- Test with a pattern of 12, 24 and 48 markers and measure performance. Each marker is 2x2cm
- Ideal lighting: Indoor with overcast weather, no glare or complicated shadows.
- Motorola Nexus 6 2014 is used.
- The doubled buffered performance is approximated by treating the measurements as single buffered, then dividing it by two. The inverse of the throughput is thus what is actually measured.

[6] `http://docs.opencv.org/3.1.0/d9/d6d/tutorial_table_of_content_aruco.html`

Results

It is clear that Aruco performs far worse than the contenders (see fig. 4.11). Even though it is double buffered it performs worse than the single buffered alternatives. This is not surprising as the performance of Aruco motivated the development of the custom algorithm. Artoolkit performs slightly better than the single buffered custom algorithm for 12 and 24 markers. The custom scales better for more markers and has a clear advantage for 48 markers. The double-buffered custom algorithm outperforms every contender with a high margin. Arguably the most important metric for performance is how many frames takes too long to compute — causing lag. The camera frames arrive at a pace of 30 frames per second, giving the algorithms 33ms to compute the frame before lag occurs. Given that the game uses 44 markers, the double-buffered custom algorithm is the only algorithm which never fails the 33ms target. A significant improvement over the 82% failure rate of the second best (see fig. 4.11).

4.5.3. Performance test: Marker detection

Pose estimation can be a time consuming step. Artoolkit is compared to the custom algorithm for marker detection alone. Exactly the same setup as previously is used. Aruco is skipped, as it, like the custom algorithm, is based on OpenCV which uses the same pose estimation step — however without the outlier rejection. This means that Aruco's improvement will be at most as much as the custom algorithm, making it irrelevant to test given its low performance.

Results

Excluding the pose estimation lowers the computation time significantly (see fig 4.12). The custom double buffered version is still the clear winner. Artoolkit still has the advantage for 12 and 24 markers, while the custom algorithm is better for 48. The different in both cases is however smaller than with the pose estimation.

4.5.4. Tracking performance

An important metric is responsive and accurate tracking performance — if the input to the game is not accurate the game won't be enjoyable to play. It is also hard to quantify. Since the algorithms uses different markers it is not possible to feed one video stream to both and compare them. A short subjective evaluation why the custom algorithm is better for this particular use case follows:

1. The parameters of the custom algorithm are tuned assuming multiple correct markers in view. This means the sensitivity of the marker detection can be very high,

