Using Multiple On-Site Markers to Create Large-Scale Augmented Reality Experiences on Smartphones

Alex Maat
Supervisor(s): Baran Usta, Michael Weinmann, Elmar Eisemann EEMCS, Delft University of Technology, The Netherlands
22-6-2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology, In Partial Fulfilment of the Requirements For the Bachelor of Computer Science and Engineering

Abstract

Augmented Reality (AR) tracking for mobile devices is not realiable in environments where large virtual content, such as an entire virtual building, is to be displayed to the user. This paper presents a study on how multiple on-site markers can be used to better align the virtual scene to the real-world environment to improve large-scale AR experiences. The method consists of detecting QR markers from the video stream, and then updating the position, orientation, and scale of the virtual content in order to minimize the error between the real markers and their corresponding virtual markers. This allows for more accurate tracking and can be used in more complex environments. Three different evaluation methods are proposed. Additionally, experiments were conducted to explore the influence of marker density and marker layout. These experiments show that higher density layouts as well as structured layouts lead to higher accuracy and stability.

1 Introduction

Augmented Reality (AR) is the enhancement of the real world by adding virtual information. This often means displaying text, virtual models, icons, etc on top of the live camera feed of the real world. Over the last decade, AR has become increasingly popular with the improvements in both hardware and software [3].

Most AR applications use relatively small-scale virtual elements, like animals, furniture, text, or symbols. This is usually achieved by analizing the video stream and identifying planes by finding natural reference points. Another common approach for AR is imposing virtual content on top of a single marker. However, using AR to impose large-scale virtual content, that is to say entire buildings or streets, is less common. Large-scale AR apps allow users to explore and experience certain sites as if they were there in a different time frame.

To create an immersive and convincing large-scale AR experience, the virtual content needs to be aligned to the world properly, and move together with the camera movement. Specific hardware exists, such as Microsoft's HoloLens, built to deliver convincing AR experiences. However, the goal of this large-scale AR is to be accessible to visitors of specific (historic) sites. Therefore, the aim of this paper is to analyze the use of a multiple on-site markers that can be used to offer large-scale AR on mobile devices.

The paper first focuses on the methodoloy used for detecting and tracking multiple 2D on-site markers. The positions and orientations of these markers are then used to place the virtual content, which consists of content that is displayed to the user, as well as virtual markers used to align the content. After this is elaborated, the evaluation metrics are explained followed by experiments that use these metrics to compare the use of a multi-marker system in different contexts: the number of markers; the density of the markers (how many markers are visible at in a single frame); and the layouts of the markers (uniform or random offsets).

2 Related Work

There have been several studies relating to (multi-)marker based AR systems that are relevant for this paper.

One related study by Sing et al. was about using multiple image markers in a coloring book to allow the user to color in the drawing on their phone [8]. In this study they used the images in the coloring book as markers. In their paper, however, the multiple markers did not function as a means of better estimating the virtual position of the virtual content. Since each marker had their own content to be overlayed. In addition, Sing et al.'s application only uses the markers in the context close to the camera. However, in our paper the focus lies on using the markers in a larger environment, where the markers are further apart and further from the camera and might not all be visible to the camera at the same time.

Gherghina et al. did research on a marker-based tracking system using QR codes and its performance [5]. Since this study is from 2013, smartphone hardware is meanwhile at a much better state. Therefore the results in performance shown in their paper are slightly outdated. Additionally, their study did not focus on using multiple markers which our paper will.

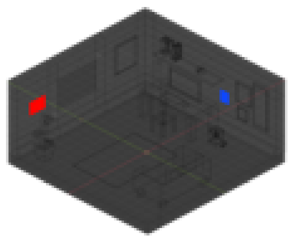
A study by Baratoff et al. focused on creating an interactive multi-marker calibration application [2]. The user first scans the area to create a model of the markers, which is then used for real-time camera pose estimation relative to the markers. The authors use pose estimation to place the virtual content. Our paper will use a different method, and instead of laying the focus on the calibration of the markers, our paper will focus on a different alignment method and three evaluation methods used for multi-marker experiments that have implications on large-scale AR.

A paper by Zauner and Haller also focused on a multi-marker calibration application for mobile devices [9]. Their paper also elaborates on how they stabilize the position and orientation of the detected markers. The alignment method explained in this paper is inspired by their work, but our paper adds quantitative evaluations in order to compare different multi-marker scenarios for implications on large-scale environments.

