Guided Assembly with Augmented Reality Upgrade

Submitted April 19, 2023, in partial fulfillment of
the conditions for the award of the degree **BSc (Hons) Computer Science with Artificial Intelligence.**

Tianxiang Song

20217424

scyts1@nottingham.edu.cn

Supervised by Dr Boon Giin Lee

School of Computer Science
University of Nottingham Ningbo China

Abstract

Augmented Reality (AR) has a huge potential on assisting manual assembly tasks by overlaying digital contents on the camera view as instructions. Compared with paper-based instructions, AR provides an interactive experience, as well as improvements in assembly accuracy and speed. However, AR-guided assembly on small pieces has been a challenging task. The small size of target object could reduce visibility of 3D model-based AR instructions, and the virtual-physical misalignment may cause assembly errors. Handling object occlusion on the digital content and detecting complete assembly steps for instruction automation are also limited by the lack of features in small objects. Thus, advancing AR solutions for guiding small piece assembly is of great significance.

In this work, we proposed several methods to improve the performance of AR-based assembly guidance system, implemented as a mobile application, and evaluated using a LEGO set to simulate small assembly pieces. The AR instruction visibility is enhanced by a combination of rendering modes such as texture, wireframe and animated preview. The registration accuracy achieves an average alignment error of less than 1mm through a flexible marker design and vision-sensor hybrid approach. Precise object occlusion on the virtual content is accomplished utilizing high-fidelity digital model and virtual-physical alignment to generate occlusion masks in real-time. Additionally, a unique instruction automation method is proposed to trigger the next AR instruction by detecting complete assembly steps, focusing on classifying two states in each step - Finished (Integrated) and Unfinished (Non-integrated), based on coincidence degree between digital wireframe and object contours. The system demonstrates a 92% triggering rate on 100 finished assembly steps, trained on images of only 10 assembly pieces out of 468. The major finding of this project is that the integration of these techniques makes AR-based guidance system reliable for small piece assembly, validated by working on a complex LEGO set.

Acknowledgements

I would like to express my highest gratitude to Dr Boon Giin Lee, my project supervisor, for his kind attitude, valuable feedback and constructive suggestions on every stage of the work. Some of the achievements could not have been done without his support.

I would also like to thank Dr Matthew Pike, the module convener, for excellent course arrangement, and staffs from the Writing Lab, for useful guidance on academic writing.

Additional appreciation goes to Prof. Wei Yan at Texas A&M University for his brilliant work in the subject area that offers me several ideas on the project. Special thanks extend to my family and my girlfriend Ms Yujiao Xiao, for company and encouragement that firmly supports me during hard times.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
CAD	Computer-Aided Design
DoF	Degrees of Freedom
FPS	Frames Per Second
GUI	Graphical User Interface
HHD	Hand Held Devices
HMD	Head Mounted Devices
IMU	Inertial Measurement Unit
SLAM	Simultaneous Localization And Mapping
UI	User Interface
UX	User Experience
VIO	Visual Inertial Odometry

Chapter 1

Introduction

This chapter introduces the background of Augmented Reality applied in manual assembly tasks, followed by motivation, aims and objectives of this project.

1.1 Background

Augmented Reality (AR) is the overlay of digital information on real scenes, and relevant techniques has made significant progress in the last three decades [1]. With its huge potential in improving manufacturing processes, AR tends to play an increasing role in the industry [2]. AR assembly studies have shown that one benefit of superimposing virtual objects on the real world is they assist in guiding the positioning of different assembly components [3], which accelerate manual assembly tasks [4, 5] and increase assembly accuracy [6]. By replacing tedious and detailed instructions with 3D models, animations and operation feedback, users could focus on assembly task itself. Richardson et al.'s experiment illustrated that tablet based AR instructions helped users to assemble with eight times greater quality and 33% less time when compared to traditional paper based instructions, and the reduced travel about the work cell granted them more time focusing on the job [7]. Another research further reveals that assembling experience in sophisticated subjects such as a robot module could be advanced by the AR approach, which is more effective and enjoyable than plain instructions [8].

The AR guidance system could be used for different stages in the assembly process. An tutoring system developed for Microsoft HoloLens is able to present a virtual "tunnel" indicating which bin a desired part is placed, demonstrating time-efficiency and picking accuracy in kit preparation [9]. Another application utilizes a tablet screen to present step-by-step AR assembly instructions for a wind turbine and solar panel, outperforming paper instructions without the need for external documentation [10].

Regarding target assembly tasks, the contributions could be subdivided into "real industrial" and "demonstrative" cases [11]. While the former concerns the construction of an AR system for assisting a real industrial challenge, such as assembly of a car engine or a motherboard [12], the latter refers to the experimental presentation of operations using basic parts or toys, employed as benchmarks or a simulation of industrial assembly [13].

1.2 Motivation

In the industry, assembling small pieces has been a challenging task [14]. For example, furniture or robot assembly sometimes involve tiny pieces that are difficult to identify, which can cause misunderstanding following paper-based instructions [15]. Therefore, an AR system that guides small part assembly is naturally worth investigating.

However, some of the limitations in AR techniques may make AR instructions unreliable for small part assembly. Hoover et al. demonstrated that the position of digital objects in Microsoft HoloLens did not align properly with corresponding physical parts, due to the lack of accuracy in image registration [16]. The misalignment problem could be more serious in small part assembly, leading to assembly errors [17]. The appearance of AR content is another concern. For model-based AR instructions, in addition to lower instruction visibility when the size of corresponding physical part is small, the proper spatial relationship between the digital content and nearby physical objects is difficult to reveal [18, 19]. Due to small depth variation, it is challenging to determine whether the virtual model should be occluded by neighboring physical objects [19]. Furthermore, detecting complete assembly steps to trigger the next AR instruction may be a greater challenge, considering the lack of features in small assembly pieces [20].

Thus, it is significant to improve relevant techniques to build a reliable and effective AR tutoring system for small part assembly. To perform this study, a specific LEGO set is chosen for simulating the challenges. LEGO block is a widely used toy to evaluate the performance of AR-based assembly guidance system [1], and a common choice is Duplo, a larger version of standard LEGO, considering general sizes of assembly pieces [18]. In this work, we use a standard LEGO set to demonstrate whether the proposed method could advance AR guided assembly on small pieces.

1.3 Aims and Objectives

To address the limitations in AR techniques as discussed in Section 1.2, the main objective of this project is to improve relevant methods, and develop an AR tutoring system to assist manual assembly tasks on the LEGO sets, as a simulation of industrial assembly on small pieces. The system is planned to be built as a mobile application, and the desired result will be a reliable AR-based system that guides small piece assembly.

The key objectives of this project include:

1. Investigate and analyze relevant techniques used in the subject area, including AR registration, occlusion handling and instruction automation.
2. Implement specialized methods for AR-guided assembly tools to improve instruction visibility, registration accuracy and occlusion efficiency. Design an auto-guidance method to present the next AR instruction by detecting complete assembly steps.
3. Evaluate performance of proposed methods and the whole application, analyze test result and validate system specifications.

Chapter 2

Related Work

This chapter outlines major studies and methodologies in AR solutions for guided assembly. In general, relevant research areas involve the presentation of AR content, depth perception for occlusion handling, and the automation of AR instructions.

2.1 AR Content

AR content overlays contextually relevant digital information, such as images, videos, or graphics, onto the real-world environment [21]. This section investigates several types of AR contents as assembly instructions, and methods to align them properly with relevant physical objects.

2.1.1 Model-based AR Instructions

There are several ways to present virtual instructions in an AR application that guide construction or training tasks, including 2D figures and 3D models [11]. However, a recent study regarding a 2D projection-based AR tutoring system demonstrates limited benefits in the training scenario, in which the speed, recall and precision does not reach personal-instructed training after 24 hours [22]. Meanwhile, it is shown by many studies that 3D model-based AR instructions could significantly accelerate assembly tasks and reduce error rate, with high fidelity 3D models of the corresponding physical parts [6, 7].

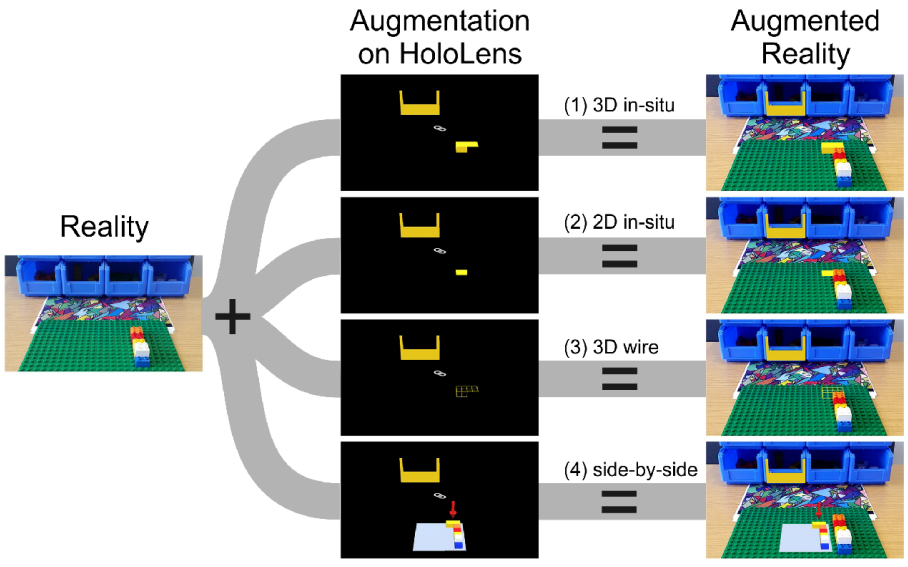


Figure 2.1: Different AR instructions in HoloLens [18]

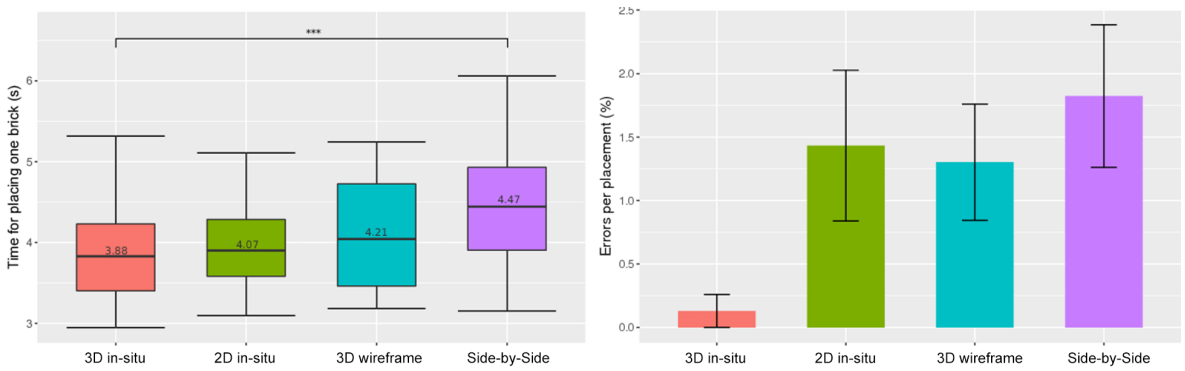


Figure 2.2: Comparison of different AR instructions on user performance [18]

Figure 2.1 illustrates several types of AR instructions in a LEGO assembly scenario using Microsoft HoloLens, and their impact on user performance is demonstrated in Figure 2.2. It is shown that compared with other AR instructions, 3D model (in-situ) methods which projects the 3D model of physical pieces into the exact position where they should be, could significantly reduce error rate and make construction faster. The side-by-side AR methods, however, result in poor user performance. Thus, 3D model-based instructions could be further advanced by presenting virtual models in the right place.

2.1.2 AR Registration

In AR applications, virtual contents usually need to be aligned with physical models [23], thus it is necessary to track the position and orientation of objects in the environment. Two vision-based approaches are widely employed, namely marker-based registration, which tracks artificial markers (such as the QR code) to calculate nearby object pose, and marker-less registration, which uses features of the object itself [11].

The registration accuracy is important in 3D model-based AR instructions as virtual models require transformation in right location, scale, and orientation [17]. The LEGO bricks, for example, are very small in size, thus low registration accuracy will result in significant misalignment between digital and real models, which might cause errors in construction [17]. One study shows that marker-based solutions are usually preferred in guided assembly because of higher accuracy compared with marker-less approaches [25]. In addition, marker-less solutions usually track a feature-rich object as demonstrated in Figure 2.3, which is hard to realize on small objects [24]. Figure 2.4 is an example of utilizing 2D image marker (left, constructed using LEGO pieces) as AR anchors to align virtual content (the green wireframe) with the LEGO model. However, constructing the image marker using LEGO pieces may be inconvenient due to additional effort in obtaining

Figure 2.3: Marker-less AR registration [24]

Figure 2.4: Marker-based AR registration [17]

the "marker" pieces and the LEGO assembly base.

Once the marker is detected, there are several ways to calculate the position and orientation of virtual objects based on estimated camera pose relative to the marker. Homography-based methods estimate the object pose by mapping the marker to a virtual plane [26]. PnP-based methods use the 3D coordinates of points on the object and their corresponding 2D image coordinates to estimate the pose [27]. Feature-based methods use feature extraction techniques such as SIFT [28] to match corresponding points between the marker and the camera image [29]. However, poor quality input image can lead to inaccurate pose estimates by these methods.

2.2 Depth Perception

Depth perception refers to the ability of an observer to perceive and differentiate distances between objects in a scene [30]. In AR applications, depth perception is critical for creating realistic and engaging experiences [31]. This section introduces two types of occlusions that can arise in AR applications, which pose unique challenges to depth perception.

2.2.1 Object Occlusion

Virtual contents in AR applications appear in front of the camera view by default. However, in the assembly process, correct depth perception is essential as some of the pieces might be installed behind other parts [25]. Object occlusion is a way of differentiating depth of virtual contents and physical objects, to ensure realistic and immersive

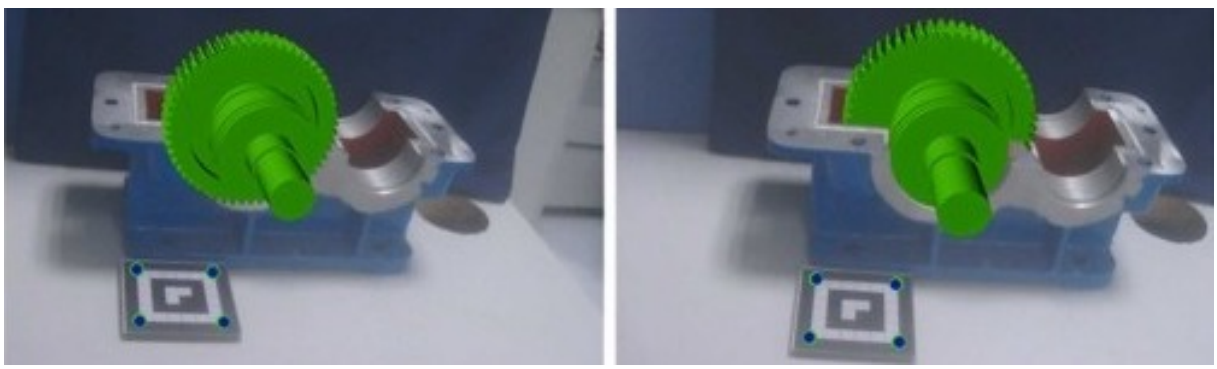


Figure 2.5: Object occlusion handling based on monocular SLAM [19]

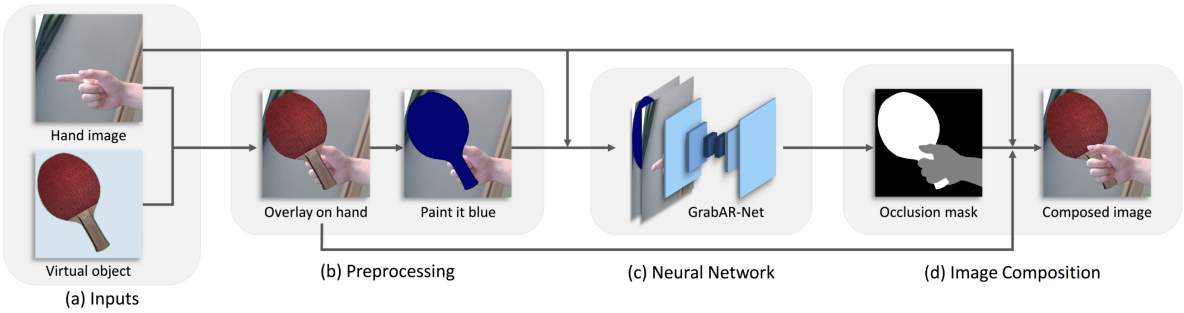


Figure 2.6: Demonstration of hand occlusion in AR application [34]

user experiences [25]. Figure 2.5 demonstrates how occlusion handling (right) could help model-based AR instruction look more realistic and less confusing.

Several object occlusion methods were proposed in the last decade. Occlusion-by-contours was used to remove ambiguity about depth perception [32], however, the accurate spatial relationship between virtual and physical objects are not revealed realistically by the contour lines. Another study uses monocular SLAM to reconstruct the assembly environment into sparse 3D points, which are then converted into depth points in a depth map for computing occlusion masks on the virtual object [19]. A recent study uses six degree of freedom (6-DoF) trackers by camera and inertial sensors to estimate a sparse depth map via stereo matching [33], demonstrating low latency on smartphones.

2.2.2 Hand Occlusion

Similarly, hand occlusion aims to present correct spatial relationship between virtual contents and user hands. The majority of vision-based methods employ a single monocular camera to track users' hands, which is highly adaptable for use with mobile devices [34]. Chun et al. proposed a simple hand identification approach for AR interactions [35], while Choi et al. established the 6-DoF pose of a hand gesture for virtual object overlay [36]. On the other hand, Song et al. utilized an RGB camera to detect hand shapes, estimate the average hand-to-camera distance to obtain occlusion mask on the digital content [37]. Nevertheless, these efforts have not yielded sufficiently precise results to resolve the hand-object occlusion.

Recently, deep learning techniques have shown promising results in hand occlusion from complex and cluttered backgrounds, due to the efficiency in hand segmentation using structures like Refined U-Net [38], Mask R-CNN [39], DeepLabv3 [40] and EgoGAN [41]. Figure 2.6 demonstrates a study on CNN-based hand occlusion, which directly predicts the occlusion mask based on hand pose, without the need to calculate depth [34]. However, this method requires complex synthetic data for training a depth-sensitive model.

2.3 Instruction Automation

Automating instruction in AR assembly tools aims to detect completed steps and providing instructions for the next task, without direct operation on the user interface [42]. This section analyzes previous works in two main research areas for AR instruction automation.