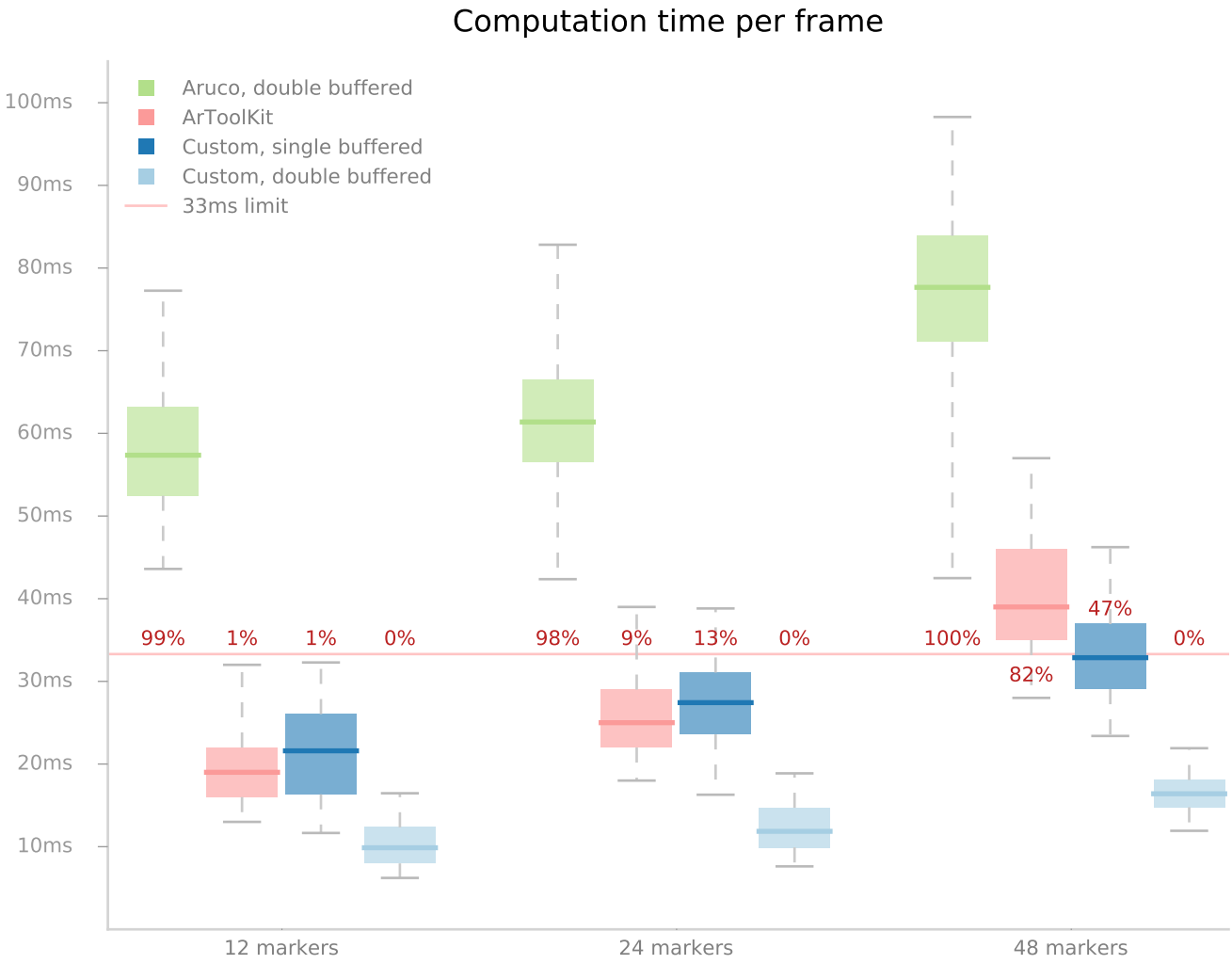


Figure 4.11.: Box plot of performance. The bottom and top of the boxes represents the first and third quartiles, with the median bar inside the box — 50% of the time the algorithm will finish within the box. The whiskers show the 2% percentile to the 98% percentile. 96% of the time the algorithm will finish within the whiskers. The red line is the 33ms mark. A computation time above this results in game lag. The accompanying red text is the percentage of frames failing to meet the 33ms deadline.

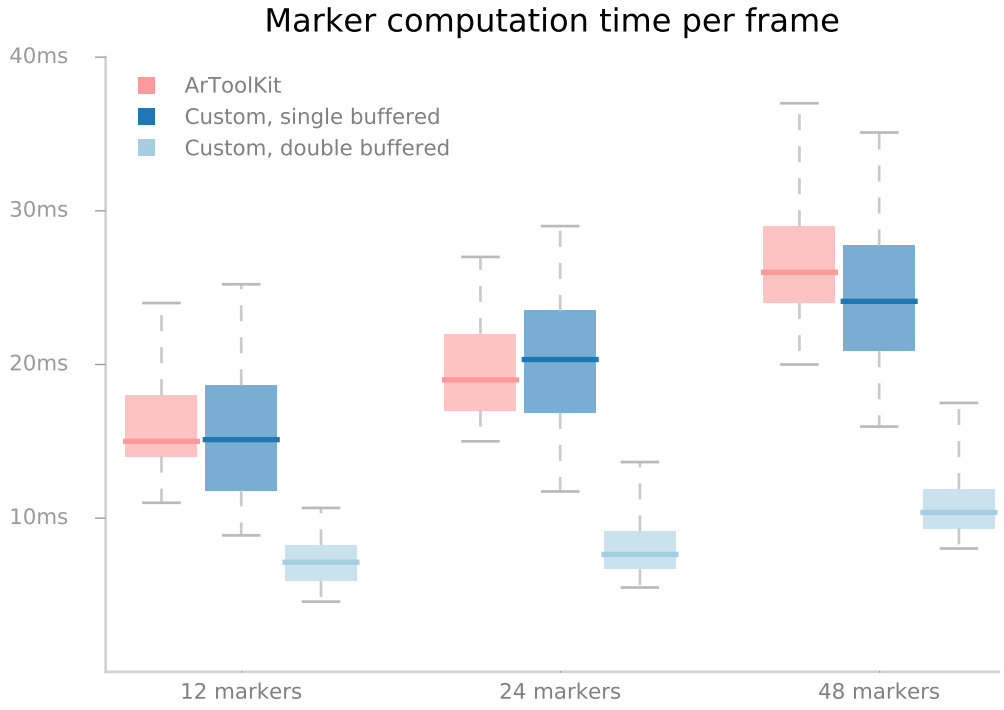


Figure 4.12.: Box plot of marker computation time. The plot uses the same graphics as fig 4.11.

giving the algorithm a high recall, at the cost of a lower precision. The pose estimation stage robustly invalidates outliers, meaning the false positive markers wont affect the result.

2. With Artoolkit the tracking performance feels like it is "on ice". The virtual objects lag behind the actual image stream. A possible explanation is that it fails to track the markers and interpolates between frames.
3. A major advantage compared to Artoolkit is that the custom algorithm uses the camera2 API. This allows the exposure time to be very low, at around 3-4ms, eliminating motion blur but increasing noise. Even for quick movements the algorithm tracks the markers very well. Due to its two pass nature the algorithm is very good at handling noisy frames.

4.6. Discussion

The marker detection algorithm provides a solution to augmented reality games on devices with constrained performance where 2D barcode markers can be used. The competing algorithms lack the speed, accuracy or responsiveness to provide for an equally enjoyable experience.

4.6.1. Future work

There are a lot of optimizations that can still be made to the algorithm, both for improving its speed and tracking performance.

Filtering A low-pass filter or ideally a Kalman filter could be added as a final step to avoid bad poses, and to interpolate when the markers cannot be found. Ideally sensor fusion with the gyroscope or accelerometer can be added to increase the accuracy. This would still allow the algorithm to be double buffered as it would be a final step after the pose has been recovered.

Pose reconstruction The pose reconstruction algorithms are directly from OpenCV and take a significant fraction of the time of every frame, compare fig 4.11 and fig. 4.12. Specializing them to the particular use case might yield better performance

Marker identification The parallel marker identification step in the algorithm uses a few slow OpenCV functions such as `findContours`. It is likely that it is possible to make a specialized version given the assumptions about the shape of the marker that is faster. Instead of computing the inner contours, and then transforming them with the homography transform, the borders of the marker cells in the original image could be computed, and the pixels counted directly.

Low level optimizations While some functions are NEON optimized, there is likely room for optimizations decreasing memory allocations and improving memory access patterns.