3 Multi-Marker Virtual Model Alignment

The goal of this paper is to present the implications of using multiple on-site markers for mobile large-scale AR applications. Therefore, we need a system that aligns the virtual AR content with the detected markers in the real world. Each location for which the application should work has a corresponding *virtual model*. This virtual model contains *virtual markers* that are placed in the same offsets from one another according to their real world positions and orientations. This virtual model also contains the *virtual content*: the visuals that should be displayed to the user, since the virtual content needs to be displayed relative to the markers. A visualization can be observed in Figure 1.

(a) *A representation of what a location in the real world may look like. The red and blue squares represent two different markers.*



(b) *A representation of the* virtual model *corresponding to the real world in Figure 1a. The markers are placed with the same positional and angular offset compared to the real world scenario.*

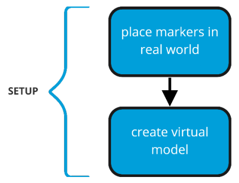
Figure 1: *Visualization of the real world and its corresponding virtual model.*

The flow of the application can be split into three parts: setup, initialize, and update. The steps for each of these three parts are visualized in Figure 2.

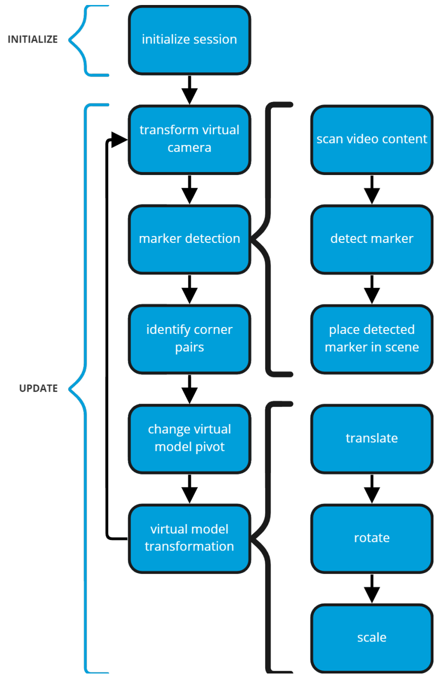
The *setup* part consists of preparing the environment. This needs to be done manually and needs to be done only once. After these steps are completed, the user can (re)run the application unlimited in the prepared area. The exact steps are explained in subsection 3.1.

The application can be used after the setup. It then starts with the *initialize* part. The details of this are elaborated upon in subsection 3.2.

The *update* part is the biggest section of the application. This part contains the main working of the alignment algorithm. The steps in this part are executed each update (frame). The update loop starts by updating the virtual camera. Which is followed by detecting markers. And finally using these markers and their correspondences with the virtual markers to transform the virtual model to align the virtual content. A detailed explanation of all the different (sub)steps in the algorithm can be found in subsection 3.3.



(a) *The flow of the setup of the application. The steps in this process need to be done manually, before the application can be used.*



(b) *The flow of the application during the initialization of the application, as well as the update loop. All these steps are handled by the application automatically.*

Figure 2: *Visualization of the flow of the application.*

3.1 Setup

Before the application can be used in a specified area, the setup steps shown in Figure 2a need to be executed. To do this, 2D markers need to be placed in the real world. The exact location depends on the experiment. An example can be found in Figure 3a.

The placed markers should have unique IDs. In this paper, this is achieved by using unique QR codes for each ID. In the Unity Editor, the images of the QR codes are added to the $ReferenceImageLibrary$ asset that contains a list of all markers as well as their physical sizes.

After the placement in the real world, the relative distances and orientations between the markers in the real world need to be measured. Then, the virtual model should be created using these measurements and the virtual markers are given their corresponding IDs. This virtual model also contains the virtual content that should be displayed to the users. The markers in the model and this content need to be placed to support the respective alignment.

(a) *An example of a setup in the real world. Two markers are placed. The right marker has ID 1 and the left marker has ID 2.*

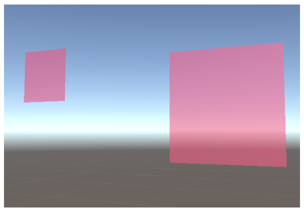

(b) *The virtual model corresponding to the real world scenario. The markers are given their respective IDs. In this whole paper, the markers of the virtual model are always indicated by red squares.*

Figure 3: *Example of the real world and virtual model setup.*

3.2 Initialize

When the setup of an environment and its virtual model are complete, the application can be run. This step corresponds to the *INITIALIZE* section in Figure 2b. At the start of the application the AR session is initialized. This AR session is managed by the ARCore implementation for Unity's ARFoundation. The Unity scene contains an *AR Session Origin* that contains a virtual camera. This virtual camera moves through the Unity scene according to how the translation and rotation is measured from the mobile device by the framework. This is done by using the smartphone's accelerometer and gyroscope [1].