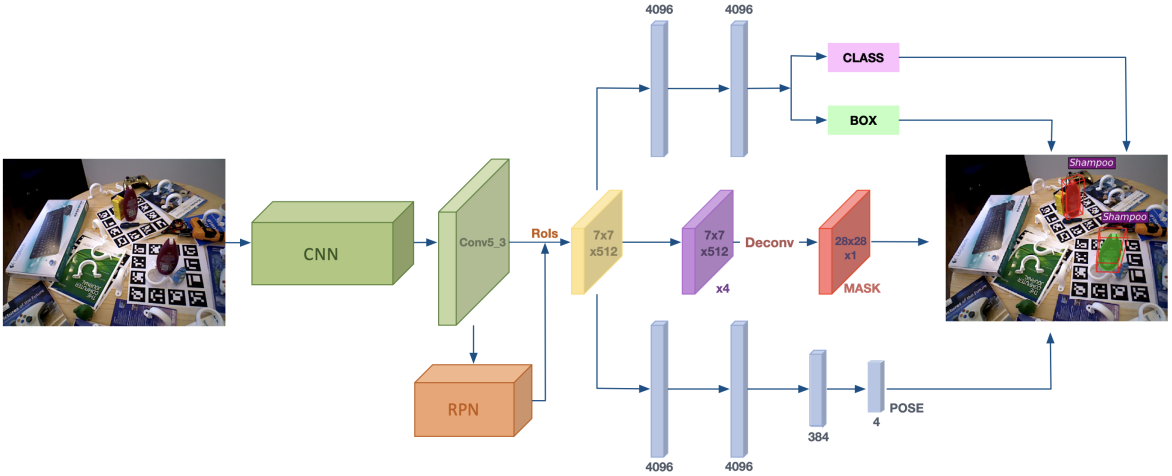


Figure 2.7: Example of CNN-based object pose estimation [48]

2.3.1 Piece Recognition

Piece recognition could be used to trigger the next AR instruction based on whether a current assembly piece is placed to its correct position. Traditional detectors identify and compare hand-crafted features, however, although certain features have been intentionally designed to withstand difficulties such as variations in scale or lighting, they still have restricted effectiveness with textureless objects [43].

Recently, two main types of CNN-based detectors are used for piece recognition. YOLO [44] and SSD [45], both one-stage detectors, possess a simplified design which allows for high frame rates of over 50 FPS. These detectors predict the probabilities of the class and the bounding box, utilizing predefined anchors on the image. On the other hand, two-stage detectors such as Faster R-CNN [46] and its subsequent works [47], employ a Region Proposal Network (RPN) that generates potential bounding boxes depending on the feature map. A fine regression of the position and precise classification of the object will be carried out based on the bounding boxes. A recent work proposed a piece recognition algorithm based on Faster R-CNN [20], however, the process of training data gathering is time-consuming as images of every piece need to be collected sufficiently. In addition, dealing with duplicated pieces may result in misclassification of an unaccomplished step, when the previously assembled pieces being identified.

2.3.2 State Estimation

While piece recognition focus on identifying an object's presence and location in a 2D image, state estimation concerns the 6-DoF pose of the object [49], as illustrated in Figure 2.7. The estimated pose of the whole physical object is used to determine the current state of assembly task and present the corresponding AR instruction. One study uses the transformation matrix to estimate the pose of objects and analyze their movement during assembly tasks, allowing for better understanding of user actions [50]. However, it is sensitive to noise and small errors, which can lead to inaccurate estimation of assembly states. A recent work uses CNN with two branches, one for object detection to identify a local area around the target, and another for 6-DoF pose estimation within that area [43]. This method is robust to noise and motion blur, whereas only 5 assembly states are evaluated and the performance on small part assembly is not guaranteed.

Chapter 3

System Design

This chapter describes target platform, developer tools, project requirements and system architecture. The application is developed following system design and software specifications. Additionally, testing plans are made based on system design.

3.1 Software and Hardware Specifications

This section specifies software and hardware for developing the AR application. The choices are justified based on background research and common knowledge.

3.1.1 Target Platform

The two dominant mobile platforms, Android and iOS, are both rich in support for AR-based applications, with advanced AR frameworks like ARKit[1] and ARCore[2]. Based on several studies [51, 52, 53], the comparison of AR applications in these platforms are summarized in Table 3.1.

Aspects	**Android** (ARCore)	**iOS** (ARKit)
Hardware Requirements	Lower	Higher
Quality of AR Experience	Good	Excellent
Developer Support	Less	More
Usability and UI/UX Design	Average	Excellent
Performance and Stability	Average	Excellent
Hardware Support	Good	Excellent
Ease of Development	Average	Easy
Supported Functionalities	< 10	10+

Table 3.1: Comparison of AR applications in Android and iOS

Regarding usability, performance and functionality in AR assembly guidance tools, iOS is chosen as the target platform since it provides generally better supports for AR applications. Although the higher hardware requirements and relevantly closer systems make iOS applications less compatible and portable, it is not in the project scope to adapt it to a wide range of devices.

[1] https://developer.apple.com/documentation/arkit
[2] https://developers.google.com/ar

3.1.2 Development Tools and Frameworks

Unity and Xcode, as two developer tools, are both capable of developing iOS AR applications. Based on some background research [54, 55], their major differences are listed in Table 3.2. Although Xcode only runs in macOS, its native support for ARKit allows developers to easily integrate AR experiences into their applications, without the need for additional plugins or libraries. Besides, Xcode contains a wide range of tools and resources, including a code editor, debugging options, and asset management tools, all of which facilitate efficient and streamlined development. While Unity offers similar capabilities in terms of AR development, it is a more general platform optimized for game development, and its toolset may not be as tailored to iOS as Xcode's. Thus, Xcode is chosen as the main developer tool for this project.

Features	**Unity**	**Xcode**
Prog. language	C#, C++, JavaScript	Swift, Objective-C
AR framework	AR Foundation (ARKit plugin)	ARKit, RealityKit, SceneKit
Build scheme	Unity project → Xcode project → iOS app	Xcode project → iOS app
Integration	3D design tools and other plugins	Other iOS features and frameworks
Performance	Optimized for game development	Optimized for iOS application
Platform	Windows, macOS, and Linux	macOS

Table 3.2: Comparison of Unity and Xcode for building iOS AR applications

3.1.3 Device and Dependencies

Following previous sections, the minimum hardware and software requirements for possible tools/frameworks used in this project are listed in Table 3.3, where ARKit, RealityKit and SceneKit are used for AR scenes, UIKit is used for UI elements, CoreML, CreateML are used for integrating ML models, and XCTest is used for software testing.

Tool/Framework	**macOS**	**iOS**	**Processor**
Xcode	High Sierra 10.13.6	-	-
ARKit	-	iOS 13	A9
RealityKit	-	iOS 13	A12 Bionic
SceneKit	High Sierra 10.13.2	iOS 11	-
UIKit	-	iOS 9	-
CoreML	High Sierra 10.13.2	iOS 13	-
CreateML	Mojave 10.14.4	iOS 16	A12 Bionic
XCTest	High Sierra 10.13	iOS 11	-
Summary	Mojave 10.14.4	iOS 16	A12 Bionic

Table 3.3: Software and hardware minimum requirements

Thus, an Apple device with iOS 16.0 and A12 processor or later is needed as the AR device, and a Mac with macOS 10.14.4 and above is required for developer tools.

3.2 Requirements

This section lists main requirements of the project. The requirements gathering process was completed based on literature review and common practices on mobile development. Afterwards, requirements prioritization is conducted and the risk of failure is analyzed. See Appendix C for system specification based on requirements.

3.2.1 Functional Requirements

Based on Section 1.3, 3.1 and Chapter 2, the main functional requirements (FR) are listed to specify what the system should do.

1. The application should be supported on the iOS version 16.0 and later.
2. The application should be supported on apple devices with A12 processor and later.
3. The application should use camera to capture real scenes to be displayed on screen.
4. The application should provide model-based AR instructions on each assembly step.
5. The application should align AR instructions with corresponding physical objects.
6. The application should provide 4 rendering options for AR instructions.
7. The application should be able to present AR instructions with correct spatial relationship to objects and user hands in the environment.
8. The application should be able to present the next or previous instruction based on different user operations on application interface.
9. The application should be able to present the next instruction automatically when a complete assembly step is detected.

3.2.2 Non-functional Requirements

The following non-functional requirements (NFR) are stated considering usability (1, 2), user experience (3, 4) and performance (5, 6).

1. The user interface should be informative and concise, where any button, switch or text fields should be clear and self-explanatory, with a text label for each switch.
2. User should be able to interact with the application without difficulty, where each operation should be easily performed, with a maximum of 2 fingers.
3. The user interface should be intuitive and clean, with a maximum of 10 elements, in order to prevent distraction and maximize the AR view.
4. The virtual content (3D models) should be rendered realistically with correct shape, color and spatial relationship, with an average alignment error less than 1 mm.
5. The virtual content should finish loading in a short time (no more than 3 seconds).
6. The virtual content should be rendered stably and fast during the whole AR session (around 60 FPS).

3.2.3 Priority and Risk

During the process of software requirements engineering, prioritization is important to decide which feature should be implemented first. However, almost all requirements have the risk of failure in their implementation, based on either time, resources or other unanticipated factors. Thus, a priority chart of each specific task with their failure rate is produced in Table 3.4. The failure rate could be estimated by common practices and related work. Considering both metrics, the task with high priority (1 is the highest) and low risk is implemented first, following a top-down order in the table.

ID	Description	Priority	Risk of Failure
FR-1	Software Support	1	5%
FR-2	Hardware Support	1	5%
FR-3	Camera Scene Capture	2	10%
FR-4	AR Instructions	2	20%
FR-8	Step Control	2	25%
FR-5	Alignment	2	30%
FR-6	Rendering Options	3	15%
FR-7	Occlusion	3	40%
FR-9	Instruction Automation	3	50%

Table 3.4: Priorities and risks of functional requirements

3.3 Architecture

This section introduces design and structure of the AR application based on functional requirements. The user and physical environment are also included to demonstrate the nature of AR as an interactive session.

3.3.1 System Structure

Figure 3.1 illustrates the overall structure of the application and relevant environments. The AR device refers to an iOS device with rear camera and is capable to run this application (FR-1, FR-2). The physical brick refers to the blocks in a specific LEGO set. When the user starts the application, the camera view will be presented in the screen (FR-3), and the first assembly instruction (the virtual brick 1) will appear once the marker image is detected (FR-4). The image marker acts as AR anchor to align virtual bricks with physical blocks, making the model-based AR instruction appear in the exact position of the current construction block, with the same scale and orientation (FR-4, FR-5).

In the process of block construction, the AR camera keeps capturing images of marker, physical blocks and user hands to achieve object alignment and hand occlusion in real-time (FR-5, FR-7). When specific user gestures are detected on the screen (e.g., a single tap), the next or previous virtual brick as an assembly instruction will be presented based on customized rendering mode (FR-6, FR-8). Furthermore, the next AR instruction will be displayed automatically once a completed step is detected (FR-9).

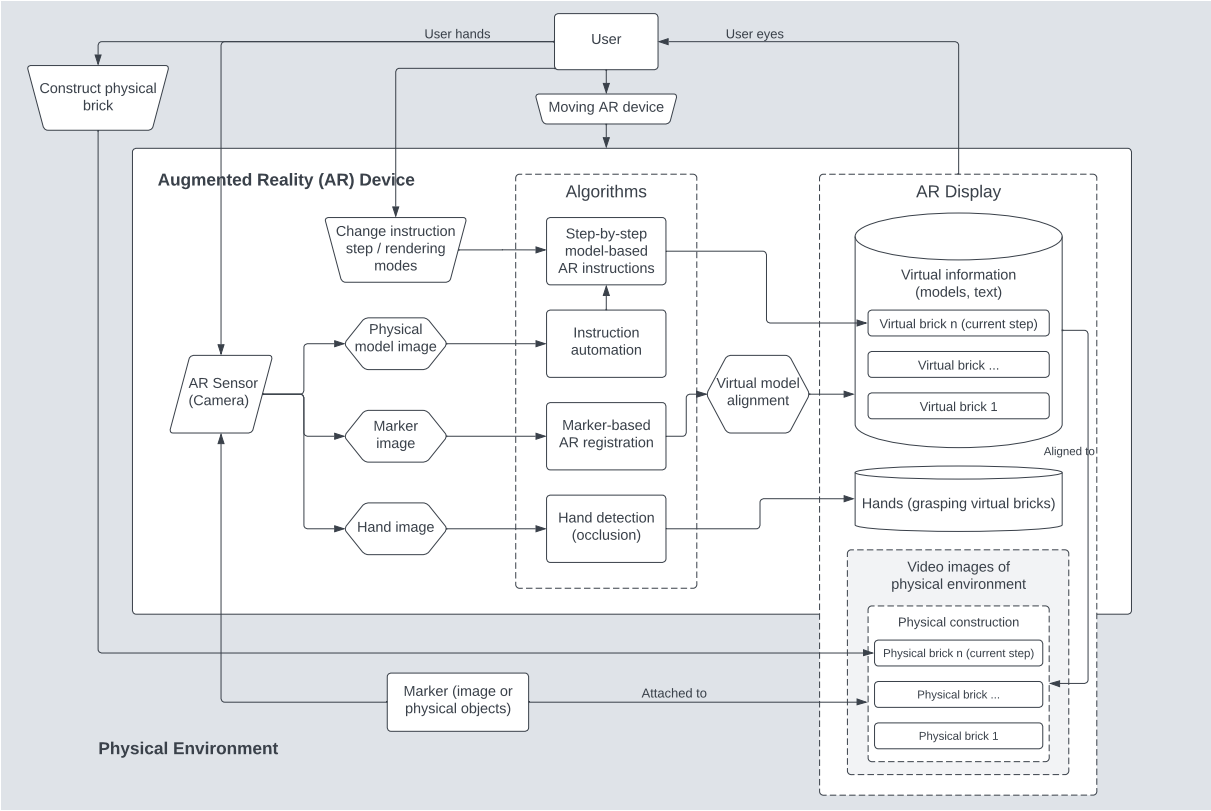


Figure 3.1: Architecture of the system

3.3.2 Class Structure

The AR application is designed following MVC pattern [56], where the `ViewController` handles user interaction on the screen (FR-6, FR-8), `NodeController` manages rendering sequence of digital models (FR-4), and `StepController` interacts with ML models for instruction automation (FR-9), as shown in Figure 3.2. See Appendix E for detailed documentation of each class and function, and project file structure.

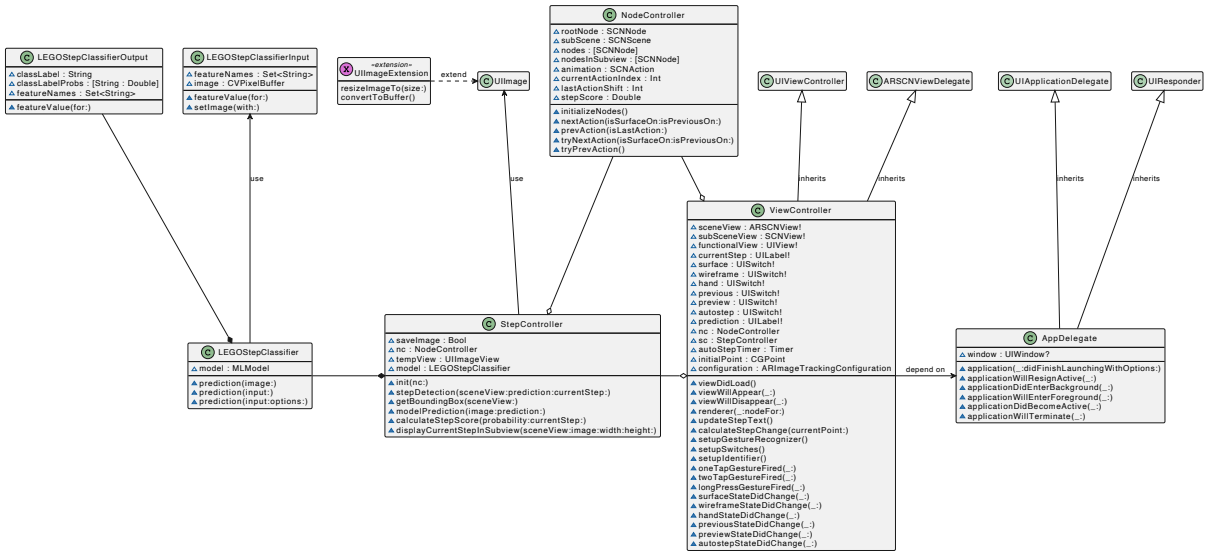


Figure 3.2: Class diagram of the application

Chapter 4

Implementation

This chapter specifies and justifies implementation details of the AR application and relevant hardware, based on software requirements (Section 3.2) and system architecture (Section 3.3). The implementation will be introduced following two main modules "AR Display" and "Algorithms" as illustrated in Figure 3.1.

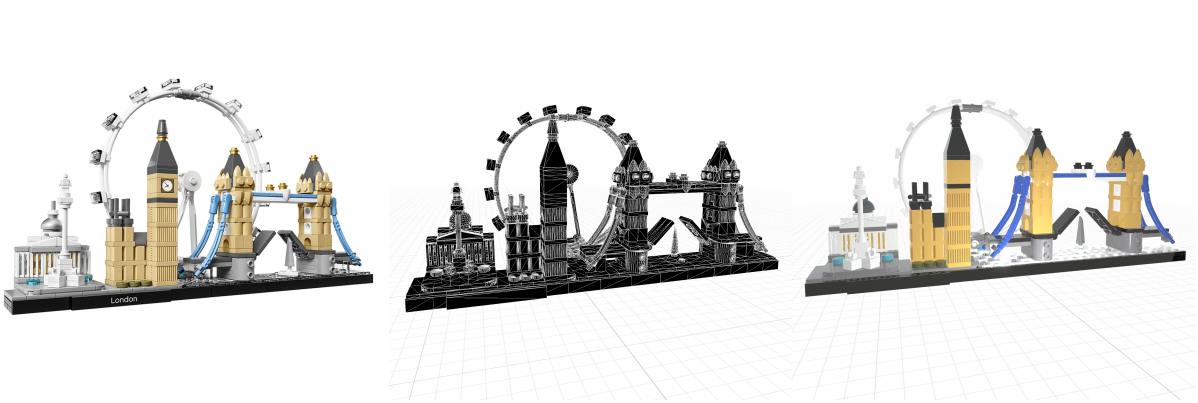
4.1 AR Display

The "AR Display" module deals with the content that are presented on the screen of the AR device. This section introduces the specific LEGO set chosen for the project, the user interface (FR-3), and a set of rendering options (FR-6) for the AR instructions.

4.1.1 Physical and Digital Model

To simulate manual assembly, the LEGO set 21034 (Architecture - London) is used for the task, as shown in Figure 4.1 (a). LEGO 21034 is constructed by 468 pieces with 98 unique shapes and 13 different colors (see Appendix D). The construction complexity is slightly higher than LEGO 21036 which is used for a similar AR guidance tool [17].

As discussed in Section 2.1.1, for greater reduction in expected construction time and placement error, model-based AR instruction is chosen to be implemented for this application. Thus, the high-fidelity digital model of LEGO 21034 prepared using CAD tools is downloaded from the internet [57]. Afterwards, lighting and reflection effects are added to make it more realistic, as shown in Figure 4.1 (c). Every piece of the virtual model



(a) Physical model (b) Digital wireframes (c) Digital model

Figure 4.1: Physical and digital model of LEGO 21034

can be rendered individually, enabling step-by-step instructions.