5. User studies

While the game has been continuously tested by several participants throughout its development, two formal user studies have been performed, one during development and one after the completion of the game.

5.1. Preliminary study: Think-aloud protocol

The first study was performed as soon as the first prototype was playable, roughly half way through the thesis work. It uses the think-aloud protocol, where participants are instructed to say whatever they think throughout the test. Each test started by a short introduction of the game and how to play it. The players participated as part of their curriculum in user usability, and were already familiar with the think-aloud protocol. Each participant played through at least two levels, one of which posed no logical challenge and were used to get the participants familiar with the controller. At this stage there were two versions of the game (see fig. 5.1):

1. The regular phone version set up on a trip-pod with its camera capturing the game board from a favorable angle.
2. A version using the Epson Moverio BT-200 augmented reality glasses.

The participants played the game on the tripod for about 5 minutes each, and were then given the option to try out the augmented reality glasses. 1 female and 5 males participated in the study.

5.1.1. Results

All participants quickly figured out how to control the game. A session typically started by the participants moving the controller around and seeing how the game reacted. Two participants initially didn't get how a move was performed (see fig. 3.2c). At this time the entire square was lit up green, with no indication that the squares had to be connected to be able to go there. However they understood it without help after testing for less then a minute. Some thought it was a bit unclear what the different colors of the overlay meant, and suggested clearer graphics. Participants noted the stuttering performance of the game (about 8 frames per second at the time), but didn't see it as a big problem for the short duration of the experiment.

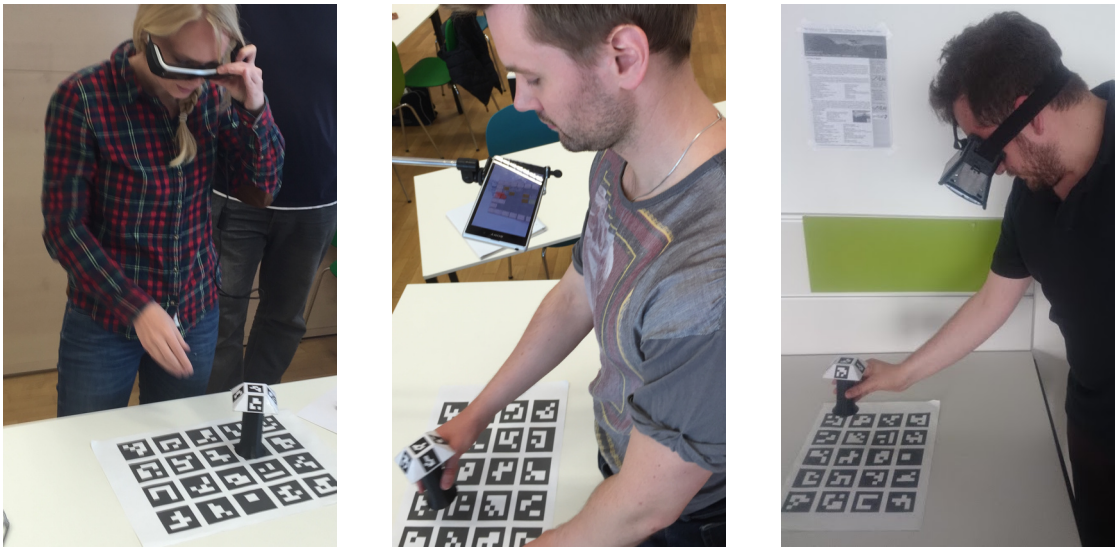


Figure 5.1.: Evaluating different technologies: 1) Epson Moverio, 2) hand-held mobile phone and 3) head mounted display (cardboard)

The female participant was the only one who preferred the Epson Moverio glasses. Likely as she was shorter and the tripod playing position was very awkward for her. Of the six participants two had not played the classic Sokoban game before.

5.1.2. Discussion

The biggest outcome of the study was that playing it with a phone on a tripod or using the Epson Moverio glasses is not ideal. It is inconvenient to have the phone in a fixed position on the tripod, especially if the participant is not tall. On the other hand it was confusing to play with the Epson Moverio glasses, as the video feed was not aligned with reality. Even if it is theoretically possible to align, it would be impractical since the slightest movement of the glasses in relation to the eyes requires a tedious realignment process.

The study promoted the development of the Google Cardboard version and the new marker detection algorithm. The graphics were also improved to be more intuitive.

5.2. Final study: Game experience

A game experience user study of the Cardboard version was carried out. The participants played up to four levels of increasing difficulty. After each level a questionnaire related to the state of mind was filled in by the participants. The participants decided the

pace of the game, including how long they would like to play. They were allowed as many restarts as they wished. After completing all levels or deciding to quit, a broader questionnaire was given about the general and augmented reality experience. Some questions are modeled after the Game Experience Questionnaire (GEQ)[24], a set of questions that measures the emotional response.

5.2.1. Purpose

The purpose of the experiment is to investigate if the augmented reality version of Sokoban is enjoyable for healthy users without any disabilities. This includes to see if the augmented reality part introduces nausea or sickness, if the controller is accurate enough and the general enjoyment of the game. If this is the case it might show potential for use in rehabilitation in the future.

5.2.2. Sample

The participants were sampled from colleagues[1] on an opt-in basis. The demographics of the study is thus well educated, technical oriented people.

10 people participated in the experiment, of which 5 male and 5 female. The demographics were two younger than 19, six between 20–29, one 30–39 and one 40–49.

Only one participant had played the original Sokoban game. 5 participants had never played a virtual reality or augmented reality game, while 5 had played it up to a few times.