3.3 Update

After the initialization is done, the update loop shown in Figure 2b starts running. This loop starts by transforming the virtual camera. As mentioned before, this is managed by the ARFoundation framework.

The next step consists of detecting the markers that are placed in the real world. The ARFoundation framework continuously processes the streaming video from the camera. When a marker is detected, it is placed in the virtual world relative to the AR session origin, where the framework thinks this marker is relative to the camera. The framework makes use of the accelerometer and gyroscope in the smartphone for this estimation [1]. The markers that are placed by the framework are now referred to as *detected markers*. It infers the orientation based on reference points on the marker. The distance between the marker and the camera is calculated by the framework by using the predefined dimensions of the markers. The position and orientation of the detected markers are continuously updated as long as the framework keeps detecting them. This usually results in tiny changes in rotation and position around an average center point of the marker. The detection an placement of these markers is visualized in Figure 4.

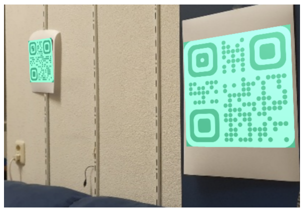


Figure 4: *When the framework detects a marker, it places a virtual version of the detected marker in the scene with the same relative position and orientation. In the whole paper, these markers are always represented by green squares.*

A *detected marker* may stay in the scene even if it is out of frame. This means it will still be considered during all calculations, it only implies that the position and orientation relative to the virtual camera is less accurate. If the framework is too unsure about a previously detected marker, this marker is removed and not considered anymore in any calculations.

After the detected markers are placed in the virtual scene, we start working on transforming the virtual model to align the virtual markers with the detected markers in order for the virtual content to be displayed at the right position. This starts with identifying corner pairs between the detected markers and the virtual markers. The real detected marker's image is unique and contains its ID. To create the corner pairs, its corresponding virtual marker is found by using this ID. Since the size, position, and orientation of the detected marker is known, the corners are calculated. This calculation works as follows:

$$\begin{aligned}\overrightarrow{c1} = \overrightarrow{marker.position} + 0.5 * width * \overrightarrow{marker.up} \\ +0.5 * height * \overrightarrow{marker.right}\end{aligned}$$

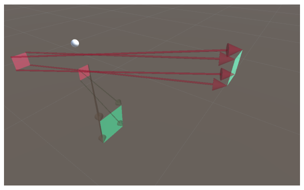
Where $marker.up$ and $marker.right$ are unit vectors of the cardinal directions in the marker's local space. The other three corners are calculated similarly, in a clockwise fashion.

The virtual model (and thus also the virtual content) needs to be translated, rotated, and scaled in such a way that it minimizes the distances between the corner pairs. Since the detected markers are managed by the framework, we can - and therefore do - only move the virtual model.

The next step consists of changing the virtual model's pivot point. This is done by moving all its children (the virtual markers as well as the virtual content) according to the offset between the current absolute position of the virtual model and the average position of all the virtual model corners that have a corresponding detected marker:

$$\begin{aligned}\overrightarrow{offset} = \overrightarrow{virtualModel.position} \\ -\overline{\overrightarrow{cornerPair.modelPosition}}\end{aligned}$$

The new pivot of the virtual model is the center of all virtual markers that have a corresponding detected marker. Note that only the detected markers are considered in the formula above, since corner pairs only exists for a virtual marker and detected marker pair. Figure 5 shows what the pivot change looks like.


(a) *The initial positions of the detected markers (green) and the virtual markers (red).*

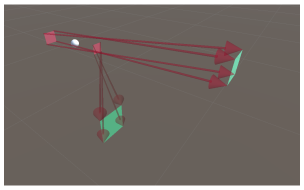
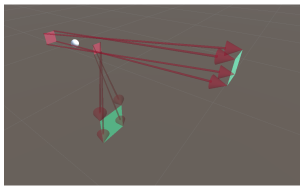

(b) *The positions of the virtual markers changed after the pivot update.*

Figure 5: *A before and after of the pivot update. The white sphere indicates the pivot of the virtual model. The red arrows indicate how the corners of the markers in the model need to be moved.*

Now that the pivot point is at the center of the markers, the virtual model is translated according to:

$$\overrightarrow{translation} = \overrightarrow{cornerPair.direction}$$

In this case `cornerPair.direction` is the vector from the virtual corner to the corresponding detected corner. As a result, the center of the virtual markers overlaps with the center of the detected markers. Figure 6 visualizes the result of the translation.