4.1.2 User Interface

The main user interface is designed to be informative and concise as shown in Figure 4.2, where any labels and switches are self-explanatory (NFR-1). The text bar on the top of screen indicates assembly status and progress, once the image marker is detected, it will present the number of current assembly step and the number of total steps. The `ARSCNView` is where the camera-captured scene and AR contents being displayed, which covers full screen to maximise the view on the assembly target (NFR-3).

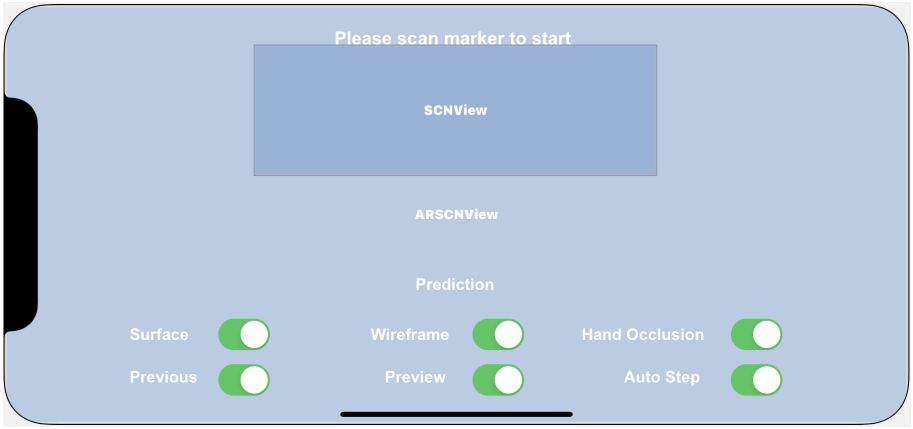


Figure 4.2: Main UI prototype created using Xcode

The functional pane on the bottom of screen contains 6 switches that provide options for instruction rendering (see 4.1.3), occlusion (see 4.2.3) and automation (see 4.2.4). The label `Prediction` will indicate whether the current step is classified as finished or not, once the switch `Auto Step` is turned on. The window `SCNView` will display each assembly piece individually, and will be hidden once the switch `Preview` is turned off.

For presentation consistency, the screenshots used in following sections are from the same model (iPhone 11) as shown in Figure 4.2. However, the application is not restricted to landscape orientation or a specific device. Thus, the concept of responsive design [58] is adopted to render screen elements well on a variety of devices and window sizes. See Appendix B for UI prototype on other devices following responsive design.

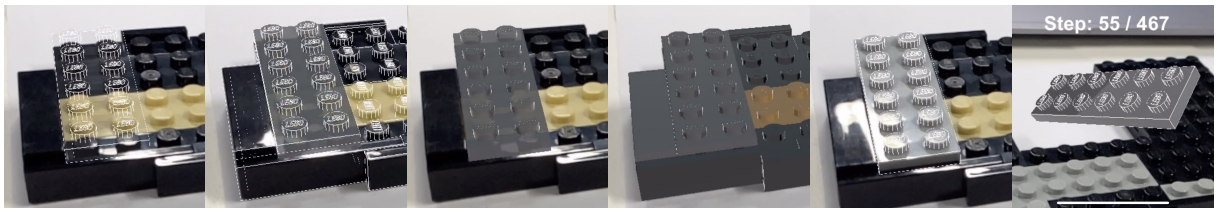
4.1.3 Rendering Options

One issue in model-based AR instructions is that sometimes the in-situ virtual model is small and similar to the background, making users difficult to identify its shape and position [18]. Thus, following rendering options are implemented to tackle relevant problems and enhance user experience, based on corresponding switch state as shown in Figure 4.2.

Surface: render the surface of each virtual block with colored, physical-based texture. This provides realistic AR instructions. Otherwise, the virtual block will be transparent.

Wireframe: display the wireframe of each virtual block. This improves visibility of virtual blocks in case that its color is similar to the background.

Previous: display all virtual blocks in previous construction. This shows correct spatial



(a) wireframe (b) previous+(a) (c) surface (d) previous+(c) (e) **real block**+(a) (f) preview

Figure 4.3: Different rendering options for AR instructions

relationship between the current construction block and previous ones, in case that the virtual model does not aligned properly with the physical model.

Preview: display the current virtual block with rotating animation separately in preview window below the progress bar. This demonstrates the shape of current block from different perspectives, in case that the in-place virtual model does not present clear geometry.

The 4 rendering options can result in 16 possible option combinations, partly illustrated in Figure 4.3, which demonstrates high instruction visibility.

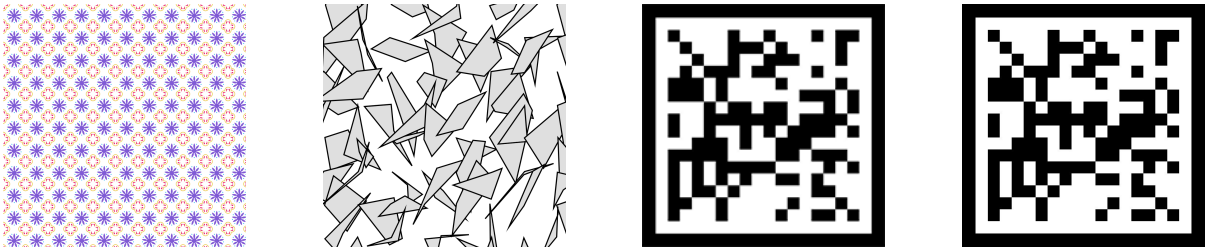
4.2 Algorithms

The "Algorithms" module contains methodologies for AR registration (FR-5), step control (FR-4, FR-8), occlusion handling (FR-7) and instruction automation (FR-9). This section introduces these methods and their implementations.

4.2.1 Marker-based Registration

To align virtual objects with corresponding physical parts, image marker is set up as the AR anchor, which is a preferred registration method for AR-guided tools, as discussed in Section 2.1.2. Markers should have specific patterns that cameras can easily recognize, and are visually distinct from their surroundings. According to a previous study [59], the reference image should be chosen following several guidelines to optimize performance:

1. Image should have a lot of detail and high contrast.
2. Repetitive patterns and low resolution images should be avoided.
3. Image should be attached to flat surfaces, and itself should not be glossy.



(a) score = 0 (b) score = 85 (c) score = 75 (d) score = 100

Figure 4.4: Different image markers and their quality score

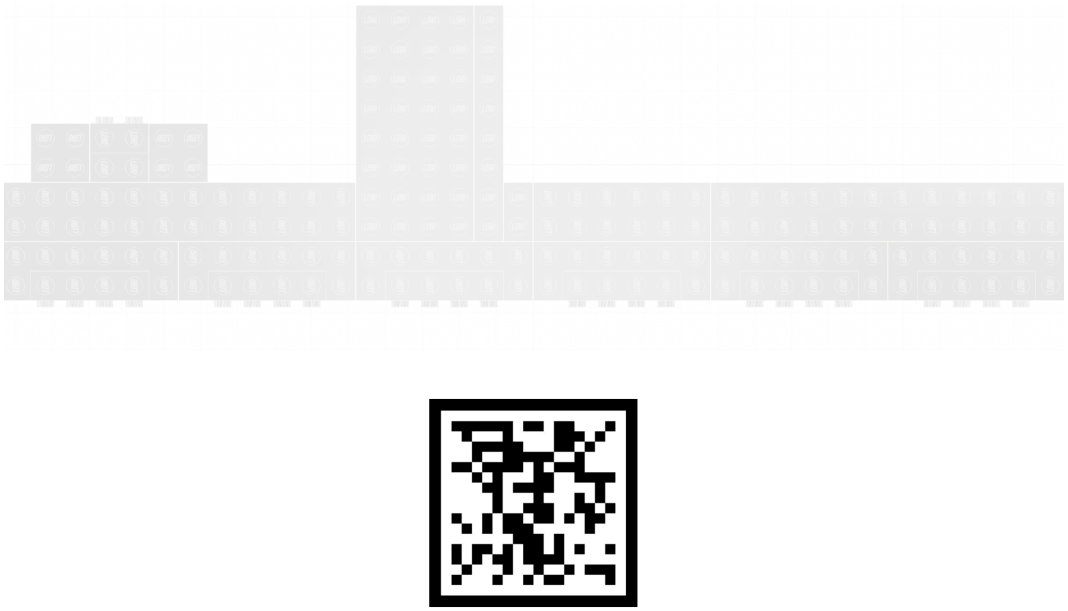
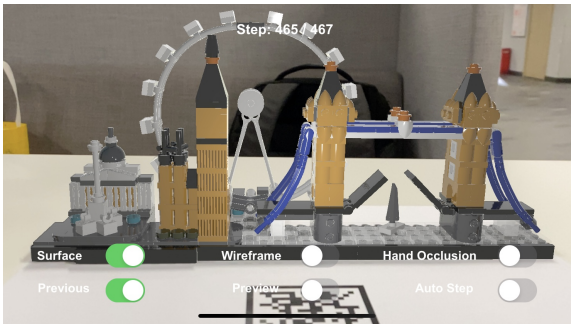


Figure 4.5: Assembly base in the size of A4 paper

Using Pattern Monster [60], Brosvision [61] and ARMaker [62], a list of image markers are generated as candidates. Their quality as a reference image is measured by `arcoreimg` in ARCore SDK [63], represented by a numeric score ranging from 0 to 100. As shown in Figure 4.4, repetitive patterns in marker (a) result in zero score, which is unsuitable for tracking. Marker (b) achieved 85% due to randomness and abundance in its quadrangles. Marker (d) further increases detail and contrast, reaching full score. However, the low-resolution (200×200) version of marker (d) (marker-c) only obtains 75% of marks.

Regarding above, marker (d) is used as the reference image, integrated with the floor layout of the original model, as shown in Figure 4.5. These two images could be printed together on an A4 paper, thus the layout of floor level could also be used to locate physical blocks. This solution is more flexible and convenient than LEGO-constructed image marker (Section 2.1.2), since the paper-printed assembly base is easy to obtain and reproduce. However, manual re-alignment might happen during construction process, as LEGO pieces cannot be fixed on the paper base and could be easily slipped. Figure 4.6 demonstrates that AR registration achieves precise model alignment even part of the marker is occluded (a), and places realistic virtual model in correct position and pose (b).

(a) Alignment: virtual wireframe and LEGO model



(b) Virtual model near complete steps

Figure 4.6: Demonstration of marker-based registration (NFR-4)

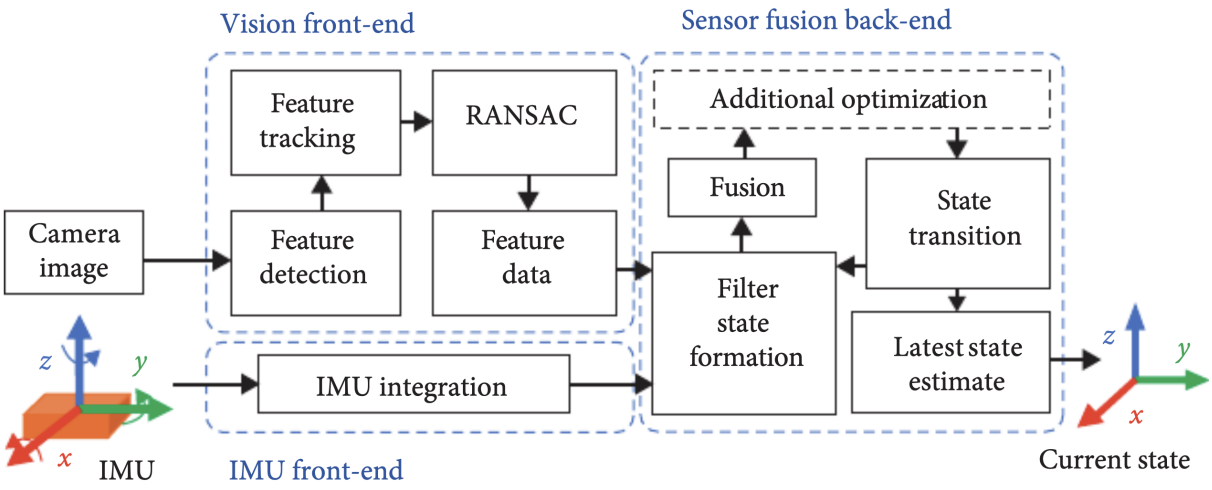


Figure 4.7: Architecture of Visual Inertial Odometry [64]

To create a correspondence between image marker and virtual contents, Apple ARKit uses Visual Inertial Odometry (VIO) that combines data from device's motion sensor and computer vision analysis of camera captured scenes [51], as illustrated in Figure 4.7. ARKit recognizes notable features in the marker image, tracks position differences of these features between video frames, and compares them with motion sensing data [65]. The vision-sensor hybrid approach aims to address the stability issue in vision-based methods as discussed in Section 2.1.2.

4.2.2 Step Control

In 3D model-based AR instructions, the virtual brick of each assembly step needs to be rendered individually in the right place, which is feasible as the whole digital model is constructed from single pieces, as discussed in Section 4.1.1. Therefore, virtual blocks are stored in an indexed array, following construction order in the assembly guide [66].

Considering ease of interaction (NFR-2), a set of gesture handlers are implemented to recognize and respond to user operations on the device screen:

Tap by one finger: present the next AR instruction based on rendering options (Section 4.1.3), if current instruction is not the last one.

Tap by two fingers: hide current AR instruction and present the last instruction based on rendering options (Section 4.1.3), if current instruction is not the first one.

Press and drag: press and drag gesture is a widely adopted method for precise and flexible selection on touch-screen devices [67]. It is implemented in this application to support instruction fast forward/backward, allowing users to jump to the desired AR instruction in a smooth way. When a one-second press on screen is detected, the instruction change is triggered per pixel move of the touch point until release, and the index of instruction (IoI) to be presented is calculated as:

$$I_{current} = I_{press} + \frac{x_{current} - x_{press}}{x_{max}} I_{max}$$

where $I_{current}$ is the current IoI, I_{press} is the IoI before dragging, I_{max} is the maximum IoI, $x_{current}$ is the x-coordinate of current touch point (during dragging), x_{press} is the x-coordinate of touch point before dragging, and x_{max} is the maximum horizontal dragging distance (the window width). This method allows instruction fast forward/backward by dragging right/left respectively, following human intuition and logic in media players. The drag gesture is only valid after one-second press, which prevents unintentional swiping that causes instruction change. The drag distance per step change depends on the screen width and number of instructions, thus users can browse all the steps in a single swipe across the screen in various sizes. It is also easy to fix excessive movement by dragging to the reverse direction before finger release.

4.2.3 Occlusion Handling

Since it is challenging to obtain depth variation in small part assembly (as discussed in Section 1.2), a simple yet effective object occlusion method is proposed, utilizing features of in-situ model-based instructions. As mentioned in Section 4.1.3 and 4.2.1, when rendering instructions using "Surface + Previous" mode as shown in Figure 4.3 (d), the current virtual block will be rendered in correct spatial relationship with other virtual blocks, which represents previously assembled physical blocks. Thus, the current virtual block as the AR instruction could present correct spatial relationship with assembled physical blocks, if previous virtual blocks occlude current virtual block, but are transparent to physical blocks, as illustrated in Figure 4.8.

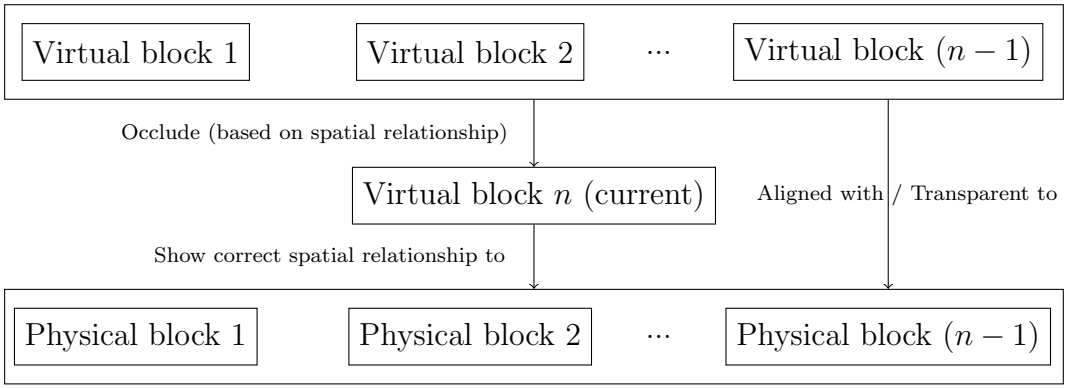
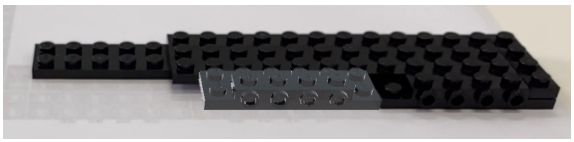


Figure 4.8: Principle of content-based object occlusion

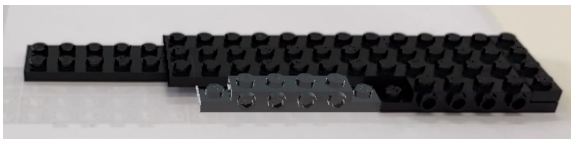
Considering the fact that each virtual block is aligned properly with corresponding physical block and equivalent in size and shape (Section 4.2.1), they could act as physical blocks to mask on the current virtual block (the AR instruction) with high accuracy. Since no calculation in depth map is required, the generation of occlusion mask could be fast to meet real-time requirements of AR applications (NFR-6).

Figure 4.9 (a) and (b) demonstrates the result of content-based object occlusion, where the virtual instruction block is half-occluded by physical blocks in the top layer, presenting proper spatial relationship and removing ambiguity on depth perception. In the application, object occlusion is disabled by default, and can be turned on by switching `Surface`, `Wireframe` off and `Previous` on in the UI as shown in Figure 4.2.

However, the limitation of this method is the lack of scalability on some assembly tasks, where the object to handle its occlusion does not have corresponding digital model to

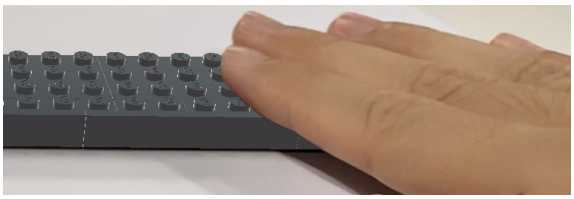


(a) Object occlusion off



(b) Object occlusion on

(c) Hand occlusion off



(d) Hand occlusion on

Figure 4.9: Demonstration of object and hand occlusion

apply masks on the AR instruction.

With regard to hand occlusion, the DeepLabv3+ [68] model is employed for hand segmentation to generate accurate occlusion mask efficiently, as discussed in Section 2.2.2. DeepLabv3+ utilizes ResNet-101 [69] architecture, atrous convolution [40] and a decoder module [70] to produce efficient and detailed segmentation. ARKit has fine-tuned and optimized the model pretrained on COCO dataset [71] for mobile devices, making it suitable for hand segmentation in AR applications [72]. Afterwards, the depth map created using the device's camera and sensors determines whether the AR content should be occluded by people [72]. Figure 4.9 (c) and (d) demonstrates high precision hand occlusion on virtual models. However, without the back-facing dual camera or LiDAR sensor exclusive to newer apple devices, the depth data cannot be obtained so that the hands will always occlude virtual content regardless of distance [73].