5.2.3. Experiment process

1. The purpose and process of the experiment is explained. The participant signs a consent form.
2. The participant answers demographic questions and their previous experience with Sokoban and augmented reality games.
3. The Test level is demonstrated using the regular screen version. How to play the game is explained.
4. The participant puts on the Google Cardboard and plays the Test level.
5. The following is repeated for the levels Omega, Alpha, Star, TrickyThree in that order (see fig. 5.2).
    a) The participant puts on the Google Cardboard and plays the level as many times as they like. The participant may at any time request a restart of the level.

[1] Employees of Know-Center, a data science research company.

b) If the game performance suffers because of overheating, the phone is cooled with a cool pack for 30 seconds.

c) When the participant completes the level or they request to stop, they take off the headset, and the "In-Game" module of the GEQ is presented for that level.

d) Simultaneously the smartphone is cooled with a cool-pack.

6. After the experiment several questions are asked:

   a) The Post-Game module of GEQ.

   b) Augmented reality particulars.

   c) Overall impressions.

   d) Other free text comments.

7. The experiment is finished.

A session typically lasted between 30–50 minutes. See `https://goo.gl/WgVHUI` for the questions asked and a summary of the responses. See `https://goo.gl/WzfIh0` for the individual responses.

5.2.4. Results

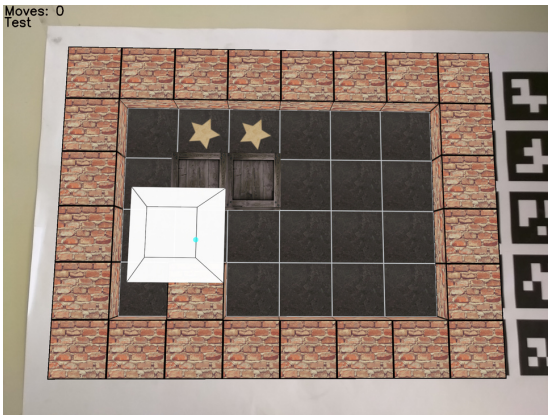
The survey indicates that the participants enjoyed the game. They rated The mean overall experience 7.2 out of 10 (higher is better). 9 answered yes to "Would you like to play the game again" (see fig 5.3).

The biggest complaint was that the smartphone overheated. This became more problematic for the harder levels, as the participants required more playtime. For the hardest level, *TrickyThree*, it was not uncommon to take one or two short pauses to cool down the phone.

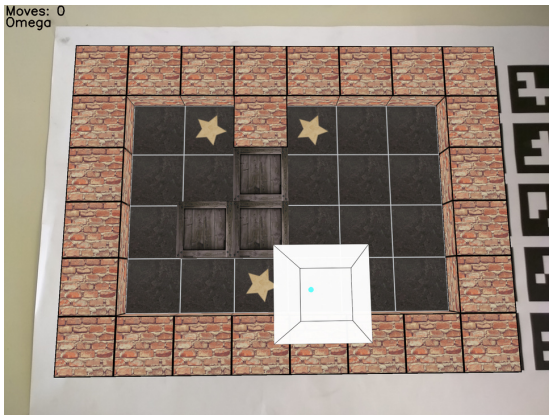
Two participants rated their augmented reality sickness as "fairly", while the median was "slightly". The comfort of the headset was rated relatively high, at a median of "fairly". The accuracy of the controller was generally perceived as high, with all but two participants rating it at least "fairly".

By inspecting the correlation[2] (see fig. 5.4) between the overall experience and various factors we can recover interesting information.

[2]Pearson product-moment correlation coefficient is a measure of the linear correlation between two variables X and Y, giving a value between +1 and −1 inclusive, where 1 is total positive correlation, 0 is no correlation, and −1 is total negative correlation. — Wikipedia

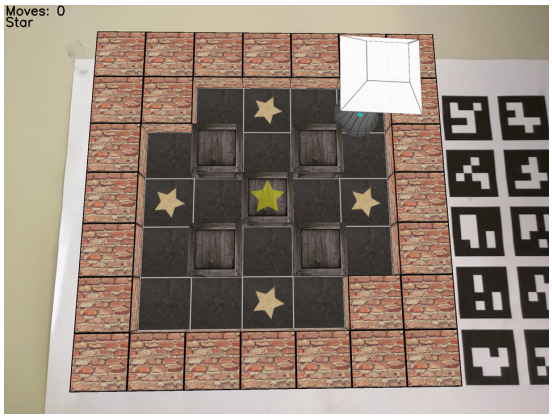


(a) Test

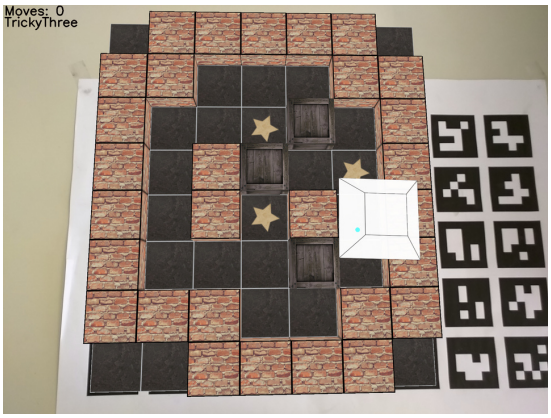


(b) Omega. All participants solved this level, everyone using 5 moves.

(c) Alpha. 9 participants solved this level, using on average 8.3 moves.



(d) Star. 8 participants solved this level, using on average 13 moves.