(a) *Before translation.*

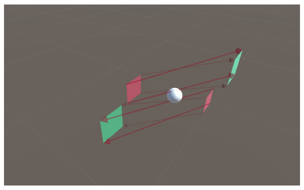

(b) *After translation.*

Figure 6: *The virtual model pivot is moved to the center of the detected markers.*

The next step is to rotate the virtual model to further reduce the corner pair errors. To get the rotation to perform, the desired rotation for each pair is calculated as a quaternion. Quaternions are used to overcome the gimbal lock of rotation matrices [7]. To then average the quaternions of all pairs, the quaternions are first used to create a sum of all the forward and up directions:

$$\overrightarrow{forwardSum} = \overrightarrow{cornerPair.desiredRotation * \overrightarrow{(0,0,1)}}$$

$$\overrightarrow{upSum} = \overrightarrow{cornerPair.desiredRotation * \overrightarrow{(0,1,0)}}$$

Both these vectors are then divided by the number of pairs to get the average forward/up combination of all the desired rotations. From these vectors, a final quaternion is calculated and this is used to rotate the virtual model.

There is one situation that breaks this calculation, which needed a special case to handle. When the direction vectors of the corner pairs cancel each other out, the resulting rotation is the identity quaternion. This happens for example when the desired rotation is exactly 180 degrees. To fix this, we change the desired rotation to a randomized small rotation (-0.1 degrees to 0.1 degrees for each euler axis) when we detect that the calculated rotation is close to resulting in no rotation. When there indeed was supposed to be no rotation, this slight rotation is not noticeable. But when the rotation is necessary, the angle keeps growing each update until it converges. A sequence of these rotations can be observed in Figure 7.

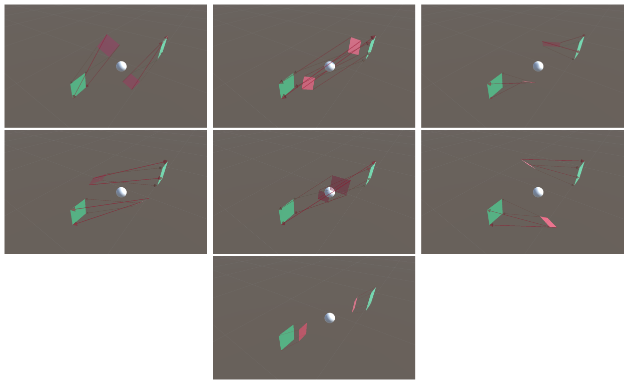


Figure 7: *The sequence of applied rotations. After a few iterations, the distance between the corner points does not decrease anymore by rotating the model.*

The last step of the update cycle is calculating and applying the right scale. For this, we again average the desired scales for all corner pairs.

For a single corner pair, the desired scale is calculated by projecting the target corner position onto the line that originates from the pivot of the virtual model and goes through the corner of the virtual marker. This line indicates the 'path' that corner takes in space when the scale of the virtual model is changed. The projected point is the point on that scale line that has the smallest distance to the target point. The formulas below calculate this closest point.

$$t = \overrightarrow{scaleLineDirection} \cdot \overrightarrow{originToPoint}$$

$$\begin{aligned}\overrightarrow{closestPoint} &= \overrightarrow{virtualModelOrigin} \\ &+t * \overrightarrow{scaleLineDirection}\end{aligned}$$

Now that we have the point on the scale line that is closest to the target corner point, we can calculate the desired scale for this corner pair. For both the current position of the corner and the closest point, the distances are calculated to the virtual model origin. Dividing these two distances gives the desired scale:

$$\begin{aligned} desiredScale = {} & distanceFromOrigin \\ & /currentDistanceFromOrigin \end{aligned}$$

A visualization of this method can be observed in Figure 8.