4.2.4 Instruction Automation

The proposed instruction automation method is based on state classification in local areas (SCLA), inspired by a related work in state estimation [43], as discussed in Section 2.3.2. The principle is to superimpose digital wireframes on the image of physical models, and determine whether the step is completed based on coincidence degree between the wireframe and physical model by a CNN classifier. A semi-automatic image capture method is applied to gather synthetic training data efficiently. The system aims to provide robust and stable instruction automation using less training data, to achieve good generalization on the small part assembly scenario with a large number of pieces.

Motivation

Although piece recognition and state estimation are widely used for AR instruction automation (Section 2.3), there are several challenges unique to this task:

Lack of features. Most of LEGO blocks to identify are small in size with single color and repetitive patterns (see Appendix D), which may result in poor performance in piece recognition methods. In addition, similar patterns and colors between the target object and nearby LEGO blocks could make the recognition result even less reliable. Thus, the

proposed method extracts a local area around the target before making prediction.

Number of states. The LEGO set used in this project have 468 pieces (Section 4.1.1), resulting in 468 assembly states. Thus, to automate AR instruction by 468-label state estimation might not be practical as collecting training data for each assembly state is time-consuming. Furthermore, the estimation result may be inaccurate since the difference in two consecutive states is not distinguishable, due to small size in a single block. However, it is observed that in a complete step, the digital instruction wireframe will coincide with its physical block, while an incomplete step does not. Therefore, the proposed method only focuses on two states of each assembly step - Finished and Unfinished, regarding their difference when overlaying digital wireframes.

Duplicated pieces. Over half of the pieces in this LEGO set are duplicates (Appendix D), some pieces even appear in 40 consecutive steps [66]. Thus, piece recognition is unreliable since it may be confused on duplicates and make false predictions. However, focusing on two states of each assembly step does not suffer from this issue.

Considering these challenges, it makes sense to use a 2-state classifier for analyzing video frames. Since it focuses on the features of single frame (the digital-physical model coincidence degree) rather than spatio-temporal relationships between adjacent frames, the system adopts image classification model instead of video classification.

Method

As shown in Figure 4.10, the instruction automation system (IAS) consists of two modules, one for image cropping and another for step scoring. The image cropping module runs every three video frames during the AR session, and the output image is fed into the CNN to generate a 2-dim array of 2-label classification results (Finished/Unfinished). The step scoring function continuously takes the CNN output over time, producing the completeness score of current assembly step. The next AR instruction will be presented based on whether the completeness score meets a threshold value.

AR overlay. When instruction automation is enabled, the rendering options is set to "Wireframe" only (see Section 4.1.3), to present edges of the physical block to determine its coincidence degree with the instruction wireframes. Hiding wireframes of previous physical blocks prevents their impact on prediction.

Image cropping. Previous works have shown that applying CNN to local area around

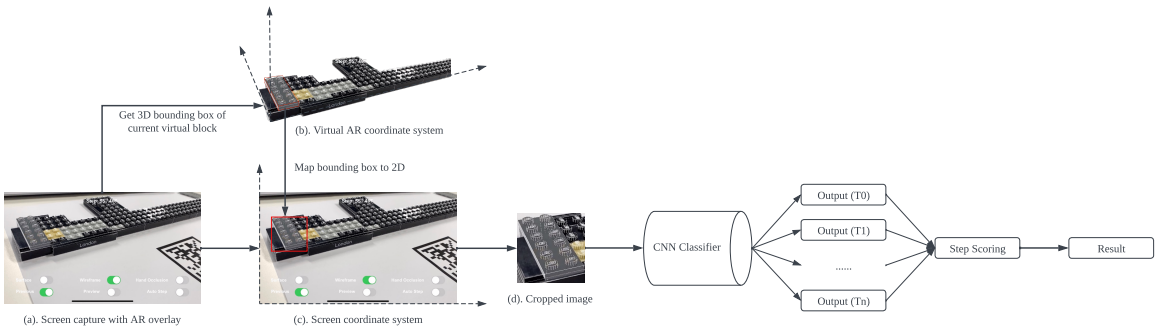


Figure 4.10: Architecture of instruction automation system

the classification target rather than the whole image reduces the impact of variations in the background and provides more reliable results (Section 2.3.2). Due to the alignment between virtual and physical blocks (Section 4.2.1), there is a more efficient way to obtain bounding boxes around the assembly target. As shown in Figure 4.10 (b), the local 3D coordinate system of the AR content (SceneKit node) records the position and size of the current AR instruction block and its 3D bounding box. The minimum and maximum corner points of the 3D bounding box is then projected to the 2D pixel coordinate system of the renderer to get the 2D bounding box on the screen (Figure 4.10 (c)). The screenshot will be cropped based on the resulting bounding box (Figure 4.10 (d)). Since it is assumed that the only change in each assembly step is a single LEGO block, and the physical block should coincides with its digital wireframe in complete steps (Section 4.2.1), the cropped image containing the block's digital wireframe provides sufficient information to determine whether the assembly step is completed.

CNN classifier. MobileNetV2 [74] is employed for state classification, due to its considerable performance and computational efficiency on mobile devices. The model takes 299×299 images with pre-trained weights on ImageNet [75] and is fine-tuned on the data as described in Section "Dataset".

Step scoring. Considering the probability that the CNN could make incorrect predictions, it is not reliable to determine whether an assembly step is completed based on single video frame. Thus the step scoring function continuously takes CNN outputs and updates completeness score over time. The completeness score S of i^{th} assembly step in j^{th} time period is calculated as:

$$S_i^{(j)} = \sum_{t=0}^{j} (\alpha(P_f)_i^{(t)} - \beta(P_u)_i^{(t)}) \tag{4.1}$$

where P_f, P_u are the probability distribution over predicted output classes "Finished" and "Unfinished" respectively; α, β are coefficients that control the dependence of S on P_f and P_u. At the beginning of each assembly step, the score and the time period number are set to 0. Each time period starts when the CNN makes a prediction and lasts until the next prediction. The accumulated scoring function tolerates a few incorrect predictions and provides a generally reliable estimation based on multiple frame samples.

Dataset

The dataset contains 1,198 images of two assembly states (599 Finished, 599 Unfinished) cropped from screen captures using the same method as shown in Figure 4.10, focusing on the features of target object and removing unrelated backgrounds. By moving the AR device during image capture at frame level, the synthetic data is obtained from different perspectives efficiently. See Appendix E for dataset location.

Since the LEGO set has 468 pieces, it is difficult to collect images of two states for each assembly step. Regarding common features among several steps and duplicated pieces, the images of 10 most distinctive steps are collected, partly illustrated in Figure 4.11, where the first row demonstrates unfinished states of each step, and the second row shows their finished states. For each image in Figure 4.11, a number of variants are collected from different perspectives. In general, the dataset has following properties:

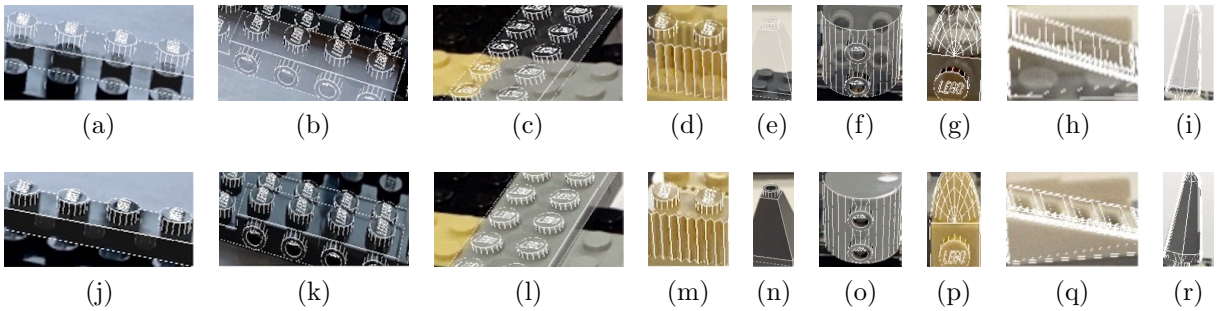


Figure 4.11: Sample items in the dataset

Geometry variance. The dataset contains normal plates (j, l), special plates (k), bricks (m), slopes (n), arches (o), and other decoration elements (p, q, r), demonstrating a high level of geometry variance, which helps the classification model better understand object-wireframe relationships.

Color variance. The objects in dataset contains 5 unique colors, including black (j, k), yellow (m, p), dark grey (n, r), medium grey (l, o) and white (q), covering around 80% of pieces (see Appendix D), which helps improve generalization of the classification model.

Universality. Each object in the dataset is representative and similar to several pieces (see Appendix D), which is efficient to train a classification model that generalize well on the LEGO set in this task.

Performance. Each image in the dataset is manually inspected to ensure accurate alignment between digital wireframe and physical model, which helps the classification model better distinguish "finished" state from "unfinished" state - where the main difference is the coincidence degree between digital wireframe and physical model.

Training

Training is performed using TensorFlow[1] framework on Colab[2] platform. The classification model (see 4.2.4) is downloaded from Keras[3] with pre-trained weights on ImageNet [75], fine-tuned on our dataset, and finally converted to CoreML (`.mlmodel`) format for use in the AR application. See Appendix E for training code location.

Data augmentation is applied to the training set for better generalization, considering possible data variations, including a rotation and shear range of $-20°$ to $20°$, a zoom range of 0.8 to 1.2, a width and height shift up to 20% and a random horizontal flip.

Training parameters including validation split of 20%, batch size of 128, and Adam optimizer with learning rate of 0.001. The accuracy and cross entropy loss during 50 training epochs are visualized in Figure 5.5 in Section 5.2.2.

Discussion. The accuracy measured on validation set may not guarantee a similar performance on classifying states of other assembly steps that are not collected in the dataset, the user experience is also affected by step scoring function, see 5.2.2 for evaluation.

[1] https://www.tensorflow.org/overview
[2] https://colab.research.google.com/
[3] https://keras.io/api/applications/mobilenet/

Chapter 5

Evaluation

This chapter specifies testing plans during software development and presents evaluation results from system testing, which provides comprehensive evaluation on the software.

5.1 Development Testing

Development testing is conducted consistently throughout the software development life cycle, to detect bugs or defects at an early stage, and reduce additional time and cost for debugging. This section introduces several testing plans to evaluate whether the software conforms to its (detailed) system design (Section 3.3), as shown in Figure 5.1.

The XCTest [77] framework is used to implement and perform all the development tests, including unit testing, integration testing and UI testing.

5.1.1 Unit Testing

Unit testing is conducted to validate the smallest pieces of code that can be logically isolated in the software. They are categorized by three main classes, namely `ViewController`, `NodeController` and `StepController`, as shown in Figure 3.2.

The unit tests have validated several functions for each controller class. However, not all functions are covered because some requires specific inputs or dependencies that cannot be mocked easily to ensure they are tested independent on other parts of the software.

The unit testing report is illustrated in Table 5.1, where code coverage on `NodeController` reaches 100%, ensuring proper presentation in AR instruction flow. However, since

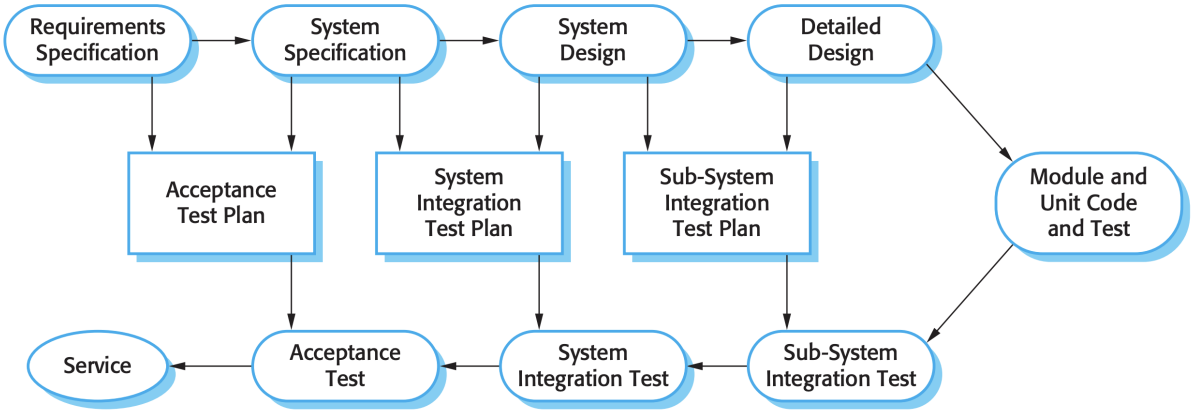


Figure 5.1: Testing phase in plan driven approach [76]

Class under test	Function tested	Testing objectives
`ViewController`	`viewDidLoad()`	Ensure screen elements could be initialized properly
	`updateStepText()`	Ensure progress text showing correct content
	`setupSwitches()`	Ensure functional switches have correct initial states
`NodeController`	`initializeNodes()`	Ensure the number of nodes (AR models) stored in the list is correct
	`nextAction()`	Ensure the next AR instruction could be presented correctly
	`prevAction()`	Ensure the previous AR instruction could be presented correctly
	`tryNextAction()`	Ensure boundary cases are handled (no next instruction)
	`tryPrevAction()`	Ensure boundary cases are handled (no previous instruction)
`StepController`	`calculateStepScore()`	Ensure step score is properly calculated given CNN predictions

Table 5.1: Unit testing targets and objectives

`ViewController` contains functions dependent on other classes, and some functions in `StepController` take certain image as input, they cannot be fully covered by unit testing. Despite this, unit testing has validated some of their basic functions, such as screen element initialization, text updating and step scoring given mocked CNN outputs.

5.1.2 Integration Testing

Integration testing is implemented to validate sub-systems which combines several functions and corresponds to modules in the system design, as shown in Figure 3.1. There are two sub-systems being tested:

Instruction rendering system handles rendering modes of AR contents (Section 4.1.3). It is tested to ensure proper rendering mode could be enabled or disabled by state changes in corresponding switches, including "Surface", "Wireframe", "Previous" and "Preview".

Instruction flow system handles user action on the application interface to present desired AR instructions (Section 4.2.2). It is tested to ensure that proper instruction could be presented under certain user action, including single finger tap, two fingers tap and press & drag.

The verification of two sub-systems has proved that the application is capable of rendering AR contents in proper sequence and various configurations under different user actions. However, some sub-systems are not tested individually due to their dependence on other sub-systems, which are tested together in full system testing (Section 5.2). The sub-systems not covered by integration testing include:

AR registration system detects image markers and consistently aligns AR contents to corresponding physical models (Section 4.2.1). It is not tested because it is hard to determine the quality of alignment if AR content could not be presented properly.

Occlusion handling system handles object occlusion and hand occlusion (Section 4.2.3). It is not tested since object occlusion relies on accurate registration and instruction flow.

Instruction automation system automates presentation of AR instruction based on completeness of each assembly step (Section 4.2.4). It is not tested because it requires AR registration to achieve accurate image cropping.

5.1.3 UI Testing

UI testing aims to check whether screen elements are visible and accessible following presentation logic, and whether the application responds properly to user interactions. It ensures user interface (Section 4.1.2) is consistently maintained in terms of appearance and functionality, which cannot be validated by unit testing or integration testing. UI tests are implemented for:

Element visibility, including text labels, preview window and functional switches, as shown in Figure 4.2. The element visibility is checked for two UI states: before and after recognizing image marker, where functional switches and their labels are only visible after image marker is detected.

Element interactivity. Some elements including AR scene view and functional switches, should be able to interact with to trigger corresponding actions. Their interactivity are checked to ensure the interaction interface is provided.

Gesture handling, including single finger tap and two fingers tap, as discussed in Section 4.2.2. Although this is similar to integration testing for instruction flow system, UI testing simulates real gestures on the interface, while integration testing directly tests functions called after recognizing gestures. Thus, UI testing ensures user gestures could be recognized by the application and triggers corresponding actions.

However, in UI testing, XCTest framework can only simulate user interaction, and has access to screen elements rather than certain class or function. Additionally, some gestures and screen elements are only valid or visible after image marker is detected. Therefore, UI testing is not fully automated because the image marker needs to be presented near device's camera when conducting UI testing.

5.2 System Testing

System testing evaluates interactions between various components that make up the system, and validates system design (Section 3.3) and specification (Appendix C) before release, as shown in Figure 5.1. This section presents system testing result following performance driven strategies, where (loading / running) performance and (registration / occlusion / automation) accuracy are evaluated. In addition, all system specifications are validated with reference to certain sections, as listed in Appendix C.

5.2.1 Performance Analysis

The performance is evaluated regarding application loading time and running frame rate. Since the software contains large digital LEGO model (39.7 MB), the loading and rendering speed is critical for user experience. The iPad Air (3rd generation) with A12 processor and 3GB RAM is used for performance testing as it meets the minimum hardware requirements for the application (Section 3.1.3). Theoretically, the testing result demonstrates the minimum expected performance on all supported devices.

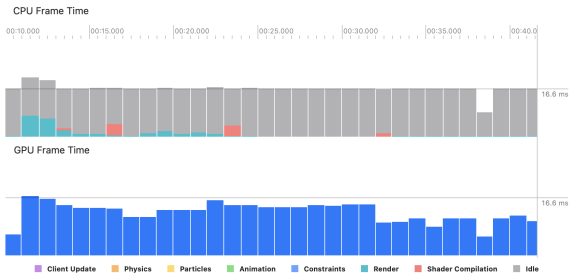
Table 5.2 shows loading time of several events during application lifecycle, analyzed by Instruments [78] software. The results demonstrates fast loading speed, in terms of initialization (508.77 ms) and launching. As discussed in Section 5.1.3, the system finishes camera and UI initialization in state 1, where users are able to scan for the image marker.

It enters state 2 when image marker is detected, digital LEGO model is loaded and the first piece of model (AR instruction) is rendered. Thus, the application takes 531.51 ms to enter state 1 where the application becomes active, and additional 232.31 ms for loading the whole model and ready for rendering, which is a highly acceptable speed and more than 4 times faster than NFR-5.