(e) TrickyThree. No participant solved this level.

Figure 5.2.: The levels played for the game experience study

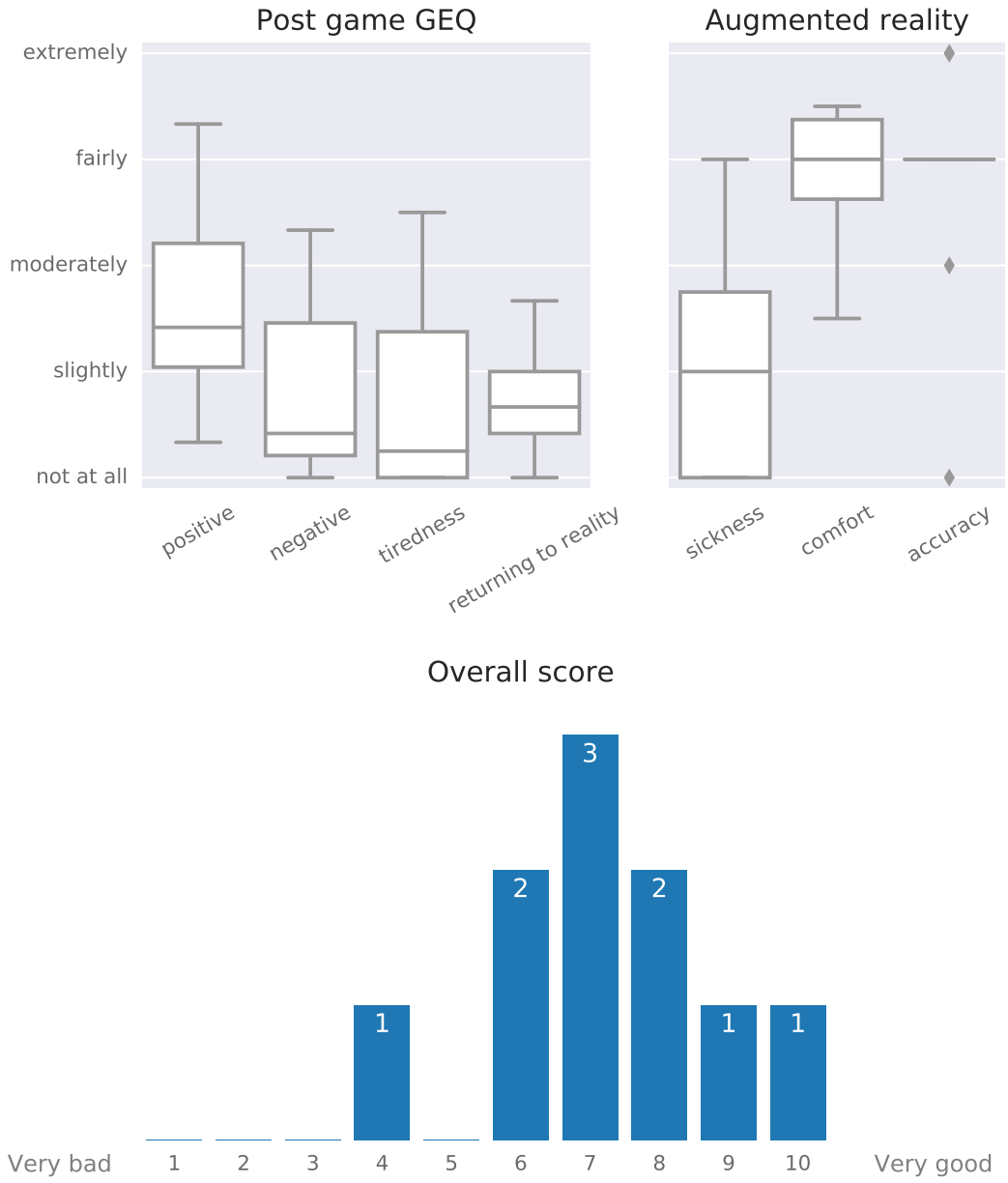


Figure 5.3.: Post game impressions.

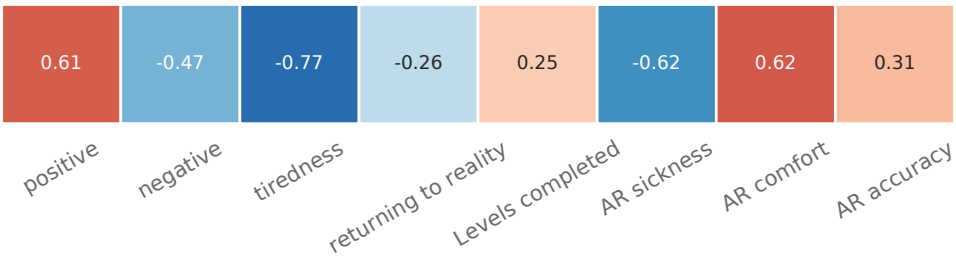


Figure 5.4.: Correlation with overall experience.

The most important factor was low tiredness, followed by low AR sickness (also know as cybersickness, motion sickness) and the comfort of the headset. There is a moderate correlation between positive emotions and inversely so for negative emotions (as computed by the GEQ).

The immersion (returning to reality), levels completed, and the AR accuracy did not have a great correlation with the overall score. It should however be noted that there was a great consensus in both the AR accuracy, see fig. 5.3, and the number of levels completed — 8 participants solved 3 out of 4 levels.