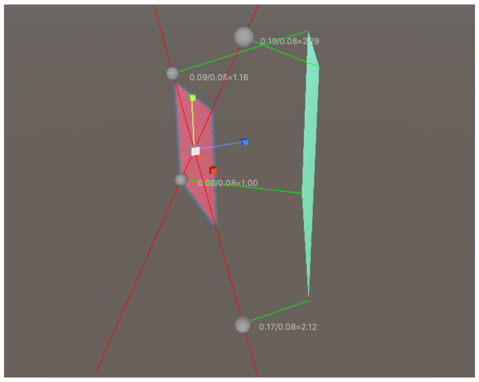
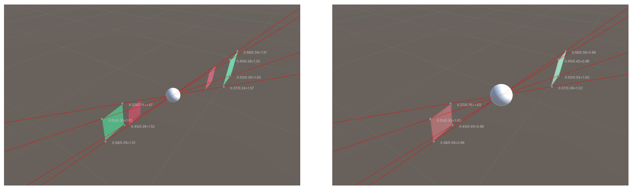


Figure 8: *A visualization of how the desired scale is calculated. The origin of the virtual model is at the center of the red marker. The red lines indicate the scale lines that originate from the virtual model origin and go through the four corners. For each corner pair, the point closest to the target corner on the scale line is shown with a white sphere. The green lines help to visualize which points correspond to which corners. The labels show what scale is required for the virtual model in order for the corner to move to that closest point.*

To get the final scale that will be applied, the average is taken from the desired scales of all corner pairs:

$$scale = \overline{desiredScale}$$

Figure 9 shows the result of changing the scale.



(a) *Before the scale is applied.* **(b)** *After the scale is applied.*

Figure 9: *Applying the scale is the last step in the alignment algorithm.*

4 Evaluation Metrics

To be able to compare different multi-marker setups to each other, different evaluation metrics are used. The evaluation is done on three distinct aspects: accuracy, stability, and performance.

Accuracy The first metric used for evaluation is that of accuracy. This metric measures the distance between the detected marker corners and their corresponding target corners. This is visualized in Figure 10. The values of all the corner pairs are accumulated and the sum is normalized by dividing by the number of pairs. The larger this number, the less the virtual scene is able to be aligned with the detected scene. The final equation is as follows:

$$\begin{aligned} accuracy = {} & |(\overrightarrow{cornerPair.targetCorner} \\ & -\overrightarrow{cornerPair.detectedCorner})|/numPairs \end{aligned}$$

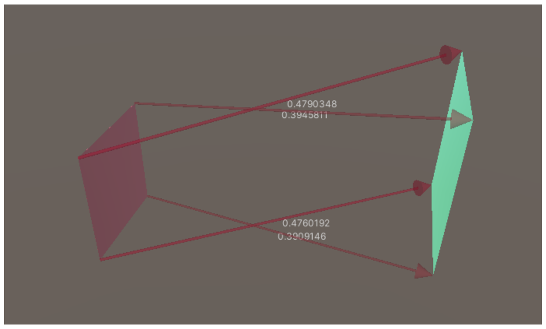


Figure 10: *Arrows are drawn between all corner pairs, indicating how the corners need to move in order to reduce the error. The numbers indicate the lengths of the arrows. The normalized sum of these lengths are used as accuracy metric.*

Stability Evaluating the stability is done in three different ways. Since the application updates the positions and orientations of all markers that are visible to the camera every frame, the application also updates the position, rotation, and scale of the virtual content. This means the virtual content is not stationary during the session.

To visualize and quantify the positional stability, we keep track of the x, y, and z coordinates of the virtual content for each frame. The smoothness and flatness of these graphs indicate the positional stability: the flatter the graph the more stable the position.

For the rotational stability, instead of tracking the absolute values of the orientation for each frame, we track the change in angle. Therefore, larger spikes indicate more rotational instability.

The last stability metric is done by tracking the scale. Since the virtual content is always scaled homogeneously, we only have to track a single number for the scale for each frame.

Performance The performance of the algorithm is quantified by measuring the number of ticks between the start and end of the algorithm, during each frame. A C# tick is the smallest unit of time, equal to 100 nanoseconds [4].

5 Experiments and Results

The experiments in this section have all been performed with the same hardware and software configuration. The application is run through the Unity Editor. It makes use of the AR Foundation Editor Remote package in order to run the experiments wirelessly on a smartphone through the editor [6]. The Unity Editor was run on a Windows 10 laptop with the Intel i7-8750H CPU and 16GB RAM. The smartphone used in the experiments is the Samsung Galaxy S10+ with 128GB of storage.

For the first experiment we analyze the performance, accuracy, and stability based on the number of detected markers. The evaluation methods used are described in section 4. The real-world environment used for the experiment consists of four markers, one on each wall of a square room of 20m².