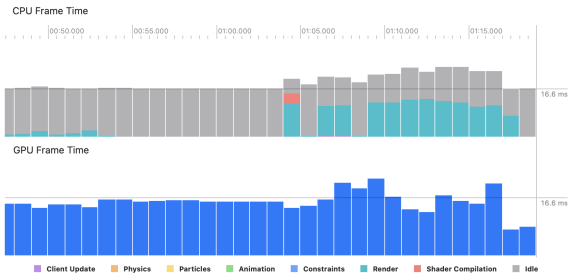
Event	**Description**	**Duration** (ms)
Initialization	Process creating	124.32
	System framework	249.70
	Static runtime	134.75
Launching	UIKit initialization	22.51
	UIKit scene creation (state 1)	0.23
	UIKit scene creation (state 2)	226.50
	Initial frame rendering	5.81

Table 5.2: Events loading time during application lifecycle

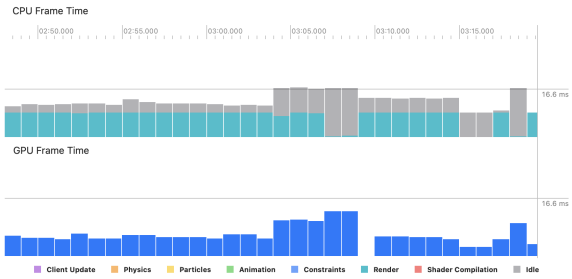
Figure 5.2 visualizes CPU and GPU frame time under different configurations (see 4.1.3, 4.2.3 and 4.2.4), which determines frame rate, a critical performance metric in AR applications. All the measurements are taken in last few guidance steps, where most of the digital model could be rendered, so that the overall frame rate is expected to be the lowest throughout the process. When rendering all virtual blocks with realistic texture, wireframe and separate preview (Figure 5.2 (a)), the GPU frame time fluctuates slightly around 12 ms and the CPU frame time (apart from idle) is negligible. In this graphics-intensive scenario, the rendering speed depends on the GPU and keeps above 60 FPS. Additionally, configuration (a) enables all rendering modes, where most of the elements



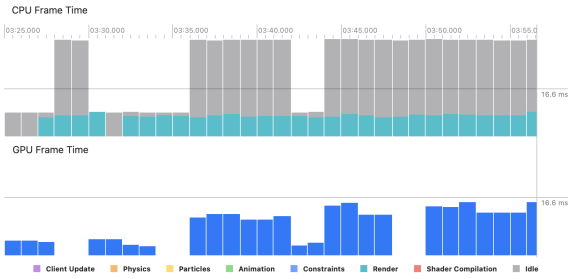
(a) Surface + Wireframe + Previous + Preview



(b) Hand occlusion + (a)



(c) Instruction automation



(d) Hand occlusion + (c)

Figure 5.2: Average CPU and GPU frame time per second in different configurations

are rendered, so the average performance in other rendering configurations (Section 4.1.3) should be even better. Thus, other rendering configurations are not evaluated, such as one for object occlusion (Section 4.2.3), where only "Previous" mode is enabled.

Considering instruction automation (Section 4.2.4), as shown in Figure 5.2 (c), the CPU frame time is significantly higher than (a) since the system continuously calculates bounding boxes and feeds cropped images into CNN model. However, the GPU load is much lighter since only "Wireframe" rendering mode is enabled, which is restrictive when instruction automation is enabled. Thus in this scenario, CPU is the bottleneck rather than GPU, and the average frame rate is much higher than (a).

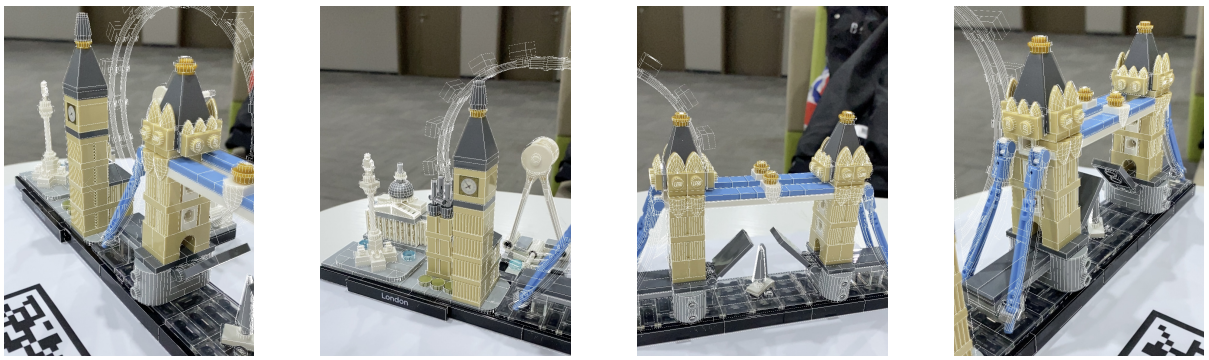
When hand occlusion is enabled on the basis of (a), as shown in Figure 5.2 (b), a slight increase in GPU frame time is observed, resulting a drop in the overall frame rate to around 60 FPS. In the second half of the graph, the CPU frame time sees a significant growth along with fluctuation in GPU frame time, which might result from intensive computation in hand segmentation. Similarly, enabling hand occlusion in addition to (c) increases the burden on GPU and reduces frame rate, as shown in Figure 5.2 (d).

In general, the application demonstrates acceptable frame rate (over 60 FPS) in each of the four computational intensive scenarios, and the frame rate fluctuation is imperceptible on mainstream apple devices with a 60 Hz display.

5.2.2 Accuracy Evaluation

While performance (Section 5.2.1) determines the smoothness of user experience, accuracy is crucial for minimizing confusion brought by the AR content [17]. The accuracy is measured considering image registration, hand occlusion and instruction automation. As illustrated in Section 4.2.3, the accuracy of object occlusion depends on registration, thus it is not discussed in this section.

Image registration accuracy is evaluated by measuring alignment error between virtual wireframes and physical models from different perspectives, as shown in Figure 5.3. Alignment error is defined as the distance between corresponding edges on the physical and virtual models, obtained by visually examining sample images. Four commonly used perspectives are examined, ranging from 20° to 60° with 10 cm camera height. In general, the wireframe aligns perfectly in most of the regions on physical model, despite some



(a) Left facing 60° (b) Left facing 30° (c) Right facing 20° (d) Right facing 45°

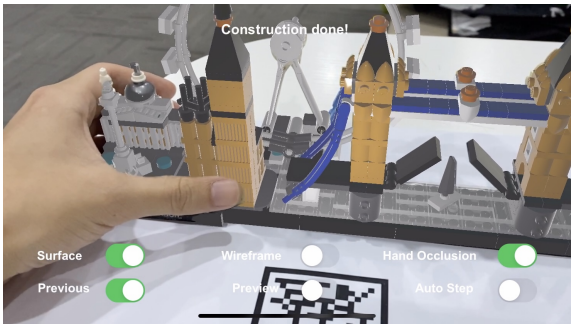
Figure 5.3: Evaluating registration performance from different perspectives

small errors in (a) top of the tower bridge, (b) center of the sky wheel, (c) sailboat under the bridge, and (d) building in the bottom left. There are also considerable misalignment in blue ropes of the tower bridge in each scenario, due to the discrepancy between digital wireframes and the physical model. Regarding true proportions, the average alignment error is found to be less than 1 mm in different perspectives.

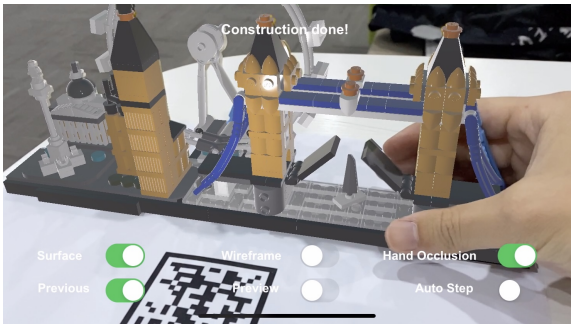
It is observed that the error is affected by following factors: (1) the fidelity of virtual model, such as misaligned blue ropes, (2) the attachment of LEGO blocks being looser or tighter with player's slight, free rotations, (3) the design of image marker and its claimed size, which determines the size of virtual model relative to the marker, and (4) the distance between camera and marker, where the error increases when the camera moves farther. In addition, the error significantly grows when certain areas of the marker is occluded or out of camera view. It is found that the registration works well with less than a quarter of marker being occluded, or half of it being out of view, as shown in Figure 4.6.

Hand occlusion accuracy is measured on sample images in Figure 5.4 with two common hand gestures during construction, using intersection over union ($IoU = \frac{IntersectionArea}{UnionArea}$), where the two areas are the number of pixels in the intersection or union between predicted occlusion mask and the expected mask. The predicted occlusion mask is rendered in blue (`#0000FF`) by `ARMatteGenerator` in ARKit, while the expected mask is obtained by manually selecting hand regions and visualized in red (`#FF0000`). To observe intersection areas, $\alpha = 0.5$ is set to both colors, resulting overlapping regions in purple, see sub-figures (c) to (f). The background is removed to calculate IoU by $\frac{C_{purple}}{C_{red}+C_{blue}+C_{purple}}$, where C_{red}, C_{blue} and C_{purple} are the pixel count in corresponding colors.

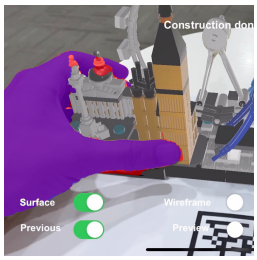
The IoU on sample images reaches 93.69% and 92.81% respectively, around 5% higher than a recent work in AR assembly tools [17]. It is observed that the occlusion error is centered on finger fount-end, especially in forefinger, though the algorithm works well



(a) Hover with left hand

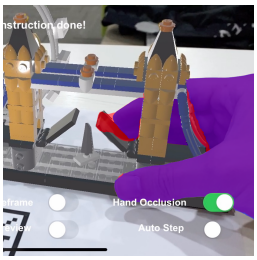


(b) Grasp with right hand



(c) Masks on (a)

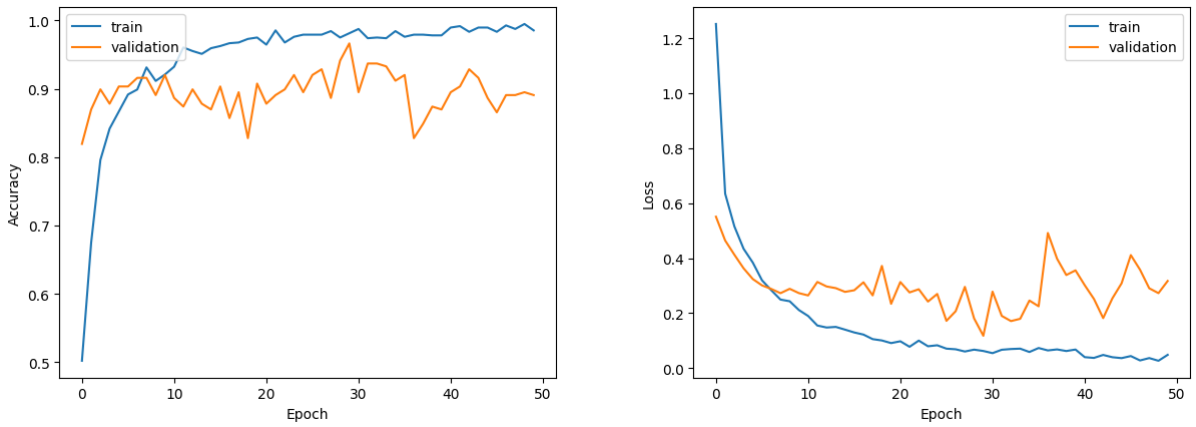
(d) IoU = 93.69%



(e) Masks on (b)

(f) IoU = 92.81%

Figure 5.4: Evaluating hand occlusion accuracy on digital model



(a) Classification accuracy in each epoch

(b) Cross entropy loss in each epoch

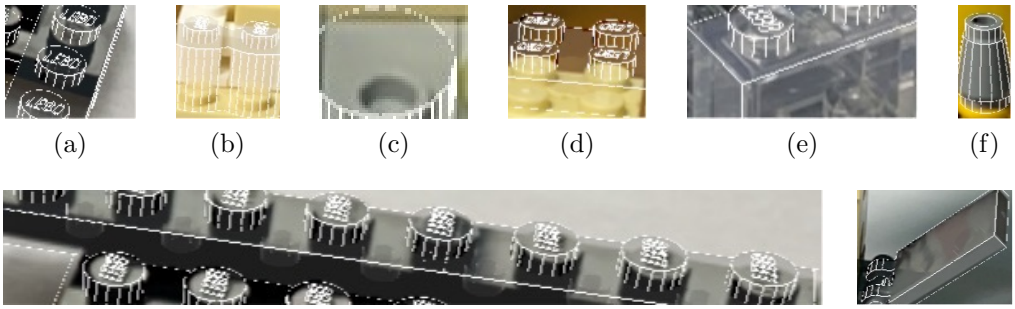
Figure 5.5: Visualization of metrics on training set and validation set

when part of the hand is occluded by physical model, in both horizontal (d) and vertical (f) directions, given that the physical model aligns properly with its digital model. The occlusion accuracy is observed to be affected by following factors: (1) lighting conditions, where accuracy drops in darker environments, (2) color similarity between hand and background, and (3) hand motion, where it performs better in still.

Instruction automation accuracy is evaluated from two aspects: (1) the classification accuracy on two assembly states. (2) the instruction triggering time (ITT) after step completion. Although (2) heavily depends on (1), the parameter setting in step scoring function (Section 4.2.4) trades off speed and reliability of instruction presentation.

The classification accuracy on train and validation set (see Figure 4.11) during 50 epochs is illustrated in Figure 5.5, where the validation accuracy peaks at 96.7% in epoch 29 and drops due to over-fitting. The accuracy is further evaluated on other images of assembly pieces that are not collected in the dataset, as shown in Figure 5.6, where (a) to (d) demonstrate "Unfinished" state, and the rest are in "Finished" state. For each item, a number of image variants from different perspectives are collected and tested to provide a reliable estimation of classification accuracy on each assembly step.

To measure ITT after step completion, the parameter α and β in Equation 4.1 are both set to 0.04, with time period of 50 ms and completion threshold of 1. Thus, the ITT



(a) (b) (c) (d) (e) (f)

(g) (h)

Figure 5.6: Sample items in the testing set

Class	Item	Count	Correct	Recall	P_f	P_u	$P_f - P_u$	**ITT** (s)
Unfinished	(a)	17	12	0.71	0.31	0.69	-0.38	-
	(b)	12	12	1.00	0.13	0.87	-0.74	-
	(c)	23	15	0.65	0.36	0.64	-0.28	-
	(d)	18	17	0.94	0.24	0.76	-0.52	-
	All	**70**	**56**	**0.80**				
Finished	(e)	17	9	0.53	0.58	0.42	0.16	7.81
	(f)	15	9	0.60	0.61	0.39	0.22	5.68
	(g)	25	22	0.88	0.82	0.18	0.64	1.95
	(h)	13	13	1.00	0.93	0.07	0.86	1.45
	All	**70**	**53**	**0.76**				

Table 5.3: Recall and instruction triggering time (ITT) on items in testing set

depends on P_f and P_u, which are the probability distribution over predicted output classes "Finished" and "Unfinished" respectively. After calculating average P_f and P_u over images of each item in Figure 5.6, ITT for each item can be estimated by $\frac{1}{0.04\times(P_f-P_u)} \times 0.05$.

Table 5.3 demonstrates testing results. In general, classification accuracy on the testing set is around 10% lower than the validation set, which might result from unique patterns in LEGO pieces that are not captured in the train/validation set (see Figure 4.11). The recall rate grouped by each item varies significantly, which could be affected by: (1) background or neighboring pieces. For example, the background of (a) with certain color or pattern looks like real objects inside the wireframe. (2) object size, usually the larger the more accurate. For instance, small object size (radius of 4mm) in (c) and (f) limits the features utilized by the classifier. (3) object color and pattern, such as transparency (e), which leads to inaccurate estimation of object-wireframe coincidence degree, and reduces classification performance. An object with patterns distinctive to its surroundings (b, h) generally demonstrates best recall rate.

It is observed that $|P_f - P_u|$ and recall rate correlates positively, and the estimated ITT for each "Finished" steps ranges from 1.45s to 7.81s, which can be further shortened by increasing parameter α and β, at the cost of resilience. Ideally, the ITT on "Unfinished" steps will be $+\infty$ since $P_f < P_u$ and the confidence score S will keep dropping until 0 (see Equation 4.1). In addition to recall rate, during experiment we observed that ITT is also affected by: (1) image registration accuracy and digital model fidelity, since state classification requires accurate object-wireframe alignment. (2) motion during instruction triggering, where S rises steadily in still (shorter ITT), and fluctuates in movement (longer ITT). (3) camera position relative to the assembly object, where ITT is shorter in certain perspectives and closer distance. After visually examining instruction automation for randomly selected 100 assembly steps under the same parameter setting, it is found that 28 finished steps with ITT < 3, 53 with $3 <$ ITT < 10, 19 with ITT > 10 where 8 fails to trigger, with a 92% triggering rate and an average ITT of 5.61. However, 16 "Unfinished" steps are kept misclassified with $P_f > P_u$, indicating a false-triggering rate of 16%.

In summary, the instruction automation system demonstrates good generalization (instruction triggering rate of 92% on 100 finished steps) under limited training samples (images of 10 assembly pieces out of 468), with an average triggering time of 5.61 second. However, the coincidence degree-based classification method has several limitations: (a)

lack of robustness in complex environments where the background has similar patterns to the target object. (b) dependence on other sub-systems, where registration accuracy and digital model fidelity are crucial factors. (c) poor performance on certain kinds of items, such as transparent pieces, which may fail to trigger the next instruction.

5.3 Limitations

Although the whole system demonstrates considerable performance and accuracy, and a fulfilment of system specifications (see Appendix C), there are several limitations in experiment design and the application, including:

Coupling between object classes. As discussed in Section 5.1.2, several sub-systems are not covered in integration testing because their dependence on other sub-systems. This could limits scalability, testability and maintainability. Thus, refactoring might be needed in future works to produce a loosely coupled system.

Lack of automation in UI testing. As discussed in Section 5.1.3, the image marker needs to be manually presented to initialize the instruction rendering system (IRS) during UI testing, which could be inconvenient in some cases. Thus, implementing specific gesture recognizer for IRS initialization might help.

Lack of testing samples. As shown in Section 5.2.2, only 4 images for registration accuracy and 2 for occlusion accuracy are evaluated, despite promising results, the accuracy may vary under different environments. More testing samples are required to provide more comprehensive and reliable results.

Lack of evaluation on other classifiers. As discussed in Section 4.2.4, MobileNetV2 [74] model is used and evaluated as the state classifier. However, other state-of-the-art classification models might be better in terms of accuracy and speed, which could be trained and evaluated in the future.

Lack of evaluation on parameter settings. As discussed in Section 5.2.2, the parameter settings on step scoring function (Equation 4.1) are kept consistent throughout the evaluation. Since theoretically the parameter α and β trades off instruction triggering speed and reliability, their clear relationship should be evaluated by measuring ITT and false-triggering rate under different parameter settings.

These limitations may result from insufficient project planning and timing, which will be discussed in Section 6.1 and 6.2, and considered in future works in Section 6.4.

Chapter 6

Conclusion

This chapter includes project management on time and resources, personal reflections on plan and experiences, contributions made through the process, and future directions.

6.1 Project Management

In the original plan, project development begins with researching on relevant AR techniques that are accessible and feasible to build the required application (Chapter 2). Then, a specific LEGO set will be chosen as the assembly target, and its 3D model will be constructed from scratch or downloaded from the internet (Section 4.1.1). Application development follows (Chapter 4), where most of the functionalities will be implemented, with development testing to ensure software quality (Section 5.1). System testing and evaluation (Section 5.2) will be conducted afterwards to validate system specification (Appendix C). Finally, the outcomes of all steps will be written in this dissertation.

The planned timeline is demonstrated in Figure 6.1 with several tasks and their corresponding objectives (Section 1.3). The green cells indicate software development tasks, and the rest are colored in blue. In general, the original plan follows Agile-Waterfall hybrid approach [79], where requirements gathering, design and implementation is structured with Waterfall, and each task is developed and tested in short sprints using Agile. Some of the tasks are conducted in parallel due to independence.