Per level results

While the test level and Omega were trivial and completed by all participants in the least amount of moves, the other levels were increasingly challenging. All except one participant completed *Alpha*, while *Star* was not completed by two. No participant completed *TrickyThree*. Participants experienced widely different emotions completing the levels, as seen in fig 5.5. While all participants agree that the challenge increases for the later levels, their reaction seemed to differ. Some expressed this positively where they became very engaged and tried hard to complete the level, while others felt frustrated and inadequate when they realized they got stuck yet again. The flow of the game seemed to improve slightly as the levels got harder. While the negative emotions are consistently relatively low, the competence and positive emotions vary widely.

5.2.5. Discussion

The experiment shows that healthy users enjoy the game. No usability problems from the first evaluation (see section 5.1) appeared. While playing the game though the Epson Moverio eyewear and tripod solution were awkward, the Google cardboard headset was comfortable enough for most participants.

After the session several participants requested more attempts for the final level outside the scope of the study — a strong indication they found the game engaging.

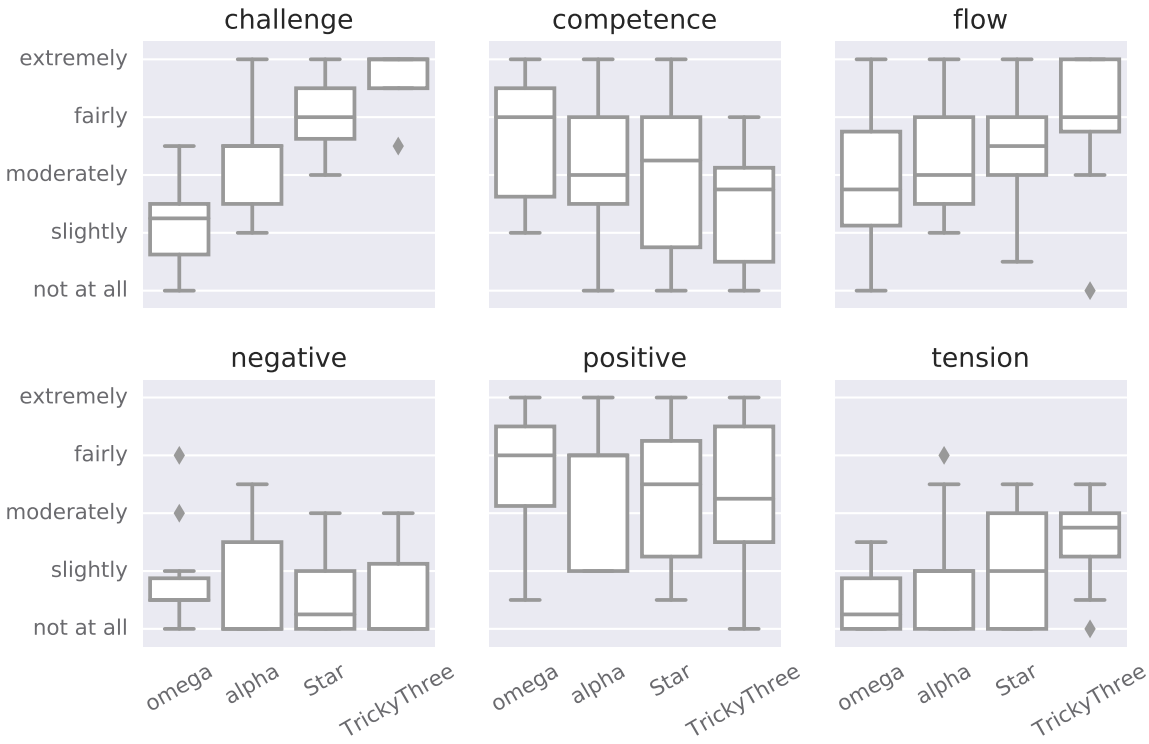


Figure 5.5.: The in-game module GEQ as completed per level.

The biggest problem reported was the overheating of the smartphone, with several participants complaining about it. Fixing the overheating — either by improving the algorithms or finding better hardware — should be the primary aim for future work.

The augmented reality experience was mostly fine, with most participants rating the accuracy as good enough, while not suffering from any nausea or other discomfort. There was one visibly nauseous participant, which might indicate that the game will never be playable by the entire population. It should however be noted that the participant was still motivated to continue solving the levels, even if she was given the option to terminate the experiment several times when displaying physical discomfort. This further indicates that the concept has potential.

6. Conclusion

The thesis successfully demonstrates that a physically controlled augmented reality game is enjoyable for healthy users. Three technologies were tried for augmenting the game to the user, a regular phone screen, smart-glasses (Epson Moverio) and Google Cardboard. The latter was shown to be the best option out of the three.

A new algorithm for marker detection is presented. With it the game can run at a continuous 30 frames per second, a task no other tested algorithm managed. Combined with the custom 3D-printed controller, the game satisfies both constraints for an enjoyable augmented reality experience — accuracy and no lag.

The user studies concludes that the augmented reality parts and the game itself were perceived positively by most users. The biggest complaint was the eventual performance degradation due to the smartphone overheating.

A shortcoming of the user study is that it did not compare against the non-augmented version of Sokoban — which might be more enjoyable than the augmented version. While the non-augmented version is not relevant for rehabilitation, it would nonetheless be interesting to compare it for healthy users.

Future work

Given the positive results of the user study, the next step would be a study on patients with spinal cord injury to evaluate if the concept carries over to grasp training. The most important focus for the application is to solve the overheating issue. Either by improving the algorithm, using better hardware, or a combination of both. If better augmented reality glasses appears on the market, it may be of interest to port the game to those devices.