Figure 11 shows the measured performance during the run of the application. This shows that there is a clear correspondence in algorithm duration based on the number of detected markers, since the minimum bounds of the number of ticks increases each time a marker is added. Since the jumps of the minimum bounds are of the same spacing each time, the algorithm performs in $O(n)$ time, where n is the number of detected markers. This makes sense, as the algorithm needs to perform calculations for all corner pairs of the markers for each of the steps.

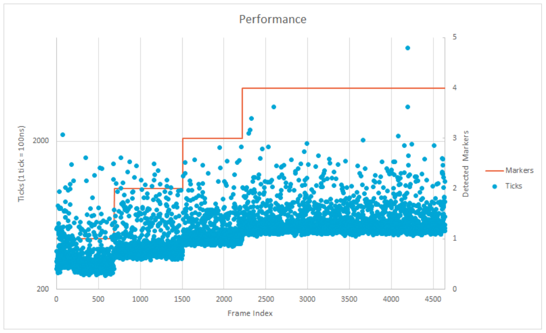


Figure 11: *The runtime of the algorithm measured during one session. The minimum bounds of the runtimes is clearly dependent on the number of detected markers.*

A visualization of the accuracy measurements are shown in Figure 12. At the very start of the application, the accuracy error starts relatively large, and then rapidly shrinks during a few frames. This happens since the algorithm takes a few iterations for it to settle on the final result. After these first few frames, during the rest of the frames where only one marker is detected, the accuracy is the highest. This is because the algorithm only takes the one detected marker into account. So it can translate, rotate, and scale the virtual model to fit that one marker perfectly.

When a second marker is detected and thus considered in the algorithm, the accuracy makes a sudden jump. The initial spike happens because the first few iterations in updating the rotation are less accurate. After a few updates, the accuracy settles. This plateau is at a lower accuracy than the plateau with only one detected marker. This occurs because of imperfections between the matchups in the virtual model and the real environment. Because of this the error cannot be reduced any further. This is, however, expected and exactly the reason why we average and use the information from all markers. The same behaviour can be observed when the third marker is detected.

The detection of the fourth marker, however, shows different behaviour. At first the error settles at a higher value. This happens due to a limitation of the current implementation. The framework wrongly updated one of the markers that was out of sight and therefore the algorithm could not properly align the virtual model. A few seconds later the wrongly detected marker is in frame again and now updated properly, reducing the error and settling for the same error as with three markers.

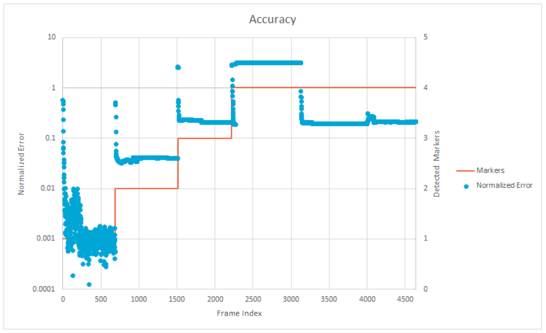
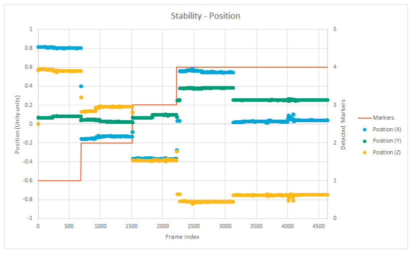


Figure 12: *The accuracy of the alignment measured during one session based on the number of detected markers.*

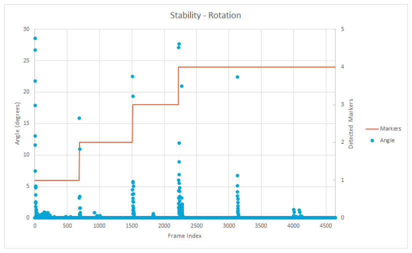
The last aspect considered for this environment is the stability. Figure 13a shows that the position changes when a new marker is detected. The change in position around frame 3200 happens because of the correction of the misdetection as explained earlier. When looking at the overall graph, it looks like the position changes a lot and thus the virtual content would seem to make big jumps during each new marker. In practice, this effect is less noticeable, since the measurements of these positional coordinates do not take into account that the pivot of the virtual model is changed.

The stability of the rotation is measured by the change in angle. This is shown in Figure 13b. The spikes only occur at the start of detecting a new marker (except for the spike around frame 3200). It then takes a few frames to stabilize the rotation.