In actual timeline illustrated in Figure 6.2, application development and system evaluation are split into sub-tasks to reflect changes clearly. Compared with planned timeline, there is a significant increase of time in literature review, and additional time in application development at the beginning of second semester, due to the increase of project scope to include implementing IAS (Section 4.2.4). Most of tasks in the first semester progressed well within the timeline, however, with the deadline extension for Interim Report due

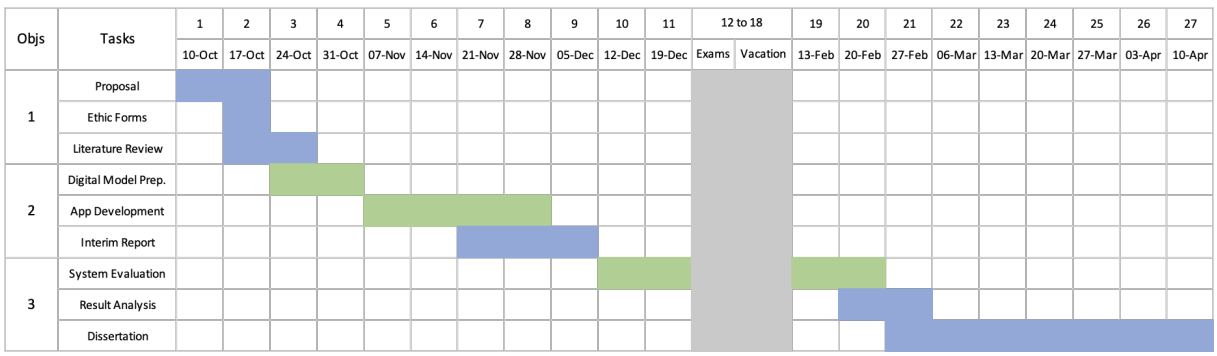


Figure 6.1: Planned timeline

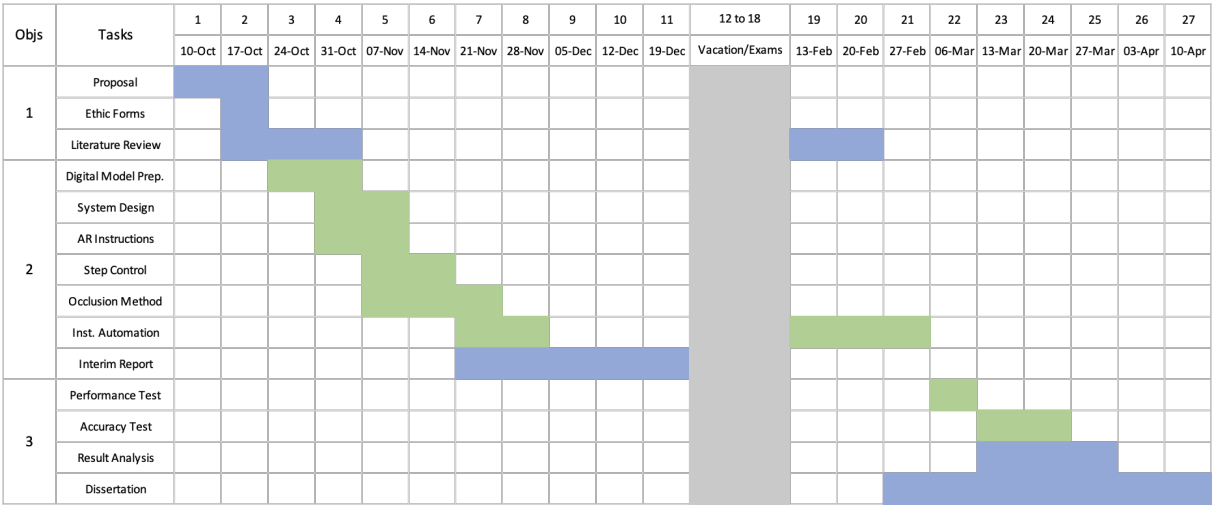


Figure 6.2: Actual timeline

to unexpected COVID incident, another two weeks were allocated to Interim Report, allowing flexible arrangement on time and resources under the pandemic.

As a result of additional time taken in background research and IAS implementation, system evaluation and result analysis were delayed for a few weeks. To mitigate the impact on subsequent tasks, they were conducted in parallel with dissertation writing on irrelevant parts. By adopting Agile with iterative development cycles, the requirement change was quickly adapted, and the influence on software quality was minimized. However, system evaluation process was shortened to ensure sufficient time for result analysis and report writing, which leads to limitations on several aspects, as discussed in Section 5.3.

6.2 Reflections

At the first stage of development, it was unclear whether to take this project as a research or software engineering work. Although a range of related works were identified with possible ways for improvement, there were several research directions in this project (Chapter 2) and it was hard to determine which one I was most interested in. After a few weeks of consideration, I decided to combine all relevant methods into a mobile application, to solve the challenging task in small parts assembly, simulated by the LEGO blocks.

Thus, although the main objective is to implement a mobile application following software engineering guidelines and code conventions (Appendix E), it is critical to improve relevant methodologies and design novel approaches if necessary, since studies on AR instructions for small part assembly is limited. Therefore, in addition to implementing basic functions in general AR assembly tools, some methods were optimized for small part assembly (see Section 6.3), which took significant time in literature review and system evaluation.

Various risks and challenges exist throughout the project, and the whole process went not as smoothly as expected. Due to the lack of experience in AR and mobile development, I learned Unity and C# in advance to save time for software development. However, during background research, Xcode (Swift) was found to be more suitable for the task (see Section 3.1.2), which I learned at the same time of literature review to minimize influence on following tasks. Apart from COVID incident as discussed in Section 6.1, another

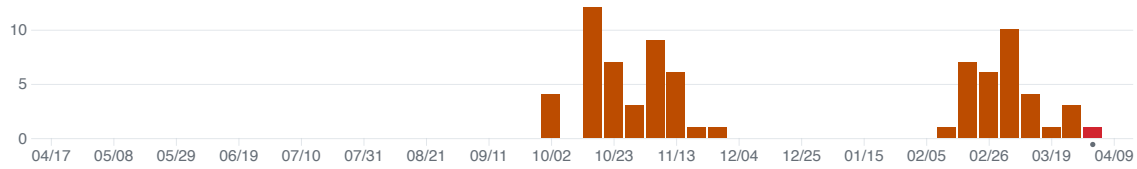


Figure 6.3: Git commit history

risk is the implementation of IAS (Section 4.2.4). The requirement on IAS was added halfway through software development to further demonstrate novelty and enhance user experience, which disrupted the original timeline and increased the difficulty level of the project. To reduce impact on the whole project, in addition to using Agile methodology, requirement prioritization (Section 3.2.3) was conducted to ensure critical functionality being implemented first, and it was decided that the new requirement will be revoked if the IAS fails to meet expectations before end of February. Although IAS was accepted at the end, the extension of software development period occupied time for system evaluation, and restricted testing comprehensiveness, as discussed in Section 6.1.

Despite scope change, the pace on project development was good and all deliverables were finished ahead of deadlines. The 76 git commits were evenly distributed over the software development period, as shown in Figure 6.3, indicating continuous contributions to the project. Through the experience, I learned that:

- Do sufficient background research before development, to avoid unnecessary work.
- Try not to change requirements halfway through the development, to prevent impact on subsequent tasks.
- Conducting independent tasks in parallel and using iterative development cycles are good practices in terms of efficiency and flexibility

To summarize, this project has been a valuable experience that broadened my knowledge in related fields and improved my skills in project planning, software development, critical thinking and technical writing.

6.3 Contributions

The main contribution in this project is the combination of several innovative methods that makes up an AR assembly guidance application, optimized for small parts assembly and validated by a complex LEGO set. Remarkable achievements include:

- A combination of several rendering options such as texture, wireframe and animated preview (Section 4.1.3), which improves visibility of model-based AR instructions.
- Accurate VIO-based registration on a flexible printed assembly base (Section 4.2.1), which achieves an average alignment error less than 1mm throughout the model.
- Algorithms for press-drag step control (Section 4.2.2), which provides a convenient and precise way for fast instruction adjustment.
- A fast and effective method to achieve precise object occlusion on the virtual content

(Section 4.2.3), utilizing high-fidelity digital model and virtual-physical alignment to generate occlusion masks in real-time.

- A smart instruction automation system (IAS, Section 4.2.4), combining an image cropper, state classifier and step scoring function that triggers the next AR instruction by detecting complete assembly steps, achieving 92% triggering rate on 100 finished assembly steps, trained on images of only 10 assembly pieces out of 468.
- A synthetic dataset consisting of 1000+ images on several LEGO blocks of two assembly states (Section 4.2.4), used for training/validating the IAS state classifier, and may be applied to further studies in instruction automation.

Specifically, the proposed method in IAS provides a efficient and robust solution to recognize complete steps in small part assembly. While it is challenging to detect small assembly pieces or identify hundreds of assembly states (Section 4.2.4), the IAS only focuses on two states - Finished (Integrated) and Unfinished (Non-integrated), based on coincidence degree between digital wireframe and object contours, achieving a high degree of generalization with less training samples (Section 5.2.2). In addition, the IAS utilizes image registration and digital modelling to extract region of interest to reduce influence from the background, and applies a step scoring function based on multiple frames to stabilize instruction triggering. As a simulation of industrial assembly, the selected LEGO set has validated the feasibility and reliability of proposed AR solution on small part assembly scenario with a large number of pieces.

6.4 Future Work

Future work may be built on following directions:

Software improvement, including (1) refactoring, to reduce coupling and improve maintainability (see Section 5.3), (2) UI improvements, such as adding a help panel, launch screen and accessibility options, and (3) improving IAS performance, such as enlarging the state-classification dataset, revising classifier architecture, training methods or step scoring function to achieve better generalization.

Further evaluation. As discussed in Section 5.3, more testing samples for accuracy measurement on image registration, object/hand occlusion and instruction automation are required. Additional evaluation could be: (1) 2-class assembly state classification accuracy on other advanced models or novel architectures. (2) instruction triggering time and false-triggering rate under different parameter settings in step scoring function (Equation 4.1), which helps find better parameters to balance the trade-off.

Method exploration. All methods have certain limitation and could be advanced. For example, the marker-based registration may be inconvenient for LEGO construction when the user needs to grasp parts and connect them in hand. Investigating more flexible yet accurate AR registration methods will continue to be important, such as marker-less registration powered by deep learning and depth imaging [80].

Application extension. This AR application could be easily extended to support other assembly tasks, such as block toys, instruments and furniture, if their high-fidelity digital models are available. In addition, the design and methodologies of the application can be applied to other AR devices like AR glasses in future development.

Appendix A

User Manual

The user manual introduces steps to install the application, launch relevant tests, and use it for LEGO assembly guidance. However, this application only runs in iOS (or equivalent version of iPadOS), and requires Xcode[1] for installation. In addition to hardware and software requirements listed in Section 3.1.3, Xcode version 14.0 and later is necessary.

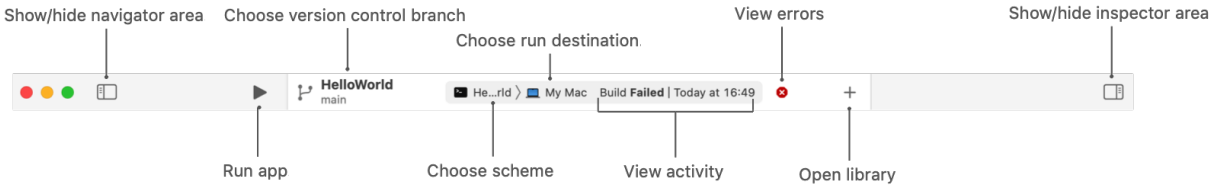


Figure A.1: Sample Xcode toolbar

A.1 Installation

In the project root directory, open `LEGOAssemblyGuide.xcodeproj` to launch the Xcode project with all necessary files included. Since ARKit is not supported in iOS simulator, you may need to connect real devices (iPhone or iPad) to your Mac by a cable. Unlock the device and confirm any pop-ups that appears in Xcode or on the device, also make sure device's developer mode[2] is enabled. Afterwards, choose run destination to your device, as illustrated in Figure A.1. You may also perform a few additional steps:

1. Specify your Apple ID in Account preferences in Xcode settings. Follow this post[3] if you do not have a developer account.
2. Specify a valid team in your project's Signing & Capabilities pane, as shown in Figure A.2 (content may vary).

After signing, you can click "Run app" button (see Figure A.1) to install and launch the application to your connected device. If you encounter "Untrusted Developer" warning, go to device Settings > General > VPN & Device Management (in iOS 16 and later), and select the profile to trust.

A.2 Testing

To run testing scripts including unit, integration and UI tests (Section 5.1), please finish installation as described in Section A.1 first. You will have several options to run the

[1] Xcode - Apple's Integrated Development Environment
[2] Apple Developer - Enabling Developer Mode on a Device
[3] Developer Support - Account Management

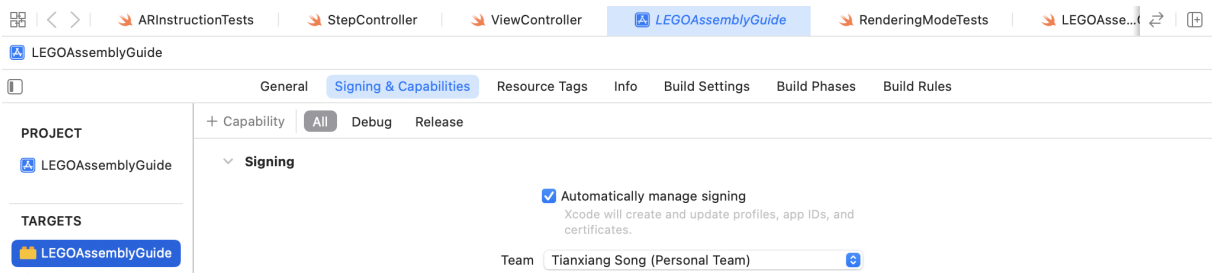


Figure A.2: Developer signing

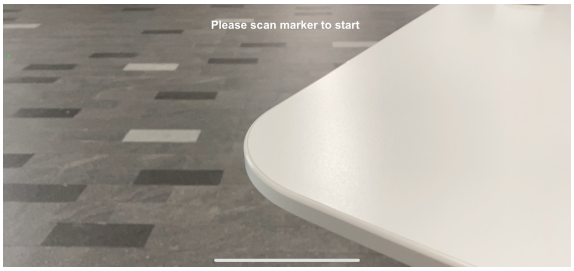
tests, following this post[4] for detailed instructions. Note that to perform UI testing, you will need to manually present the image marker for the system to change states, as shown in Figure A.3, though the rest of UI interactions are automated.

A.3 Usage

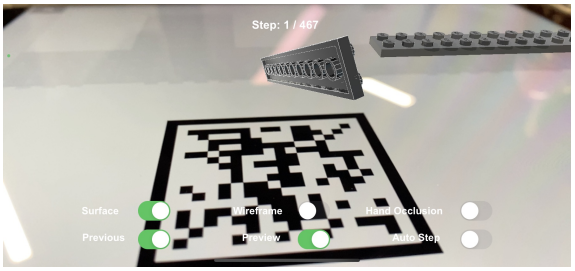
Once the application is successfully installed on your device, you can either launch it in Xcode (the last step in Section A.1) or click the app icon in your device. Note that for security reasons[5], the installed application may be expired after a few days. Please reinstall following Section A.1 if it expires.

To use the application for guided assembly, you may also need to prepare: (1) the printed assembly base in an A4 paper, as shown in Figure 4.5. The source file is included in the project (see Appendix E). (2) the LEGO set 21034[6], as illustrated in Appendix D.

After launching the application and confirming any pop-ups, the screen presents camera view with prompt "Please scan marker to start", as shown in Figure A.3 (a). When image marker in the assembly base is successfully detected, a set of functional switches and the first assembly instruction will appear, as shown in Figure A.3 (b). You may now pick corresponding LEGO block and place to where the virtual block appears. Following Section 4.2.2 to change instruction steps and Section 4.1.3 to set instruction rendering modes. In addition to enabling hand occlusion (Section 4.2.3) and instruction automation (Section 4.2.4) by corresponding switches, object occlusion (Section 4.2.3) can be enabled by switching `Surface`, `Wireframe` off and `Previous` on.



(a) Initial UI screen



(b) UI screen after detecting marker

Figure A.3: Two UI states in the application

[4] Apple Developer - Running Tests and Interpreting Results
[5] Developer Support - Provisioning Profile Updates
[6] London 21034 - Architecture

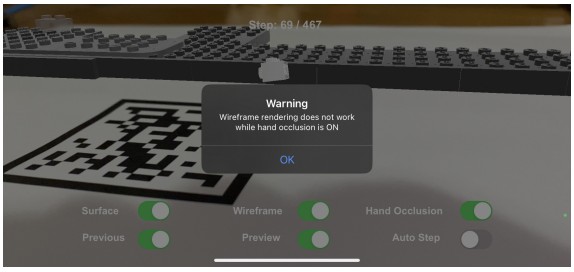
A.4 Troubleshooting

Considering different printing environments, the size of printed image marker may vary, which can cause small misalignment between digital model and corresponding physical parts. You can set size of the AR reference image to adjust size of the digital model.

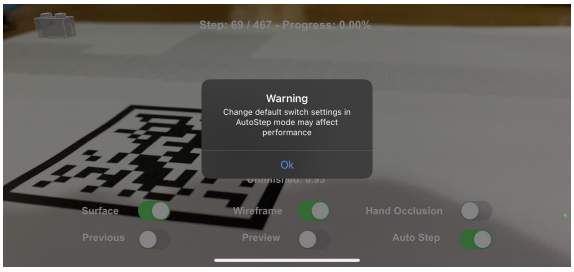
Find the image following the path shown at the top of Figure A.4, and change the width/height shown in the right side. Usually a ± 0.05 is sufficient, depending on the amount of alignment error. Note that increasing the size of reference image decreases the size of digital model, and vise versa. Afterwards, please reinstall following Section A.1.

Figure A.4: Adjust size of reference image

In addition, you may receive two warnings during the use of application as shown in Figure A.5. Due to the design of ARKit, digital wireframes cannot be rendered when hand occlusion (Section 4.2.3) is enabled. Since the state classification model is trained on synthetic images with "Wireframe" mode (Section 4.2.4), when enabling instruction automation (switch `Auto Step` on), other switch settings will be automatically changed, as shown in Figure A.5 (b). In this case, the instruction triggering performance will be affected severely by switching `Wireframe` off, mildly by switching `Previous` on, and negligibly by other switch changes.