Bibliography

[1] Viktoria Pammer et al. “Designing for Engaging BCI Training: A Jigsaw Puzzle”. In: *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*. CHI PLAY ’15. London, United Kingdom: ACM, 2015, pp. 667–672. ISBN: 978-1-4503-3466-2. DOI: 10.1145/2793107.2810290. URL: http://doi.acm.org/10.1145/2793107.2810290 (cit. on pp. 1, 3).

A digital game using a brain computer interface to control it.

[2] Johannes Breuer and Gary Bente. “Why so serious? On the relation of serious games and learning”. In: *Eludamos. Journal for Computer Game Culture* 4.1 (2010), pp. 7–24. ISSN: 1866-6124. URL: http://www.eludamos.org/index.php/eludamos/article/view/vol4no1-2/146 (cit. on p. 3).

About the concept of games that combines fun with learning or some other useful task.

[3] Damien Coyle, Jhonatan Garcia, Abdul R Satti, and T Martin McGinnity. “EEG-based continuous control of a game using a 3 channel motor imagery BCI: BCI game”. In: *Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB), 2011 IEEE Symposium on*. Apr. 2011, pp. 1–7. DOI: 10.1109/CCMB.2011.5952128 (cit. on p. 3).

A digital game using a brain computer interface to control it.

[4] Granit Luzhnica, Christoffer Öjeling, Eduardo E. Veas, and Viktoria Pammer-Schindler. “Technical Concept and Technology Choices for Implementing a Tangible Version of the Sokoban Game”. Poster paper to be presented at IEEE ISMAR 2016, Merida, Mexico. 2016 (cit. on p. 4).

[5] John Bowlby. *Attachment and loss. 1. Attachment*. Attachment and Loss. Basic Books, 1969. URL: https://books.google.at/books?id=FYEuAAAAMAAJ (cit. on p. 4).

[6] Adrian David Cheok et al. “Touch-Space: Mixed Reality Game Space Based on Ubiquitous, Tangible, and Social Computing”. In: *Personal Ubiquitous Comput.* 6.5-6 (Jan. 2002), pp. 430–442. ISSN: 1617-4909. DOI: 10.1007/s007790200047. URL: http://dx.doi.org/10.1007/s007790200047 (cit. on p. 4).

[7] Jilyan Decker, Harmony Li, Dan Losowyj, and Vivek Prakash. “Wiihabilitation : Rehabilitation of Wrist Flexion and Extension Using a Wiimote-Based Game System”. In: *Governor’s School of Engineering and Technology Research Journal* (2009), pp. 92–98. URL: http://www.osd.rutgers.edu/gs/09papers/Wii.pdf (cit. on p. 4).

[8] Eun Kyung Kim, Jong Ho Kang, Jang Sung Park, and Byung Ho Jung. "Clinical Feasibility of Interactive Commercial Nintendo Gaming for Chronic Stroke Rehabilitation". In: *Journal of Physical Therapy Science* 24.9 (2012), pp. 901–903. ISSN: 0915-5287. DOI: 10.1589/jpts.24.901 (cit. on p. 4).

[9] Loh Yong Joo et al. "A feasibility study using interactive commercial off-the-shelf computer gaming in upper limb rehabilitation in patients after stroke". In: *Journal of Rehabilitation Medicine* 42.5 (2010), pp. 437–441. ISSN: 1650-1977. DOI: 10.2340/16501977-0528. URL: http://jrm.medicaljournals.se/article/abstract/10.2340/16501977-0528 (cit. on p. 4).

[10] Hiroshi Ishii. "Tangible Bits: Beyond Pixels". In: *Proceedings of the 2nd International Conference on Tangible and Embedded Interaction*. TEI '08. Bonn, Germany: ACM, 2008, pp. xv–xxv. ISBN: 978-1-60558-004-3. DOI: 10.1145/1347390.1347392. URL: http://doi.acm.org/10.1145/1347390.1347392 (cit. on p. 4).

[11] Lesley Xie, Alissa N. Antle, and Nima Motamedi. "Are Tangibles More Fun?: Comparing Children's Enjoyment and Engagement Using Physical, Graphical and Tangible User Interfaces". In: *Proceedings of the 2nd International Conference on Tangible and Embedded Interaction*. TEI '08. Bonn, Germany: ACM, 2008, pp. 191–198. ISBN: 978-1-60558-004-3. DOI: 10.1145/1347390.1347433. URL: http://doi.acm.org/10.1145/1347390.1347433 (cit. on p. 4).

[12] Tamara M. Lackner, Kelly Dobson, Roy Rodenstein, and Luke Weisman. "Sensory Puzzles". In: *CHI '99 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '99. Pittsburgh, Pennsylvania: ACM, 1999, pp. 270–271. ISBN: 1-58113-158-5. DOI: 10.1145/632716.632882. URL: http://doi.acm.org/10.1145/632716.632882 (cit. on p. 4).

[13] Nesra Yannier, Kenneth R. Koedinger, and Scott E. Hudson. "Learning from Mixed-Reality Games: Is Shaking a Tablet As Effective As Physical Observation?" In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. CHI '15. Seoul, Republic of Korea: ACM, 2015, pp. 1045–1054. ISBN: 978-1-4503-3145-6. DOI: 10.1145/2702123.2702397. URL: http://doi.acm.org/10.1145/2702123.2702397 (cit. on p. 4).