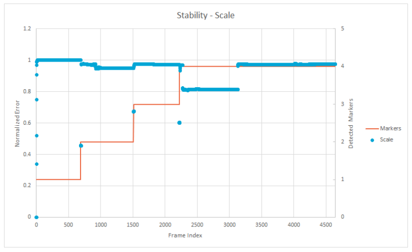
The stability of the scale shows similar behaviour as can be observed in Figure 13c. Since the scale is updated after the rotation, and the rotation takes a few updates to settle, the scale changes quite a bit during this period. However, after the rotation has settled, the scale settles as well and stays stable.



(a)



(b)



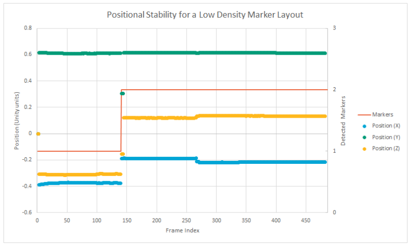
(c)

Figure 13: *The stability of the alignment during one session based on the number of detected markers. The subfigures show the stability of the position, rotation, and scale respectively.*

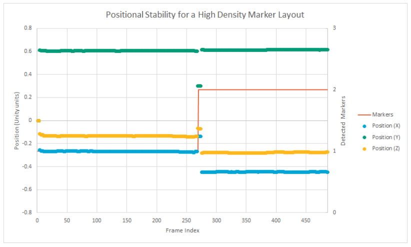
Next we look into the influence of the density of the markers on the positional stability. For this experiment we had two different real-world setups: a low density setup and a high density setup. For the setups we placed three markers next to each other, at the same height, with a spacing of 70cm and 35cm respectively.

Figure 14 shows the positional stability in both setups. There are two things to note when comparing the two graphs. The first is that the position in the high density setup stays more stable when there is no change in the number of detected markers. In Figure 14a the y and z coordinates make a jump around frame 260. Similar jumps are not observed in the high density setup in Figure 14b.

The other remark is that the change in position is smaller in the high density scenario. The jump for the z coordinate in Figure 14a is ≈ 0.45, while the jump in Figure 14b is ≈ 0.2. The reason for this is that the alignment of the virtual model gets fresh input more frequently with higher densities. Since new markers are detected more often in the same amount of space, smaller corrections in the alignment are required.



(a) *Positional stability for low density setup.*



(b) *Positional stability for high density setup.*

Figure 14: *Comparison of the positional stabilities in both a low density and high density marker setup.*

For the last experiment we look at the influence on the accuracy of two different layout types: structurally spaced, and randomly spaced. With a structurally spaced layout, it is meant that the relative distances between the markers are the same for each pair. In practice, this yielded in a layout where three markers are placed on a flat wall with a spacing of 70cm. The randomly spaced layout places the markers at more obscure positions in the environment, such as corners, where they do not distract users as much. For this experiment, this yielded in three markers placed on opposing walls.

As is visible in Figure 15, the accuracy in the structurally spaced layout already settles after the second marker is detected. When the third marker is also detected, the accuracy stays the same. On the other hand, in the randomly spaced layout, the error increases both at the detection of the second marker, as well as the third. The reason for this difference is likely that with a structurally spaced layout, the virtual model is easier to be made more accurately.

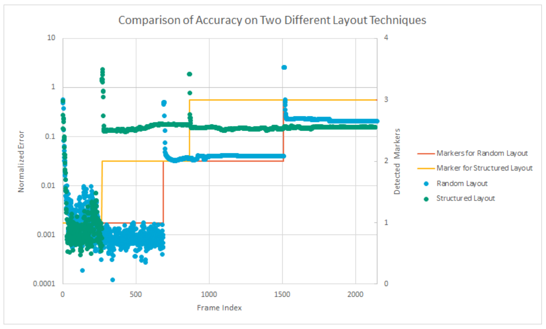


Figure 15: *A combination of the accuracy graphs for both a uniformly spaced layout and a randomly spaced layout.*

6 Discussion

The method shown in this paper has room for improvements. The first of these improvements is the ability to identify the markers better. Since the markers are QR codes that can look quite similar to each other, especially from further distances, the framework might misidentify a marker and therefore the alignment with the virtual content has a larger error. This would need more research on what visual content the markers should use to improve identification.

Another issue is the issue of misdetections. Sometimes the detections of the markers are not accurate enough to provide for a correct alignment. This could be improved by ignoring markers that are not in frame, or by ignoring markers that are outliers when compared to the positions in the virtual model.

Currently, the user needs to be quite close to a marker for the marker to be detected and identified. This breaks user immersion as the user sometimes need to manually make sure a marker gets detected. Therefore, this method currently works best when the user will be using the application close to the markers.