(a) Warning for hand occlusion



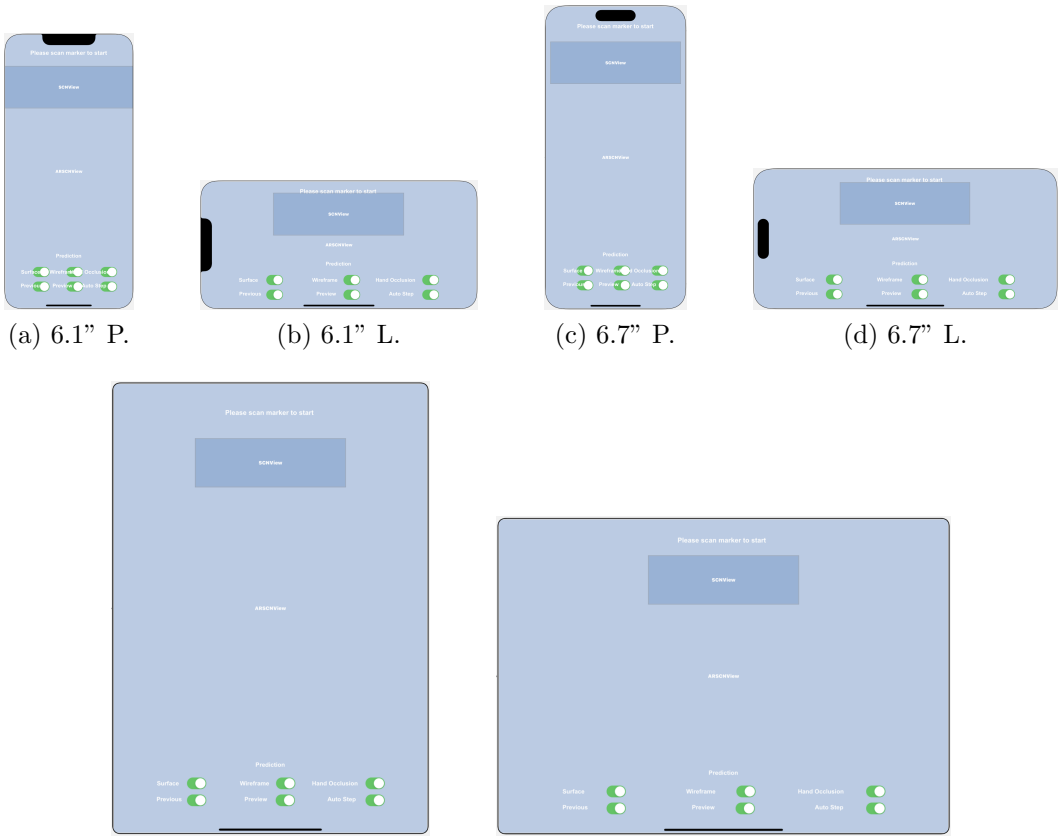
(b) Warning for instruction automation

Figure A.5: Two warnings in the application

Appendix B

Prototyping

The UI is designed to support both portrait and landscape modes in all devices capable of running this application. The prototype on mainstream screen sizes are shown as follows:



(a) 6.1" P. (b) 6.1" L. (c) 6.7" P. (d) 6.7" L.

(e) 11" P. (f) 11" L.

Figure B.1: UI prototypes on mainstream screen sizes

Following concept of responsive design [58], the layout of screen elements can automatically adapt to a wide range of sizes, including iPhone and iPad, without being too crowded or loose. All elements are not occluded by special areas (e.g., black regions in iPhone's screen). Although switches and labels overlap in iPhone's portrait mode (a, c), they can still be distinguished. You may check UI prototypes on other supported devices, by viewing `Main.storyboard` in Xcode, and selecting different models at the bottom.

Appendix C

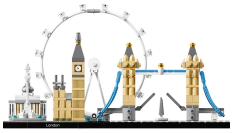
System Specification

Category	Ref.	ID	Description	Checked (tested) in
Compatibility	FR-1	1.1	The software could be installed and used on a device with iOS (iPadOS) 16.0 and later	Sec. 5.2.1
		1.2	The software could be developed using Xcode 14.0 and later	Appx. A
	FR-2	2.1	The software could be installed and used on a device with A12 processor and later	Sec. 5.2.1
		2.2	The software could be installed and used on a device with 3GB memory and larger	Sec. 5.2.1
Functionality	FR-3	3.1	The software shall access the rear camera of the device	Ch. 4, 5
		3.2	The software shall display a preview of the captured scene on the screen	Ch. 4, 5
	FR-4	4.1	The software shall include a database of 3D models of objects to be assembled	Sec. 4.2.2
		4.2	The software shall present corresponding digital models (AR instructions) of the physical part on each assembly step	Sec. 5.2
		4.3	The software shall display number of current assembly step and number of total steps	Sec. 4.1.2
	FR-5	5.1	The presented digital model shall be located at exact position of where the assembly piece should be	Sec. 4.2.1
		5.2	The presented digital model shall be visually identical to the corresponding physical part, in terms of size, shape, color and orientation	Sec. 4.1.1
		5.3	The system shall continuously align digital models with physical parts regardless of camera position, by tracking a specific image marker	Sec. 4.2.1, 5.2.2
	FR-6	6.1	The system shall provide 4 rendering options for digital models, including wireframe, texture, hide/unhide previous instructions and a preview of each assembly piece	Sec. 4.1.3
		6.2	The system shall allow users to switch among (or combine) different rendering options in 6.1, by several switches	Sec. 4.1.2, 5.1.2
	FR-7	7.1	The system shall be able to present digital models in correct spatial relationship with previously assembly parts (object occlusion) and user hands (hand occlusion)	Sec. 4.2.3, 5.2.2
		7.2	The system shall allow users to enable/disable object occlusion or hand occlusion, by several switches	Sec. 4.1.2
	FR-8	8.1	The system shall include gesture recognizers to identify different operations on the interface, including tap, press and drag	Sec. 5.1.2, 5.1.3
		8.2	The system shall allow users to change among different AR instructions based on 8.1	Sec. 4.2.2, 5.1.3
	FR-9	9.1	The system shall be able to determine the completion of each assembly step	Sec. 4.2.4, 5.2.2
		9.2	The system shall be able to present the next AR instruction automatically within a few seconds when current assembly step is finished	Sec. 5.2.2
Accessibility	NFR-1	10.1	The user interface should be informative and concise, where any button, icon or text fields should be clear and self-explanatory, with a text label for each switch.	Sec. 4.1.2
	NFR-2	11.1	User should be able to interact with the application without difficulty, where each operation should be easily performed, with a maximum of 2 fingers.	Sec. 4.1.3, 4.2.2
	NFR-3	12.1	The user interface should be intuitive and clean, with a maximum of 10 elements, in order to prevent distraction and maximize the AR view.	Sec. 4.1.2
Reliability	FR-5,	13.1	The average alignment error between virtual content and physical model should be less than 1mm	Sec. 5.2.2
	NFR-4	13.2	The object/hand occlusion (7.1) accuracy should be as higher as possible, with an average IoU $> 90\%$	Sec. 5.2.2
	FR-9	14.1	The instruction triggering time (ITT) in 9.2 should be as lower as possible, with an average ITT < 6s	Sec. 5.2.2
		14.2	The false-triggering rate (FTR) on unfinished steps (9.2) should be as lower as possible, with an average FTR < 0.2	Sec. 5.2.2
Performance	NFR-5	15.1	The software shall finish launching within 3 seconds, where the digital model finished loading and ready to be rendered	Sec. 5.2.1
	NFR-6	16.1	During the use of software, the average FPS should be greater than 60, including computational-intensive situations	Sec. 5.2.1

Appendix D

LEGO 21034 Inventory

The figures and tables in this appendix are collected from LEGO website[1] and BrickSet[2]. The inventory is validated with official assembly guide [66].



(a) Preview

(b) Package front

(c) Package back

Element	Image	Qty	Colour	Category	Design	Element name	Element in sets	Element introduced in	Design in sets	Design introduced in
243126		3	Black	Plates	2431	FLAT TILE 1X4	833	1994	5308	1991
244526		4	Black	Plates	2445	PLATE 2X12	347	1995	1324	1994
302126		3	Black	Plates	3021	PLATE 2X3	1039	1991	6918	1980
302226		2	Black	Plates	3022	PLATE 2X2	1290	1986	9201	1986
302326		12	Black	Plates	3023	PLATE 1X2	1603	1991	13404	1981
303526		1	Black	Plates	3035	PLATE 4X8	317	1993	1719	1992
306926		3	Black	Plates	3069	FLAT TILE 1X2	824	1991	7235	1991
346026		4	Black	Plates	3460	PLATE 1X8	598	1992	3250	1981
371026		5	Black	Plates	3710	PLATE 1X4	1309	1991	9367	1981
379526		2	Black	Plates	3795	PLATE 2X6	992	1994	6197	1980

[1] https://www.lego.com/en-us/product/london-21034
[2] https://brickset.com/inventories/21034-1

Element	Image	Qty	Colour	Category	Design	Element name	Element in sets	Element introduced in	Design in sets	Design introduced in
663626		3	Black	Plates	6636	FLAT TILE 1X6	505	1996	3208	1996
4653049		6	Black	Plates, Special	87609	PLATE 2X6X2/3 W 4 HOR. KNOB	61	2012	269	2010
6052126		1	Black	Plates, Special	99206	PLATE 2X2X2/3 W. 2. HOR. KNOB	285	2013	1295	2012
6174611		1	Black	Plates	29584	FLAT TILE 1X8, NO. 50	1	2017	1	2017
4121715		4	Black	Connectors	2780	CONNECTOR PEG W. FRICTION	2210	1993	2210	1993
4113917		4	Brick Yellow	Plates	3023	PLATE 1X2	744	1998	13404	1981
4114084		8	Brick Yellow	Plates	3022	PLATE 2X2	484	1998	9201	1986
4114309		3	Brick Yellow	Plates	3020	PLATE 2X4	509	1998	9965	1980
4118790		4	Brick Yellow	Plates	3021	PLATE 2X3	377	1999	6918	1980
4121921		2	Brick Yellow	Plates	3623	PLATE 1X3	292	1999	4899	1981
4125253		4	Brick Yellow	Plates	3070	FLAT TILE 1X1	203	1999	3772	1994
4159553		4	Brick Yellow	Plates	3024	PLATE 1X1	306	2001	5845	1986
4218749		20	Brick Yellow	Bricks	30136	PALISADE BRICK 1X2	153	2003	1270	1996
4224793		8	Brick Yellow	Decoration Elements	49668	PLATE 1X1 W/TOOTH	42	2004	684	2003
4612603		2	Brick Yellow	Frames, Windows, Walls And Doors	6231	WALL CORNER 1X1X1	40	2009	698	1995
4618651		4	Brick Yellow	Bricks With Bows And Arches	4490	BRICK W. BOW 1X3	44	2011	229	1994
4655900		8	Brick Yellow	Bricks	2877	PROFILE BRICK 1X2	32	2013	1259	1991
6046907		4	Brick Yellow	Decoration Elements	15208	1X2 PLATE WITH 3 TEETH	37	2014	276	2014
6092587		4	Brick Yellow	Plates	15573	PLATE 1X2 W. 1 KNOB	239	2010	3746	2002
6143058		8	Brick Yellow	Decoration Elements	15070	PLADE 1X1 M. 1 LOD. TAND	12	2016	306	2014

Element	Image	Qty	Colour	Category	Design	Element name	Element in sets	Element introduced in	Design in sets	Design introduced in
6175701		1	Brick Yellow	Bricks	29810	BRICK 2X2, NO. 103	1	2017	1	2017
4143005		1	Bright Blue	Connectors	4274	CONNECTOR PEG W. KNOB	947	2001	2171	1981
4211053		4	Dark Stone Grey	Plates	2431	FLAT TILE 1X4	454	2003	5308	1991
4211094		2	Dark Stone Grey	Plates	3022	PLATE 2X2	823	2002	9201	1986
4226221		1	Dark Stone Grey	Bricks, With Slope	3044	RIDGED TILE 1X2/45°	65	2004	146	1998
4521187		4	Dark Stone Grey	Plates, Special	60478	PLATE 1X2 W/SHAFT Ø3.2	346	2008	1683	2008
4529240		1	Dark Stone Grey	Bricks, Special Circles And Angles	59900	NOSE CONE SMALL 1X1	247	2005	2484	2002
4566028		4	Dark Stone Grey	Figure, Weapons	86208	BUTT	59	2007	106	2005
4645099		3	Dark Stone Grey	Bricks, With Slope	3688	PYRAMID BRICK 2X2X2/73°	16	2012	59	2000
6000606		4	Dark Stone Grey	Plates, Special	99780	ANGULAR PLATE 1.5 BOT. 1X2 1/2	462	2012	1906	2012
6076203		1	Dark Stone Grey	Bricks, With Slope	15571	END RIDGED TILE 1X2/45°	59	2012	416	2008
6092572		8	Dark Stone Grey	Plates	15573	PLATE 1X2 W. 1 KNOB	359	2010	3746	2002
6174938		1	Dark Stone Grey	Bricks, With Slope	3049	ATTIC 1X2/45°	12	2017	172	1986
6331689		2	Dark Stone Grey	Plates, Special	65886	PLATE 2X3 W. HOLDER	17	2017	167	2016
4168345		4	Medium Blue	Plates	3069	FLAT TILE 1X2	87	2002	7235	1991
4527526		4	Medium Blue	Plates	3070	FLAT TILE 1X1	29	2008	3772	1994
6092601		2	Medium Blue	Plates	15573	PLATE 1X2 W. 1 KNOB	26	2015	3746	2002
6366178		4	Medium Blue	Connectors	49755	3.2 SHAFT W/3.2 HOLE	7	2017	293	2016
6178243		4	Medium Blue	Tubes	32580	FLEX ROD 7M	1	2017	50	2001
4211353		1	Medium Stone Grey	Plates	2420	CORNER PLATE 1X2X2	410	2001	3727	1991

Element	Image	Qty	Colour	Category	Design	Element name	Element in sets	Element introduced in	Design in sets	Design introduced in
4211356		4	Medium Stone Grey	Plates	2431	FLAT TILE 1X4	576	2002	5308	1991
4211395		4	Medium Stone Grey	Plates	3020	PLATE 2X4	1004	2001	9965	1980
4211397		8	Medium Stone Grey	Plates	3022	PLATE 2X2	940	1998	9201	1986
4211398		7	Medium Stone Grey	Plates	3023	PLATE 1X2	1187	1993	13404	1981
4211414		9	Medium Stone Grey	Plates	3069	FLAT TILE 1X2	598	2003	7235	1991
4211415		7	Medium Stone Grey	Plates	3070	FLAT TILE 1X1	362	2004	3772	1994
4211452		3	Medium Stone Grey	Plates	3795	PLATE 2X6	653	2001	6197	1980
4654580		4	Medium Stone Grey	Plates, Special	99207	ANGULAR PLATE 1.5 BOT. 1X2 2/2	523	2012	1567	2012
4654582		4	Medium Stone Grey	Plates, Special	99781	ANGULAR PLATE 1,5 TOP 1X2 1/2	435	2012	1328	2012
6066097		6	Medium Stone Grey	Plates	15573	PLATE 1X2 W. 1 KNOB	646	2002	3746	2002
6093527		1	Medium Stone Grey	Signs, Flags And Poles	17715	SHAFT 3M Ø3.2	164	2014	853	2012
6126082		2	Medium Stone Grey	Plates	23893	PLATE 2X2 W 1 KNOB	303	2013	1511	2010
6150310		4	Medium Stone Grey	Bricks With Bows And Arches	30165	BRICK 2X2 W. BOW AND KNOBS	12	2016	385	1998
6168633		4	Medium Stone Grey	Plates, Special	28809	PLATE 1x2 W. HORIZONTAL HOLE Ø4,85 REV.	241	2015	476	2015
6176433		4	Medium Stone Grey	Plates, Special Circles And Angles	25269	1/4 CIRCLE TILE 1X1	132	2017	1872	2016
6270081		1	Medium Stone Grey	Animals And Creatures	64727	DORSAL FIN Ø3.2 MM	16	2014	128	2009
6278156		4	Medium Stone Grey	Connectors	41005	STICK Ø 3.2 W. HOLDER	142	2016	501	2014
4211807		4	Medium Stone Grey	Connectors	3673	CONNECTOR PEG	1092	1993	1301	1981
6299492		1	Multicombination	Figure Accessories In Hand	53017	MINI FIGURE TROPHY	21	2016	79	2016
6284595		3	Olive Green	Plates, Special Circles And Angles	35381	FLAT TILE 1X1, ROUND	18	2016	1405	2014

Element	Image	Qty	Colour	Category	Design	Element name	Element in sets	Element introduced in	Design in sets	Design introduced in
6102756		1	Silver Metallic	Bricks, Special Circles And Angles	15395	DOME 2X2, INVERTED W. ONE STUD	40	2015	350	2014
6240204		23	Transparent	Plates	28653	PLATE 1X2	169	2010	957	2010
6244904		2	Transparent	Bricks	35743	BRICK 1X2 W/O PIN	131	2010	417	2010
6251294		40	Transparent	Plates	35386	FLAT TILE 1X2	70	2014	482	2011
6274739		4	Transparent Light Blue	Plates, Special Circles And Angles	35380	FLAT TILE 1X1, ROUND	113	2015	1198	2013
4523159		5	Warm Gold	Plates, Special Circles And Angles	6141	PLATE 1X1 ROUND	462	2008	7204	1990
241201		6	White	Plates	2412	RADIATOR GRILLE 1X2	374	1994	4484	1991
302301		9	White	Plates	3023	PLATE 1X2	1462	1992	13404	1981
302401		7	White	Plates	3024	PLATE 1X1	784	1986	5845	1986
307001		3	White	Plates	3070	FLAT TILE 1X1	477	1994	3772	1994
379401		4	White	Plates	3794	PLATE 1X2 W. 1 KNOB	766	1992	2511	1991
407001		6	White	Bricks, Special	4070	ANGULAR BRICK 1X1	348	1994	2500	1991
447701		2	White	Plates	4477	PLATE 1X10	245	1993	1807	1991
654101		4	White	Bricks, Special Ø4.85 Hole And Connecting Bush	6541	TECHNIC BRICK 1X1	160	1994	1521	1994
6023804		2	White	Decoration Elements	61409	ROOF TILE W. LATTICE 1x2x2/3	85	2013	1192	2008
6046905		2	White	Decoration Elements	15070	PLADE 1X1 M. 1 LOD. TAND	113	2014	306	2014
6057414		4	White	Plates, Special	11458	PLATE 1X2 W. HOR. HOLE Ø 4.8	107	2014	721	2013
6058177		4	White	Bricks, Special	11211	BRICK 1X2 W. 2 KNOBS	258	2014	1689	2012

Element	Image	Qty	Colour	Category	Design	Element name	Element in sets	Element introduced in	Design in sets	Design introduced in
6093053		3	White	Plates, Special Circles And Angles	18674	PLATE ROUND W. 1 KNOB	291	2015	1273	2015
6099412		1	White	Plates	15397	PLATE 3X3, CROSS	24	2015	135	2014
6116602		3	White	Signs, Flags And Poles	21462	LIGHT SWORD - BLADE	93	2010	901	2008
6156667		1	White	Plates	26603	FLAT TILE 2X3	165	2016	877	2016
6168642		3	White	Plates, Special Circles And Angles	28626	PL.ROUND 1X1 W. THROUGHG. HOLE	231	2015	1579	2014
6347293		4	White	Plates, Special	52738	PLATE 1X1 W/HOLDER VERTICAL	65	2017	588	2010
6348055		36	White	Plates, Special	44842	PLATE 1X1 W. UP RIGHT HOLDER	120	2014	1085	2010
6178597		4	White	Tubes	85526	OUTER CABLE 192 MM	1	2017	5	2009
6261390		2	White	Connectors	42128	ANGLE ELEMENT, 157,5 DEGR. [3]	12	2017	188	2009
6293820		2	White	Beams, Special	65450	LT SUSPENSION	1	2017	7	2017

Colour summary

Colour	Unique	Total
Black	15	54
Brick Yellow	16	88
Bright Blue	1	1
Dark Stone Grey	12	35
Medium Blue	5	18
Medium Stone Grey	19	81
Multicombination	1	1
Olive Green	1	3
Silver Metallic	1	1
Transparent	3	65
Transparent Light Blue	1	4
Warm Gold	1	5
White	22	112
Total	**98**	**468**

Category summary

Category	Unique	Total
Animals And Creatures	1	1
Beams, Special	1	2
Bricks	4	31
Bricks With Bows And Arches	2	8
Bricks, Special	2	10
Bricks, Special Circles And Angles	2	2
Bricks, Special Ø4.85 Hole And Connecting Bush	1	4
Bricks, With Slope	4	6
Connectors	6	19
Decoration Elements	5	24
Figure Accessories In Hand	1	1
Figure, Weapons	1	4
Frames, Windows, Walls And Doors	1	2
Plates	46	247
Plates, Special	11	73
Plates, Special Circles And Angles	6	22
Signs, Flags And Poles	2	4
Tubes	2	8
Total	**98**	**468**

Appendix E

Additional Documentation

E.1 Project File Structure

In project root directory, the folder hierarchy is as follows:

- `LEGOAssemblyGuide` - main files for the application
  - `Assets` - including UI Markup Language, AR reference image (see 4.2.1), digital LEGO model (see 4.1.1), CNN model (see 4.2.4) and app icons
  - `Data` - dataset for assembly state classification (see 4.2.4), including source code for CNN training and testing (`train.ipynb`)
  - `Sources` - source code of the main application
- `LEGOAssemblyGuideDocs` - files for HTML documentation, see E.2
- `LEGOAssemblyGuideIntegrationTests` - code for integration testing, see 5.1.2
- `LEGOAssemblyGuideUITests` - code for UI testing, see 5.1.3
- `LEGOAssemblyGuideUnitTests` - code for unit testing, see 5.1.1

In addition, you can find two supplementary PDF files:

- `AssemblyBaseA4` - print it in A4 paper as assembly base with image marker
- `AssemblyGuideBook` - official assembly guide for LEGO 21034

E.2 Software Documentation

There are three ways to view the software documentation:

1. Open `LEGOAssemblyGuide.doccarchive` in project root with Xcode
2. Visit `https://stx666michael.github.io/` in a web browser
3. Direct to `LEGOAssemblyGuideDocs/documentation/legoassemblyguide` and open `index.html` in a web browser

E.3 Code Conventions

A set of code conventions applies following Swift Style Guide[1].