[14] Harish Damodaran and Sergei Adamovich. "Examining the Manipulation of the Dynamic Properties of Virtual Objects to Optimize Upper Extremity Rehabilitation Activities." In: *Proceedings of the 2010 IEEE 36th Annual Northeast Bioengineering Conference (NEBEC)*. IEEE. 2010, pp. 1–2. ISBN: 9781424469246 (cit. on p. 4).

[15] Joseph Culberson. *Sokoban is PSPACE-complete*. 1999 (cit. on p. 5).

[16] Elan Dubrofsky. *Homography Estimation*. 2009. URL: https://www.cs.ubc.ca/grads/resources/thesis/May09/Dubrofsky_Elan.pdf (cit. on p. 13).

Explaining the Homography transform.

[17] Tom Dalling. *Explaining Homogeneous Coordinates & Projective Geometry*. URL: http://www.tomdalling.com/blog/modern-opengl/explaining-homogenous-coordinates-and-projective-geometry (cit. on p. 15).

Several concepts relating to projective geometry explained. Used for reconstructing the perspective and rendering the augmented objects with OpenGL.

[18] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: http://doi.acm.org/10.1145/358669.358692 (cit. on p. 17).

The RANSAC method of excluding outliers.

[19] Bernard A. Galler and Michael J. Fisher. "An Improved Equivalence Algorithm". In: *Commun. ACM* 7.5 (May 1964), pp. 301–303. ISSN: 0001-0782. DOI: 10.1145/364099.364331. URL: http://doi.acm.org/10.1145/364099.364331 (cit. on p. 17).

The disjoint set data structure. Also know as union-find.

[20] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. "Automatic generation and detection of highly reliable fiducial markers under occlusion". In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292. ISSN: 0031-3203. DOI: http://dx.doi.org/10.1016/j.patcog.2014.01.005. URL: http://www.sciencedirect.com/science/article/pii/S0031320314000235 (cit. on pp. 22, 30).

The first marker-detection algorithm used. Later replaced by a custom better performing algorithm.

[21] S. Garrido-Jurado, R. Muñoz-Salinas, F.J Madrid-Cuevas, and R. Medina-Carnicer. "Generation of fiducial marker dictionaries using mixed integer linear programming". In: *Pattern Recognition* 51 (2016), pp. 481–491. ISSN: 0031-3203. DOI: http://dx.doi.org/10.1016/j.patcog.2015.09.023. URL: http://www.sciencedirect.com/science/article/pii/S0031320315003544 (cit. on p. 22).

The method used to generate the Aruco markers.

[22] Linda G. Shapiro and George C. Stockman. *Computer vision*. Upper Saddle River, NJ: Prentice Hall, 2001, pp. 69–73. ISBN: 0-13-030796-3. URL: http://opac.inria.fr/record=b1128947 (cit. on p. 23).

Conventional algorithm to find connected components in a binary image.

[23] Ezio Malis et al. *Deeper understanding of the homography decomposition for vision-based control. INRIA Research Report #6303*. 2007. URL: https://hal.inria.fr/inria-00174036v3/document (cit. on p. 26).

[24] Wijnand IJsselsteijn, Yvonne de Kort, and Karolien Poels. “The Game Experience Questionnaire: Development of a self-report measure to assess the psychological impact of digital games.” In: (). URL: https://pure.tue.nl/ws/files/21666907/Game_Experience_Questionnaire_English.pdf (cit. on p. 38).

A standardized set of questions used to compute various scores for games used in the second user study.

A. Log file

For each level played a a log file is created in the "sokoban" folder on the Android device's externally accessible storage. The log file consists of two parts, one metadata part and one part for the continuous information generated by the game. The log file begins with the metadata part, ended by a blank line, after which the continuous part starts. The name of the log file is the level played and the current date.

Metadata part

File entry	Explanation
Level: Alpha	The level played.
WxH: 8 5	Number of markers (width x height) of the game board.
MarkerLengthPx: 200	The length of the game board marker and the spacing
MarkerSpacePx: 40	between them. (in pixel units)
VirtualSize: 200	The size of the virtual squares (in pixel units) One marker unit is the size of the virtual size.
PixelPerMM: 3.54333	Millimeter per pixel unit.
LiftedThreshold: 500	Threshold for the controller to be considered in lifted state. (in pixel units)
RequireSignal: 0	1 If the user was required to press space after each move, 0 otherwise.

Data

The continuous part consists of one line per event. Each event consists of multiple values, separated by space.

- The first value is always number of microseconds since the start of the level.
- The second value is a character deciding what's in the next columns, see the table below

Character	Explanation
L <x> <y>	The controller was lifted from position x, y.
P <x> <y>	The controller was put down at position x, y.
M <x> <y> <x'> <y'>	A box was moved from x ,y to x',y'.
C <n>	The game was completed in n moves.
X <v>×21	The translation and rotation of the game board, controller and relative game board. See table below.

Where the values for the X entry are:

Column	Explanation
0	Time since the start of the level (microseconds)
1	X
2	Time used to find the markers (microseconds)
3	Time used to find the pose (microseconds)
4–6	Rodrigues rotation vector for the controller (in pixel units)
7–9	Translation vector for the controller (in pixel units)
10-12	Rodrigues rotation vector for the game board (in pixel units)
13-15	Translation vector for the game board (in pixel units)
16	x position of the controller relative to the board (in marker units)
17	y position of the controller relative to the board (in marker units)
18	1 if the controller is in the lifted state, 0 if it is not.

A Rodrigues vector is a way to unambiguously describe the 3 degrees of freedom of a rotation. For further information see the Rodrigues function at `http://docs.opencv.org/3.1.0/d9/d0c/group__calib3d.html`.