7 Responsible Research

Responsible research consists of two main aspects: ethicality and reproducability.

With ethicality, concerns of discrimination or biases play a role. Since the research for this paper did not include user studies, there is no issue of implicit biases that may be present in user study groups. Additonally, the system presented is only a prototype used to perform and analize experiments. Therefore, concerns in fair user interaction are not applicable.

The reproducability of the research is ensured by including comprehensive explanations of how the systems work in section 3 and what limitations they have in section 6. Additionally, the environment of the different experiments are also made clear by stating what hardware and software were used as well as describing the real world environments that were used. Both of these can be found in section 5.

8 Conclusion

This paper presented a method for multi-marker AR alignment. The method uses averaging of the position, rotation, and scale to align the virtual content with the real world markers. Additionally, it elaborated on three different evaluation techniques to evaluate the accuracy, stability, and performance.

The paper presented experiments on the implications of multi-marker AR on large-scale AR. These experiments showed that using markers spread throughout the environment can be used to give the application a fresh update on the position of the user. Which in turn realigns the virtual content for better accuracy. We also showed that the runtime of this algorithm is $O(n)$, where n is the number of detected markers. Additionally, it was shown that a higher density of markers in the environment yields more stability. Furthermore, layouts of uniformly spaced markers outperformed randomly spaced markers due to the imperfections in measurements during the setup of the environment and corresponding virtual model.

The implementation of the method has room for improvement. Especially the marker detection and identification part, since this is the current bottleneck for using the application on larger distances from the markers.

The proposed alignment method in combination with the conducted experiments allowed for a base of large-scale marker-based AR as well as insights on marker distribution and layout, which benefits the future of large-scale AR.

References

[1] Intro to AR Foundation — dots-tutorial.moetsi.com. https://dots-tutorial.moetsi.com/ar-foundation/intro-to-ar-foundation, 2021. [Accessed 19-Jun-2022].

[2] G. Baratoff, A. Neubeck, and H. Regenbrecht. Interactive multi-marker calibration for augmented reality applications. In *Proceedings. International Symposium on Mixed and Augmented Reality*. IEEE Comput. Soc, 2002. doi: 10.1109/ismar.2002.1115079. URL https://doi.org/10.1109/ismar.2002.1115079.

[3] Yunqiang Chen, Qing Wang, Hong Chen, Xiaoyu Song, Hui Tang, and Mengxiao Tian. An overview of augmented reality technology. *Journal of Physics: Conference Series*, 1237(2):022082, June 2019. doi: 10.1088/1742-6596/1237/2/022082. URL https://doi.org/10.1088/1742-6596/1237/2/022082.

[4] dotnet bot. TimeSpan.Ticks Property (System) — docs.microsoft.com. https://docs.microsoft.com/en-us/dotnet/api/system.timespan.ticks?view=net-6.0#remarks, 2022. [Accessed 18-Jun-2022].

[5] Alexandru Gherghina, A. Olteanu, and N. Tapus. A marker-based augmented reality system for mobile devices. In *2013 11th RoEduNet International Conference*. IEEE, January 2013. doi: 10.1109/roedunet.2013.6511731. URL https://doi.org/10.1109/roedunet.2013.6511731.

[6] Kyrylo Kuzyk. AR Foundation Editor Remote — Utilities Tools — Unity Asset Store — assetstore.unity.com. https://assetstore.unity.com/packages/tools/utilities/ar-foundation-editor-remote-168773, 2022. [Accessed 18-Jun-2022].

[7] E.E.L. Mitchell and A.E. Rogers. Quaternion parameters in the simulation of a spinning rigid body. *SIMULATION*, 4(6):390–396, June 1965. doi: 10.1177/003754976500400610. URL https://doi.org/10.1177/003754976500400610.

[8] Angeline Lee Ling Sing, Awang Asri Awang Ibrahim, Ng Giap Weng, Muzaffar Hamzah, and Wong Chun Yung. Design and development of multimedia and multi-marker detection techniques in interactive augmented reality colouring book. In *Lecture Notes in Electrical Engineering*, pages 605–616. Springer Singapore, 2020. doi: 10.1007/978-981-15-0058-9_58. URL https://doi.org/10.1007/978-981-15-0058-9_58.

[9] Jürgen Zauner and Michael Haller. Authoring of mixed reality applications including multi-marker calibration for mobile devices, 2004. URL http://diglib.eg.org/handle/10.2312/EGVE.EGVE04.087-090.