[1] `https://google.github.io/swift/`

References

[1] Mathilde Drouot et al. "The visual impact of augmented reality during an assembly task". In: *Displays* 66 (Jan. 2021), p. 101987. DOI: 10.1016/j.displa.2021.101987.

[2] Antonio Uva et al. "Evaluating the effectiveness of spatial augmented reality in smart manufacturing: a solution for manual working stations". In: *The International Journal of Advanced Manufacturing Technology* 94 (Jan. 2018). DOI: 10.1007/s00170-017-0846-4.

[3] Gabriel Evans et al. "Evaluating the Microsoft HoloLens through an augmented reality assembly application". In: May 2017, p. 101970V. DOI: 10.1117/12.2262626.

[4] W. Friedrich. "ARVIKA-augmented reality for development, production and service". In: *Proceedings. International Symposium on Mixed and Augmented Reality*. 2002, pp. 3–4. DOI: 10.1109/ISMAR.2002.1115059.

[5] Kimberly Weaver et al. "An empirical task analysis of warehouse order picking using head-mounted displays". In: vol. 3. Jan. 2010, pp. 1695–1704. DOI: 10.1145/1753326.1753580.

[6] Dušan Tatić and Bojan Tešić. "The application of augmented reality technologies for the improvement of occupational safety in an industrial environment". In: *Computers in Industry* 85 (Feb. 2017), pp. 1–10. DOI: 10.1016/j.compind.2016.11.004.

[7] Trevor Richardson et al. "Fusing Self-Reported and Sensor Data from Mixed-Reality Training". In: 2014.

[8] Sameer Alnajdi, Malek Alrashidi, and Khalid Almohammadi. "The effectiveness of using augmented reality (AR) on assembling and exploring educational mobile robot in pedagogical virtual machine (PVM)". In: *Interactive Learning Environments* 28 (Nov. 2020), pp. 964–990. DOI: 10.1080/10494820.2018.1552873.

[9] Robin Hanson, William Falkenström, and Mikael Miettinen. "Augmented reality as a means of conveying picking information in kit preparation for mixed-model assembly". In: *Computers & Industrial Engineering* 113 (2017), pp. 570–575. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2017.09.048. URL: https://www.sciencedirect.com/science/article/pii/S0360835217304655.

[10] Sasan Sattarpanah Karganroudi et al. "A novel assembly process guidance using augmented reality for a standalone hybrid energy system". In: *The International Journal of Advanced Manufacturing Technology* 122 (Sept. 2022), pp. 1–21. DOI: 10.1007/s00170-022-10122-5.

[11] Michela Mura and G. Dini. "Augmented Reality in Assembly Systems: State of the Art and Future Perspectives". In: Apr. 2021, pp. 3–22. ISBN: 978-3-030-72631-7. DOI: 10.1007/978-3-030-72632-4_1.

[12] Giles Westerfield, Antonija Mitrovic, and Mark Billinghurst. "Intelligent Augmented Reality Training for Motherboard Assembly". In: *International Journal of Artificial Intelligence in Education* 25 (Mar. 2015). DOI: 10.1007/s40593-014-0032-x.

[13] K. Baird and Woodrow Barfield. "Evaluating the effectiveness of augmented reality displays for a manual assembly task". In: *Virtual Reality* 4 (Dec. 1999), pp. 250–259. DOI: 10.1007/BF01421808.

[14] Chih-Hsing Chu and Ching-Hung Ko. "An experimental study on augmented reality assisted manual assembly with occluded components". In: *Journal of Manufacturing Systems* 61 (Apr. 2021). DOI: 10.1016/j.jmsy.2021.04.003.

[15] Peter Plapper et al. "Augmented Reality in Manual Assembly Processes". In: Sept. 2020.

[16] Melynda Hoover et al. "Measuring the Performance Impact of Using the Microsoft HoloLens 1 to Provide Guided Assembly Work Instructions". In: *Journal of Computing and Information Science in Engineering* 20 (Jan. 2020), pp. 1–28. DOI: 10.1115/1.4046006.

[17] Wei Yan. "Augmented reality instructions for construction toys enabled by accurate model registration and realistic object/hand occlusions". In: *Virtual Reality* 26 (June 2022), pp. 1–14. DOI: 10.1007/s10055-021-00582-7.

[18] Jonas Blattgerste et al. "In-Situ Instructions Exceed Side-by-Side Instructions in Augmented Reality Assisted Assembly". In: June 2018, pp. 133–140. DOI: 10.1145/3197768.3197778.

[19] Wang Li et al. "Real-time occlusion handling for augmented reality assistance assembly systems with monocular images". In: *Journal of Manufacturing Systems* 62 (2022), pp. 561–574. ISSN: 0278-6125. DOI: https://doi.org/10.1016/j.jmsy.2022.01.012. URL: https://www.sciencedirect.com/science/article/pii/S0278612522000139.

[20] Yin Xuyue et al. "An Automatic Interaction Method Using Part Recognition Based on Deep Network for Augmented Reality Assembly Guidance". In: Aug. 2018, V01BT02A018. DOI: 10.1115/DETC2018-85810.

[21] Ronald T Azuma. "A survey of augmented reality". In: *Presence: Teleoperators and Virtual Environments* 6.4 (1997), pp. 355–385.

[22] Sebastian Büttner, Michael Prilla, and Carsten Röcker. "Augmented Reality Training for Industrial Assembly Work - Are Projection-based AR Assistive Systems an Appropriate Tool for Assembly Training?" In: Apr. 2020, pp. 1–12. DOI: 10.1145/3313831.3376720.

[23] Kamal Oudrhiri and Henry Fuchs. "Enhancing the Realism of Augmented and Mixed Reality Applications through Optical See-Through Head-Mounted Displays". In: *2015 IEEE Virtual Reality (VR)*. IEEE. 2015, pp. 323–324.

[24] Ke Wang et al. "A fast object registration method for augmented reality assembly with simultaneous determination of multiple 2D-3D correspondences". In: *Robotics and Computer-Integrated Manufacturing* 63 (2020), p. 101890. ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2019.101890. URL: https://www.sciencedirect.com/science/article/pii/S0736584519301644.

[25] Eleonora Bottani and Giuseppe Vignali. "Augmented reality technology in the manufacturing industry: A review of the last decade". In: *IIE Transactions* 51 (Mar. 2019), pp. 284–310. DOI: 10.1080/24725854.2018.1493244.

[26] Xiaowei Li et al. "Computing homography with RANSAC algorithm: a novel method of registration". In: *Electronic Imaging and Multimedia Technology IV*. Vol. 5637. SPIE. 2005, pp. 109–112.

[27] Viet-Toan Truong, Jhih-Siang Lao, and Ching-Chun Huang. "Multi-camera Marker-based Real-time Head Pose Estimation System". In: *2020 International Conference on Multimedia Analysis and Pattern Recognition (MAPR)*. 2020, pp. 1–6. DOI: 10.1109/MAPR49794.2020.9237775.

[28] David Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* 60 (Nov. 2004), pp. 91–. DOI: 10.1023/B:VISI.0000029664.99615.94.

[29] Daniel Wagner et al. “Real-Time Detection and Tracking for Augmented Reality on Mobile Phones”. In: *IEEE Transactions on Visualization and Computer Graphics* 16.3 (2010), pp. 355–368. DOI: 10.1109/TVCG.2009.99.

[30] Donald D Hoffman et al. “Vergence–accommodation conflicts hinder visual performance and cause visual fatigue”. In: *Journal of Vision* 8.3 (2008), pp. 33–33.

[31] Ronald T Azuma. “Recent advances in augmented reality”. In: *Proceedings of the 2001 IEEE and ACM international symposium on Augmented reality*. IEEE. 2001, pp. 197–206.

[32] Anastacia MacAllister et al. “Comparing Visual Assembly Aids for Augmented Reality Work Instructions”. In: 2017.

[33] Julien Valentin et al. “Depth from motion for smartphone AR”. In: *ACM Transactions on Graphics* (2018). URL: https://dl.acm.org/citation.cfm?id=3275041.

[34] Xiao Tang et al. “GrabAR: Occlusion-aware Grabbing Virtual Objects in AR”. In: *Graphics* (2019). DOI: 10.48550/ARXIV.1912.10637. URL: https://arxiv.org/abs/1912.10637.

[35] Wendy Chun and Tobias Höllerer. “Real-time hand interaction for augmented reality on mobile phones”. In: Mar. 2013, pp. 307–314. ISBN: 9781450319652. DOI: 10.1145/2449396.2449435.

[36] Junyeong Choi et al. “iHand: An interactive bare-hand-based augmented reality interface on commercial mobile phones”. In: *Optical Engineering* 52 (Feb. 2013), pp. 7206–. DOI: 10.1117/1.OE.52.2.027206.

[37] Jie Song et al. “Joint Estimation of 3D Hand Position and Gestures from Monocular Video for Mobile Interaction”. In: Apr. 2015, pp. 3657–3660. DOI: 10.1145/2702123.2702601.

[38] Tsung-Han Tsai and Shih-An Huang. “Refined U-Net: A New Semantic Technique on Hand Segmentation”. In: *Neurocomputing* 495 (Apr. 2022). DOI: 10.1016/j.neucom.2022.04.079.

[39] S.Amin Dadgar and Guido Brunnett. “Hand Segmentation with Mask-RCNN Using Mainly Synthetic Images as Training Sets and Repetitive Training Strategy”. In: Feb. 2023, pp. 220–228. DOI: 10.5220/0011658900003417.

[40] Liang-Chieh Chen et al. *Rethinking Atrous Convolution for Semantic Image Segmentation*. 2017. arXiv: 1706.05587 [cs.CV].

[41] Wenqi Jia, Miao Liu, and James M. Rehg. “Generative Adversarial Network for Future Hand Segmentation from Egocentric Video”. In: *Computer Vision and Pattern Recognition* (2022). DOI: 10.48550/ARXIV.2203.11305. URL: https://arxiv.org/abs/2203.11305.

[42] Alexander Neb and Florian Strieg. “Generation of AR-enhanced Assembly Instructions based on Assembly Features”. In: *Procedia CIRP* 72 (2018). 51st CIRP Conference on Manufacturing Systems, pp. 1118–1123. ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2018.03.210. URL: https://www.sciencedirect.com/science/article/pii/S2212827118303706.

[43] Yongzhi Su et al. “Deep Multi-State Object Pose Estimation for Augmented Reality Assembly”. In: Aug. 2019. DOI: 10.1109/ISMAR-Adjunct.2019.00-42.

[44] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: (June 2015).

[45] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. In: vol. 9905. Oct. 2016, pp. 21–37. ISBN: 978-3-319-46447-3. DOI: 10.1007/978-3-319-46448-0_2.
[46] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (June 2015). DOI: 10.1109/TPAMI.2016.2577031.
[47] Kaiming He et al. “Mask R-CNN”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP (June 2018), pp. 1–1. DOI: 10.1109/TPAMI.2018.2844175.
[48] Thanh-Toan Do et al. “LieNet: Real-time Monocular Object Instance 6D Pose Estimation”. In: *British Machine Vision Conference*. 2018.
[49] Yu Xiang et al. “PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes”. In: *Robotics: Science and Systems* abs/1711.00199 (2017).
[50] Alia Rukubayihunga, Jean-Yves Didier, and Samir Otmane. “Towards assembly steps recognition in augmented reality”. In: Mar. 2016, pp. 1–5. DOI: 10.1145/2927929.2927953.
[51] Zainab Oufqir, Abdellatif El Abderrahmani, and Khalid Satori. “ARKit and ARCore in serve to augmented reality”. In: *2020 International Conference on Intelligent Systems and Computer Vision (ISCV)*. 2020, pp. 1–7. DOI: 10.1109/ISCV49265.2020.9204243.
[52] Harmony. *Apple vs Google: AR Kit vs AR Core*. https://www.harmony.co.uk/apple-vs-google-augmented-reality/. Accessed: 2023-03-18.
[53] CGIFurniture. *Augmented Reality for Furniture Shopping: Android vs iOS*. https://cgifurniture.com/agmented-reality-for-furniture-shopping-android-vs-ios/. Accessed: 2023-03-18.
[54] Apple Inc. *Xcode*. https://developer.apple.com/xcode/. Accessed: 2022-11-21.
[55] Unity. *AR Foundation*. https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.0/manual/index.html. Accessed: 2023-03-18.
[56] “Model-View-Controller Pattern”. In: *Learn Objective-C for Java Developers*. Berkeley, CA: Apress, 2009, pp. 353–402. ISBN: 978-1-4302-2370-2. DOI: 10.1007/978-1-4302-2370-2_20. URL: https://doi.org/10.1007/978-1-4302-2370-2_20.
[57] Legolizer. *LEGO 21034 Architecture London digital model*. https://www.turbosquid.com/3d-models/london-lego-model-1274878. Accessed: 2022-11-21.
[58] Cheri Mullins. “Responsive, Mobile App, Mobile First: Untangling the UX Design Web in Practical Experience”. In: *Proceedings of the 33rd Annual International Conference on the Design of Communication*. SIGDOC ’15. New York, NY, USA: Association for Computing Machinery, 2015. ISBN: 9781450336482. DOI: 10.1145/2775441.2775478. URL: https://doi.org/10.1145/2775441.2775478.
[59] Martin Hirzer. *Marker Detection for Augmented Reality Applications*. Oct. 2008.
[60] Pattern Monster. *Customizable SVG patterns for your projects*. https://pattern.monster/. Accessed: 2022-11-21.
[61] Brosvision. *AUGMENTED REALITY MARKER GENERATOR*. https://www.brosvision.com/ar-marker-generator/. Accessed: 2022-11-21.
[62] Shawn Lehner. *ARMaker*. https://shawnlehner.github.io/ARMaker/. Accessed: 2022-11-21.
[63] Google Developer. *The arcoreimg tool*. https://developers.google.com/ar/develop/augmented-images/arcoreimg/. Accessed: 2022-11-21.

[64] Trung Nguyen et al. “CKF-Based Visual Inertial Odometry for Long-Term Trajectory Operations”. In: *Journal of Robotics* 2020 (June 2020), pp. 1–14. DOI: 10.1155/2020/7362952.

[65] Apple Developer. *Understanding World Tracking*. https://developer.apple.com/documentation/arkit/configuration_objects/understanding_world_tracking. Accessed: 2022-11-21.

[66] LEGO Group. *LEGO 21034 Construction Guide*. https://www.lego.com/cdn/product-assets/product.bi.core.pdf/6199019.pdf. Accessed: 2023-03-18.

[67] Joonho Chang and Kihyo Jung. “Development of a press and drag method for hyperlink selection on smartphones”. In: *Applied Ergonomics* 65 (2017), pp. 269–276. ISSN: 0003-6870. DOI: https://doi.org/10.1016/j.apergo.2017.07.001. URL: https://www.sciencedirect.com/science/article/pii/S0003687017301552.

[68] Liang-Chieh Chen et al. *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*. 2018. arXiv: 1802.02611 [cs.CV].

[69] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[70] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. “Segnet: A deep convolutional encoder-decoder architecture for image segmentation”. In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.

[71] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 740–755.

[72] Jayven Nhan. “People Occlusion”. In: *Mastering ARKit: Apple’s Augmented Reality App Development Platform*. Berkeley, CA: Apress, 2022, pp. 435–445. DOI: 10.1007/978-1-4842-7836-9_22. URL: https://doi.org/10.1007/978-1-4842-7836-9_22.

[73] Apple Developer. *Capturing Photos with Depth*. https://developer.apple.com/documentation/avfoundation/additional_data_capture/capturing_photos_with_depth. Accessed: 2023-03-18.

[74] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV].

[75] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[76] I. Sommerville. *Software Engineering*. International Computer Science Series. Pearson, 2011. ISBN: 9780137053469. URL: https://books.google.com.sg/books?id=l0egcQAACAAJ.

[77] Apple Developer. *XCTest*. https://developer.apple.com/documentation/xctest. Accessed: 2022-11-21.

[78] Apple Inc. *Instruments*. https://help.apple.com/instruments/mac/10.0/. Accessed: 2023-04-01.

[79] S. Balaji and M. Sundararajan Murugaiyan. “Waterfall vs. V-Model vs. Agile: A comparative study on SDLC”. In: 2012.

[80] He Liu and Ferdinando Rodriguez Y. Baena. “Automatic Markerless Registration and Tracking of the Bone for Computer-Assisted Orthopaedic Surgery”. In: *IEEE Access* 8 (2020), pp. 42010–42020.